

# The Proceedings of EuroPLoP 2009



## 14th annual European Conference on Pattern Languages of Programming

Irsee, Germany, July 8-12, 2009

Edited by:

- Allan Kelly, Software Strategy Ltd.
- Michael Weiss, Carleton University, Department of Systems and Computer Engineering, Ottawa, Canada

## Table of Contents

- [Preface](#)

### Workshop A "Alexander"

Workshop Leader: **Andreas Fießler**

1. [Junkies Like Us -- Finding Your Way Through The Collaborative Web \(not submitted for proceedings\)](#)  
Andreas Rüping
2. [Enterprise Architecture Management Patterns for Enterprise Architecture Visioning](#)  
Sabine Buckl, Alexander M. Ernst, Florian Matthes, Christian M. Schweda
3. [Roles in a Software Project](#)  
Andreas Fießler
4. [Applied Pattern for Strategy Management of Technology Entrepreneurship and Innovation MSc Program](#)  
Yanka Todorova, Petko Ruskov, Elissaveta Gourova, Mark Harris
5. [Performance of Open Source Projects](#)  
Michael Weiss

### Workshop B "Buckminster Fuller"

Workshop Leaders: **Uwe Zdun, Paris Avgeriou and Neil Harrison**

1. [The Role of Analysis Patterns in Systems Analysis](#)  
Anna Bobkowska, Jakub Grabowski
2. [A service for software pattern selection](#)  
Aliaksandr Birukou, Michael Weiss
3. [Applying Architectural Patterns for Parallel Programming: Solving the One-dimensional Heat Equation](#)  
Jorge L. Ortega-Arjona
4. [Towards Formalized Adaptation Patterns for Adaptive Interactive Systems](#)  
Matthias Bezold
5. [A Pattern driven Approach against Architectural Knowledge Vaporization](#)  
Uwe van Heesch, Paris Avgeriou
6. [Reusable Architectural Decisions for DSL Design: Foundational Decisions in DSL Projects](#)  
Uwe Zdun, Mark Strembeck

### Workshop C "Le Corbusier"

Workshop Leader: **Allan Kelly**

1. [A Pattern Vocabulary for Product Distribution](#)  
Allan Kelly
2. [Business Patterns for Knowledge audit implementation](#)  
Albena Antonova, Elissaveta Gourova
3. [Applying Distributed Development Patterns](#)  
Lise B. Hvatum
4. [Business Plan Conception Pattern Language](#)  
Wim Laurier, Pavel Hruby, Geert Poels
5. [Software Design Reviews](#)  
Nora Ludewig



## Workshop D "DaVinci"

Workshop Leader: **Christian Kohls**

1. [Design Patterns for Creativity](#)  
Petros Georgiakakis, Symeon Retalis
2. [PLITS:A Pattern Language for Intelligent Tutoring Systems](#)  
Dina Salah, Amir Zeid  
[Adaptive Patterns for Intelligent Tutoring Systems](#)  
Dina Salah, Amir Zeid  
**Note:** These papers were originally submitted as one paper and split following workshop review.
3. [A Pattern Language for Online Trainings](#)  
Christian Kohls
4. [Patterns for Handling Students' Feedback](#)  
Axel Schmolitzky, Till Schümmer
5. [Pattern for Graduate Student Company Life Cycle](#)  
Petko Ruskov, Milena Stoycheva, Yanka Todorova

## Workshop E "Eames"

Workshop Leader: **Dietmar Schütz**

1. [Patterns for Product Line Engineering](#)  
Christa Schwanninger, Michael Kircher
2. [Reverse Variability Engineering](#)  
Dietmar Schütz
3. [Digital Signature with Hashing and XML Signature patterns](#)  
Keiko Hashizume, Eduardo B. Fernandez, Shihong Huang
4. [Dealing With Complexity](#)  
Klaus Marquardt
5. [Variability Patterns](#)  
Markus Voelter
6. [A Pattern Language of Black-Box Test Design for Reactive Software Systems](#)  
Alain-G. Vouffo Feudjio, Ina Schieferdecker

## Workshop F "Foster"

Workshop Leader: **Tim Wellhausen**

1. [A Good Fort has a Gap](#)  
Arto Juhola
2. [Towards a Pattern Language which Supports the Migration of Systems from an ET/P to a TTC Software Architecture](#)  
Michael Pont, Farah Lakhani, Anjali Das
3. [Invocation Flow Lines: Patterns of Invocation and Message Processing in Object Remoting Middleware](#)  
Stefan Sobernig, Uwe Zdun
4. [Hierarchical Property Loader](#)  
Tim Wellhausen, Martin Wagner, Gerhard Muller
5. [Software Architecture Patterns for Distributed Embedded Control System](#)  
Veli-Pekka Eloranta, Johannes Koski, Marko Leppänen, Ville Reijonen
6. Two Simple Patterns to Support the Development of Reliable, Real-time Embedded Systems (not submitted for proceedings)  
Anjali Das, Farah N. Lakhani, Ayman K. Gendy, Michael J. Pont

## Focus Group and Open Space reports

1. [Summary of the open session on pattern selection and pattern repositories](#)  
Edited by Aliaksandr Birukou

## Appendix

Selected photographs

# Enterprise Architecture Management Patterns for Enterprise Architecture Visioning

Sabine Buckl, Alexander M. Ernst, Florian Matthes, Christian M. Schweda

Chair for Informatics 19

Technische Universität München

eMail: {buckls, ernst, matthes, schweda}@in.tum.de

January 20, 2010

## 1 Introduction and Overview

Enterprise architecture (EA) management is one of the major challenges of modern enterprises. It aims at aligning business and IT in order to optimize their interaction. The general make-up of the enterprise is reflected in the EA, which comprises both business and IT aspects – ranging from visions (business, as well as IT visions are of interest), via business processes, and business applications, to infrastructure elements, like e.g. application servers or hardware.

Documenting and managing the EA is an advanced topic, as the application landscape, which is part of the EA often includes a few hundreds up to a few thousand business applications and their interconnections in a medium-sized or large company. Thereby, managing the EA is a task, that has to be executed as the need for a flexible IT is an integral concern of most companies. Nevertheless, other reasons for maintaining an EA documentation exist, such as compliance requirements or economic causes, i.e. the cost reduction of the IT function.

This article includes patterns on *EA Visioning*, which are part of the *EAM Pattern Catalog*, a *pattern language for enterprise architecture management* [BEL<sup>+</sup>07, BELM08, BEL<sup>+</sup>08, Ern08, Ern09], which uses a pattern-based approach to EA management. The complete *EAM Pattern Catalog* is available online at <http://eampc-wiki.systemcartography.info> [Cha09] and currently includes 162 EAM patterns. For a detailed explanation of the concept of EAM patterns refer to [Ern09]. The intention behind the article is to further extend and enhance the already documented EAM patterns and to document not yet described ones in order to advance the EAM pattern language.

The rest of this section list some remarks to writer's workshop participants, gives a short overview about the intended audience, and a map of included EAM patterns and their references.

---

Copyright retain by author(s). Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.

### 1.1 Intended Audience

This article and the herein included patterns are intended for people concerned with governing the information technology (IT) of a company, aligning business and IT, and people concerned with bringing together information about business and IT aspects of the enterprise. Especially the topic of EA visioning is addressed by the patterns included in this article.

Potential Readers for this article are: people caring about strategies and visions for EAs from business and from IT, business architects, enterprise architects, and business application owners.

### 1.2 Map of included EAM Patterns

The EAM patterns included in this article are part of a larger pattern language and therefore relationships between EAM patterns are an integral part of this approach. Figure 1 shows a pattern map visualizing these relationships and descriptions about their type. The pattern map also includes references to patterns which are not included in this article. Patterns are referenced by their names, page numbers are included in brackets.

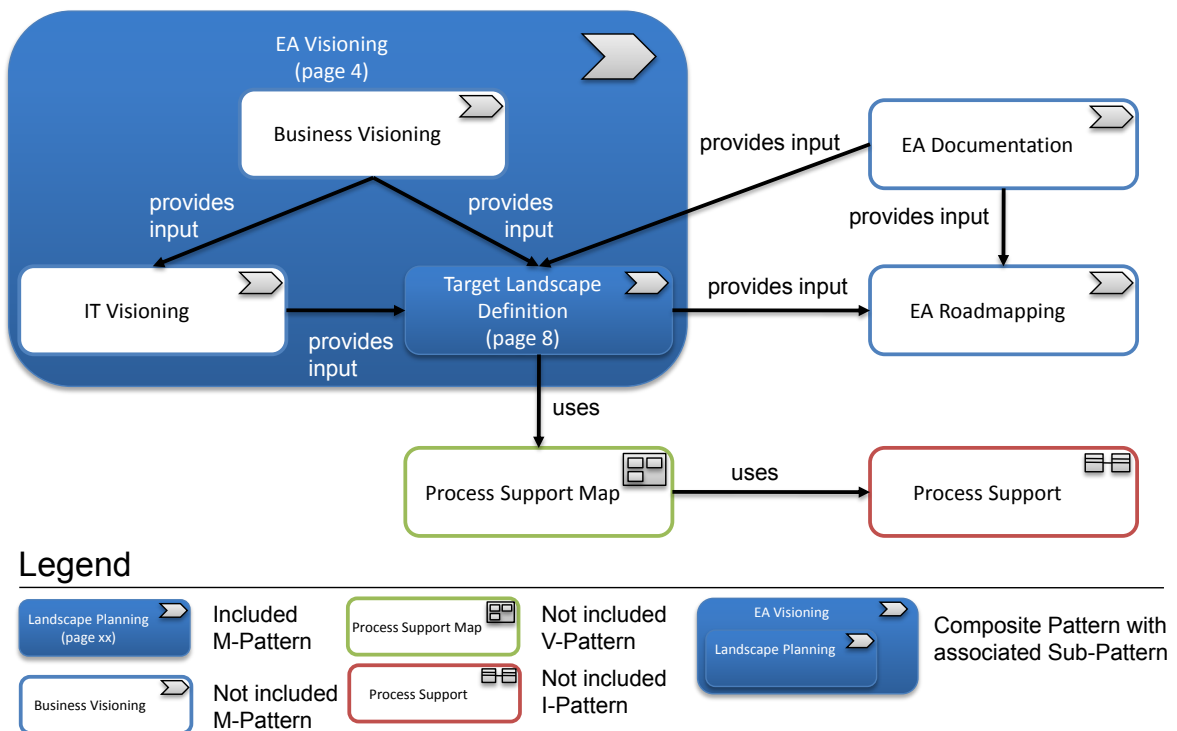


Figure 1: Pattern Map for this Article

The following EAM patterns are included in this article.

- EA VISIONING (see page 4)
- TARGET APPLICATION LANDSCAPE DEFINITION (see page 8)

These EAM patterns are not included in this article and have to be documented or can be found in the EAM Pattern Catalog Wiki [Cha09]:

- BUSINESS VISIONING documents the steps to develop a business vision, which is aligned with the business strategy of the company.
- IT VISIONING provides insights about how to develop and maintain a vision about the future development of the IT. An IT vision is not as concrete as an IT target and should match the IT strategy of the company.
- EA DOCUMENTATION cares about how to document the elements an enterprise architecture consists of, like business applications, business processes, etc.
- EA ROADMAPPING uses the information documented about the enterprise architecture to create and maintain a roadmap on the future development of the enterprise architecture.
- PROCESS SUPPORT MAP (see [Cha09]) visualizes, which BusinessApplications support which BusinessProcesses at which OrganizationalUnits.
- PROCESS SUPPORT (see [Cha09]) shows how information about which organizational unit uses which business application to support which business process can be stored.



## 2 EA Visioning

EA VISIONING describes the general process of EA visioning. The term *EA strategy* is widely used interchangeably with the term *EA vision*. We prefer the later term in accordance to [Gro08], where a *vision* is referred to as distant goal, while a *strategy* is understood as series of activities to pursue such goal. Based on BUSINESS VISIONING, IT VISIONING, and input from the current EA DOCUMENTATION a TARGET LANDSCAPE DEFINITION is derived, which then provides input for EA ROADMAPPING resulting in projects that have to be conducted in order to adapt the EA according to the defined vision.

### 2.1 Example

The department store *SoCaStore* has to continually adjust its business vision to the changing economic environment and to ensure consistency as well as appropriateness of this vision in respect to regulatory requirements. Additionally, emerging IT-trends and new technologies available make it necessary to adapt the IT vision, in order to effectively use the resulting opportunities. From the business and IT strategy, SoCaStore wants to develop and reshape the vision of the EA to achieve an optimal alignment of business and IT under the changed circumstances.

### 2.2 Context

An enterprise, which wants to create a holistic vision of its future EA, taking into account market trends, regulatory changes, and emerging IT-technologies to achieve increase alignment between business and IT.

### 2.3 Problem

You want to ensure that the vision of your company's EA factors in the relevant environmental changes and provides both a consistent business and IT vision to guide the evolution of the EA. **How can you prevent an organization from losing its ability to develop effective long-term strategies?**

The following *forces* influence the solution:

- **Market orientation versus innovative visioning** If the EA visioning goes the same ways as the visioning in other companies acting in the same market, the enterprise might be able to compete with the other companies. Nevertheless, visioning in such a direction is likely to limit the corridor of evolution, especially in respect to new markets.
- **Market uncertainty versus stable markets** Operating in uncertain markets may result in better opportunities but also implies higher risk. Stable markets constitute the contrary situation. In what kind of market is your company operating and how does this influence your EA visioning.
- **Separation of concerns versus smooth transition** The EA visioning process benefits from knowledge and experience of the people fulfilling certain roles. These roles nevertheless demand strongly different skill sets, e.g. in business visioning and IT visioning. In consequence, the boards for performing these activities could be separated

strictly, which might negatively affect innovative power, especially when IT as enabler is considered.

- **Holistic visioning versus selective visioning** The success of EA visioning is dependent on the focus on the business content. A broader focus may lead to better overall results but it is more difficult to get the required input by the business. How do you find the right focus of your visioning approach?
- **Continuous adaptation versus one time approach** Continuously adapting the vision of your EA may lead to the best results but requires high efforts. A one time approach does not require high investments but may be outdated soon. What is a good space of time for reconsidering the created EA vision?
- **Regulatory instability** Rules and regulations do not stay the same over time. They change and require changes concerning the vision of your EA. How do you find a balance between adapting to regulatory changes and keeping already developed visions of the EA?

## 2.4 Solution

The development of an EA vision is a compound process consisting of distinct activities as indicated in Figure 2. These activities themselves are quite coarse grained and are detailed in separate M-Patterns: BUSINESS VISIONING, IT VISIONING, and TARGET APPLICATION LANDSCAPE DEFINITION (see page 8). In the notion of a *composite pattern* [Cop96, BHS07], this EA VISIONING describes the coordinating process of EA visioning, caring about the correct execution of the contained activities. Subsequently, we sketch the role of these activities and detail on the exchange of information and knowledge between connected activities.

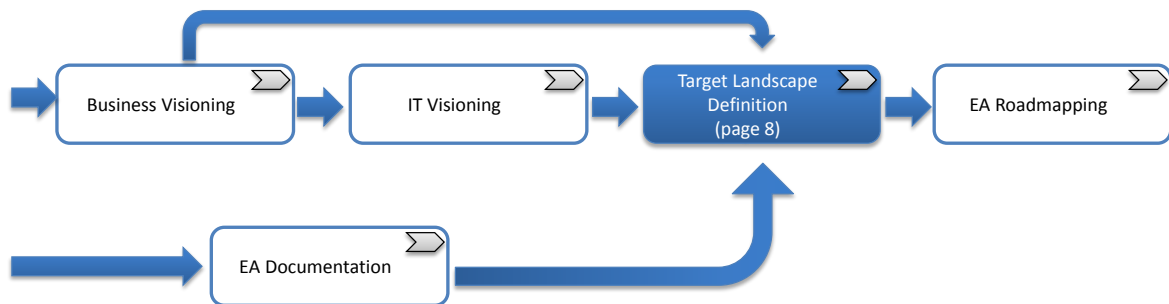


Figure 2: EA Visioning Process

**Business Visioning** In this activity, business plans and visions are developed. Therefore, current trends in the market fields, which the company is acting in, are analyzed and possible scenarios for the future development of the markets are created and prioritized according to their likelihood. Additionally, related market fields should be taken into consideration to supply ideas for diversification or to give indications on potential future developments. From this input, a joint vision of the business is developed and a complementing business mission is formulated. Subsequently, both vision and mission are detailed to goals and strategies, respectively. The Business Motivation Model (BMM) [Gro08] of the Object Management Group

(OMG) establishes a language, which can be used during business visioning and sketches additional process steps, which could be supportive during the execution of this activity.

**IT Visioning** The business vision and mission from business visioning provide input to this activity, where new technologies and IT trends are discussed and analyzed in respect to the applicability for supporting the business plans. During the activity, standardization endeavors targeting the IT support in the respective business area are assessed.

**Target Landscape Definition** The business vision determines the framework for the target landscape to be developed. The target landscape is often (see e.g. [EHH<sup>+</sup>08]) alluded to as *to-be* landscape. We decided to stay to the former terminology, as the term *to-be* could also apply to *planned landscape*, i.e. landscapes, which result from the execution of concrete project portfolios. For in-depth discussions on this topic see e.g. [BDM<sup>+</sup>08]. The IT vision lays the basis for the target landscape by defining concepts, standards, and technologies, which are preferably used in realizing the IT support for the business vision. The documented current landscape provides valuable input for the discussions on the target landscape, especially in areas, where business and IT vision do not differ substantially from the currently established business and IT plan, or where business and IT vision do not exert influence upon.

## 2.5 Implementation

EA VISIONING should incorporate people from the business, as well as from the IT part of your company. This is important as EA VISIONING has an high impact on the future development of your company and the capability to support new business requirements.

Developing or revising the EA VISIONING depends on the planning cycle of your company, which may also be dependent to market demands. Manufacturing companies for example usually feature longer planning cycles then telecommunication companies.

## 2.6 Variants

A variant of EA VISIONING is that there is no business vision, which can be used as an input for the IT vision. In this case an IT vision is created without input from business. This usually results in an IT vision which does not meet the future development and goals of the company. Therefore, this variant should be avoided.

## 2.7 Known Uses

The approach documented in EA VISIONING is in use in the following companies:

- BMW

The approach documented in EA VISIONING can be used in the following EA management tools

- planningIT (alfabet AG)
- ARIS IT Architect (IDS Scheer AG)

## 2.8 Consequences

**Market orientation versus independent visioning** EA VISIONING may encompass higher future benefits if you don't only consider your own market because you may penetrate other markets which you right now cannot address appropriately. If you try that approach you have the problem, that you cannot consider every available market. For this reason, you have to select a few markets you orient at for EA VISIONING. Orienting only on your own market may be the safe way but may lead to lower benefits in the future.

**Market uncertainty versus stable markets** The business vision does not only account for today's market situation, but has to anticipate future market trends and identify developing opportunities. Especially the latter are inevitably associated with risks, so that during business visioning you have to decide on an appropriate spread between safe and risky business goals.

**Separation of concerns versus smooth transition** The separation of concerns, i.e. the assignment of domain experts *only* to the boards deciding on the business and IT vision respectively, may be helpful for keeping this boards small and agile. Further, bringing together experts from one domain reduces the potential for misconceptions during the discussion, which is in such case more likely to be based on a consistent understanding of the used terms. In contrast, bringing together experts from the business and IT domain for visioning can help to leverage the full potential of IT as an enabler for business opportunities. Additionally, a joint discussion board for visioning may help to avoid error-prone translations between the domain terminologies.

**Holistic visioning versus selective visioning** On one hand, focusing you EA VISIONING to only some business aspects of your company may reduce the required effort. On the other hand this may lead to sub optimal overall solutions for your whole company. For this reason, you have to find the right focus for you EA VISIONING.

**Continuous adaptation versus one time approach** Doing EA VISIONING once is relatively simple and requires less effort compared to an iterative approach, which regularly checks for required changes to your EA vision. Although, you should consider to establish a continuous approach if you have once invested in an EA VISIONING. The benefits exceed the required effort.

**Regulatory instability** Rules and regulations usually have to be adopted in an appropriate way. For this reason, you have to find a solution to incorporate changes to rules and regulations within you EA VISIONING. In some cases there are transition periods, which you can use to adopt your vision to match the new guidelines.

## 2.9 See Also

EA VISIONING is a composite pattern and therefore the following sub patterns should be considered:

- BUSINESS VISIONING
- IT VISIONING
- TARGET APPLICATION LANDSCAPE DEFINITION (see page 8)

Additionally EA DOCUMENTATION should be considered as it provides input for TARGET APPLICATION LANDSCAPE DEFINITION.

### 3 Target Application Landscape Definition

TARGET APPLICATION LANDSCAPE DEFINITION describes the process of defining a target landscape derived from the business and IT vision of the enterprise. In addition, the current documentation of the EA is used as input for the development process. Thereby, the target landscape defines the vision of future business processes and the support provided by the IT.

#### 3.1 Example

The department store *SoCaStore* wants to gain a common understanding how the target landscape of their enterprise looks like. As the existing application landscape has grown historically, the impacts and influences current projects might have on the application landscape are hardly predictable and the direction in which the landscape should be developed is not clear. Therefore, a common vision of the *optimal* future landscape derived from the strategies – both business and IT – of the enterprise is necessary.

#### 3.2 Context

An enterprise, which wants to gain a common understanding how the target landscape providing optimal business support according to the currently defined strategies looks like.

#### 3.3 Problem

You want to ensure that the evolution of your application landscape takes a direction, which conforms to the strategies of your enterprise. You want to know how an optimal future landscape would look like. **How do you define an optimal target landscape according to the strategies of your enterprise?**

The following *forces* influence the solution:

- **Planning intervals versus effort** Small planning intervals may be required to do detailed planning, but require high efforts as you need up-to-date information. Which planning intervals should be selected for landscape planning?
- **Complete control versus laissez-fair** Do you want to control all projects changing the application landscape within your company or is there a limit in project size or cost, which allows to ignore smaller projects?
- **Long-term versus medium-term planning** Do you just care about a medium-term planning cycle or do you consider a long-term, visionary target?
- **Efficiency versus thoroughness** Detailed planning of the target landscape up to the level of individual components of business applications may be required, but results in high efforts, which may not be worthwhile. What is the right level of detail for planning a target landscape for your enterprise?
- **Business versus IT demands** The demands of the business units of a company may conflict with the demands of the IT units concerning the future development of the landscape. How do you find a good balance between these conflicting requirements?



- **Long-term integrity versus short-term business benefits** Planning of a target landscape should consider long-term integrity of the landscape, e.g. to reduce heterogeneity, this usually conflicts with short-term business benefits. What is a good balance between those two aspects?
- **Ideal target landscape versus pressure of time** Developing an ideal target landscape requires time to develop it. This conflicts with pressure of time in an operative department. How much time can you invest to define a target landscape useful for the future development?
- **Legal requirements versus freedom of choice** In some situations legal requirements may apply, which delimit the freedom of choice for the target landscape development. What legal requirements do you have to consider?
- **Holistic planning versus partial planning** Considering the whole landscape in target landscape planning may lead to optimal results but demands high efforts. Contrary, planning of parts of the landscape requires less effort but may lead to suboptimal results. How do you balance holistic versus partial target landscape planning?

### 3.4 Solution

In order to develop target landscapes of the enterprise, a step-wise approach to decrease complexity should be used [EHH<sup>+</sup>08]. Before executing the five steps illustrated in Figure 3, you should decide on the level of detail regarding the planning and about the planning interval in order to decide on the effort required to execute the process.



Figure 3: Target Landscape Definition Process

#### Step 1:

Prior to starting with the actual development of the target landscape some decisions about the approach used have to be taken. Based on the environment and context of the initiative a *disruptive* or *evolutionary* approach should be chosen. Whereas a disruptive approach is well suited if a comprehensive reorganization of the enterprise is desired, e.g. after a merger has taken place, the evolutionary approach provides a softer transformation as it is build on existing structures of the enterprise and supports smoother transitions. The choice between these two approaches implies major impacts on the following steps.

#### Step 2:

If a disruptive approach is followed, the organizational model of the enterprise is developed from scratch based on the business vision of the enterprise. Otherwise, if an evolutionary approach is used, the current organizational model is augmented utilizing the business vision [Sch08]. Elements of this model, e.g. business processes, domains, business units, etc. are used as clusters to structure the landscape and decrease complexity. These clusters may be used to split-up the planning activities. This may lead to sub optimal results but requires less effort. If the complexity of the landscape demands a stronger clustering, the clusters may contain subclusters, e.g. business processes may be structured into sub business processes.

**Step 3:**

Based on the business and IT vision of the enterprise the areas should be identified where business support is provided by the enterprise itself and where support is gained from third parties via outsourcing. Thereby, regulatory limitations, like taxes, originating e.g. from country-specific laws are taken into consideration.

**Step 4:**

Based on the developed framework, derived from the organizational model – business processes and organizational units – the business support provided by IT has to be derived. Thereby, the business and IT vision of the enterprise is considered to identify parts of the application landscape where horizontal or vertical integration should be applied. Although both kinds of integration lead to synergy effects e.g. homogenization and cost reductions, they might not be applicable:

**Vertical integration**, which refers to uniform process support for different organizational units, products, or locations, is e.g. not applicable if the business vision of the enterprise asks for diversification in different markets.

**Horizontal integration**, which means that several successive business processes are continually supported by one business application, is e.g. not applicable if the IT vision demands for different kind of IT support during the execution of two sequenced business processes.

Furthermore, regulatory limitations as mentioned in Step 3 should be considered during the definition of business support provided by business applications.

**Step 5:**

The derived target landscape needs to be communicated among the various stakeholders of EA management to gain an enterprise-wide understanding of the future vision of the application landscape. In this step you should consider the required level of detail in documenting the planned landscape. Documenting on business application component level may be very accurate but requires high effort.

Different versions of PROCESS SUPPORT MAP (see [Cha09]) are commonly used to document the picture of the target landscape. In order to create these documentations, the respective data has to be stored in a repository implementing PROCESS SUPPORT (see [Cha09]).

### 3.5 Implementation

There is no ideal planning interval for TARGET APPLICATION LANDSCAPE DEFINITION but it should be aligned with the interval defined for developing/revising the strategies of your company.

The time required to execute the steps described in TARGET APPLICATION LANDSCAPE DEFINITION may also vary based on the size of the application landscape respectively the cluster selected for landscape planning. Another factor influencing the execution is the selected level of detail. In both cases you have to balance between effort and benefit.

There are various people, which should be incorporated in TARGET APPLICATION LANDSCAPE DEFINITION. *Business application owners* should be considered when trying to replace one business application by another one, e.g. when trying to increase vertical integration. *Enterprise architects* should also be incorporated as they have a more holistic view on the application landscape in contrast to the business application owners, which focus on single

business applications. The enterprise architects should be in lead of the process of TARGET APPLICATION LANDSCAPE DEFINITION and should thereby keep contact to the people deciding on the business and the IT strategy of your company.

### 3.6 Variants

Variants of TARGET APPLICATION LANDSCAPE DEFINITION may emerge if the subject of planning changes. The previously described steps may apply to business applications but they may also apply e.g. to services. This may result in a more fine grained planning leading to higher efforts but also to higher flexibility because services are typically smaller in functionality or supported capability.

### 3.7 Known Uses

The approach documented in TARGET APPLICATION LANDSCAPE DEFINITION is in use in the following companies:

- BMW
- Nokia Siemens Networks
- Munich Re

The approach documented in TARGET APPLICATION LANDSCAPE DEFINITION can be used in the following EA management tools

- planningIT (alfabet AG)
- ARIS IT Architect (IDS Scheer AG)

Similar approaches for TARGET APPLICATION LANDSCAPE DEFINITION can be found in literature, see e.g. [Der06].

### 3.8 Consequences

**Planning intervals versus effort** At first sight it may be appealing to be able to plan the future development whenever this is needed. To be able to do this requires high efforts as the information needed for the future planning has to be up-to-date the whole time and this fact has a high impact on the information collection style used in the company (see [MJBS09]). E.g. it is not possible to update information about the application landscape once a year if you want to do continuous planning. An advantage of continuous planning is that you are able to react instantly on new demands. The method described in TARGET APPLICATION LANDSCAPE DEFINITION supports various planning intervals, but you should always consider the required effort, which is tied to the length of the planning intervals. Typically planning according to TARGET APPLICATION LANDSCAPE DEFINITION is done once a year.

**Complete control versus laissez-fair** You can try to control every project affecting your application landscape, but in this case you will have to spend most of your time on performing this task. Another way to cope with this situation is to define a limit, e.g. on project size or costs, which has to be exceeded for the project to come into focus in planning the target application landscape. You may now miss some of the smaller projects, but you can focus on the important ones.

**Long-term versus medium-term planning** The easy way is to only care about the next planning period, because you don't have to care about the strategies or goals of your company, and you can find an optimal solution for your current problems. But you will miss an overall goal, bringing together all future developments. If you also consider long-term planning you have to care about the strategies and goals of your company and even have to define your own strategies or goals but on the long run this approach will pay off.

**Efficiency versus thoroughness** Selecting the right level of detail for TARGET APPLICATION LANDSCAPE DEFINITION is no easy to address task. As a rule of thumb you should always think about what amount of information do you really need when you want to implement your planning and what efforts are required to reach this level of detail. Typically it should be enough to stay on the level of business applications restraining from going into more detail.

**Business versus IT demands** Demands from business and from IT units usually differ. One reasons for this is that business is first of all interested to get a required functionality as fast as possible into place for the lowest price, without considering the future development of the business application and its surrounding application landscape. This conflicts with IT units demands, which should incorporate the future operation and development of the business application. In cases where this conflict appears you should try to find a compromise between the two positions.

**Long-term integrity versus short-term business benefits** This force is similar to business versus IT demands. And the resolution is also similar. Try to find a compromise between the two positions at least in a long-term perspective.

**Ideal target landscape versus pressure of time** Pressure of time is a problem that is always hard to address, but you should consider the result of not spending a minimum amount of time on TARGET APPLICATION LANDSCAPE DEFINITION. This may result in a future landscape, which becomes more and more hard to manage and at some point the investments needed to improve the landscape to be manageable again exceed the costs required for TARGET APPLICATION LANDSCAPE DEFINITION.

**Legal requirements versus freedom of choice** In some cases legal requirements may apply in TARGET APPLICATION LANDSCAPE DEFINITION. In this cases the possibilities are limited to increase the freedom of choice again. For this reason, you should try to get along with the restrictions and try to find a solution still fitting your future demands.

**Holistic planning versus partial planning** Holistic planning should result in an overall better solution then restricting TARGET APPLICATION LANDSCAPE DEFINITION e.g. to a single cluster. But the efforts to plan the overall solution may exceed the benefits of an overall optimal solution. Again, try to find a solution which balances both approaches to find the solutions fitting your demands.

### 3.9 See Also

In order to support the implementation of TARGET APPLICATION LANDSCAPE DEFINITION the PROCESS SUPPORT MAP (see [Cha09]) should be considered. Additionally, EA VISIONING as a composite pattern should be considered for supplemental advice which other patterns should be taken into account.

## 4 Acknowledgment and Outlook

This section includes acknowledgments to the people who supported the creation of this article and gives an outlook to the next steps in the development of the EAM pattern approach.

### 4.1 Acknowledgments

We want to thank all participants of the writer's workshop of EuroPloP09 and especially our shepherd **Wolfgang Keller** for the time they spent for reading, commenting, and discussing this article.

### 4.2 Next Steps in EAM Pattern Approach Development

The *EAM Pattern Catalog* is currently available at <http://eampc-wiki.systemcartography.info>, based on the results of an extensive online survey. Certainly, the EAM patterns should continually be revised for readability and understandability and be extended to give more detailed guidance in addressing the problems of EA practitioners, preferably by an EAM Patterncommunity.

In order to improve the current version and to further exploit the advantages of patterns in EA management, an excerpt of the *EAM Pattern Catalog* had been included in this document to be discussed in the pattern community.

## References

- [BDM<sup>+</sup>08] S. Buckl, T. Dierl, F. Matthes, R. Ramacher, and C. M. Schweda. Current and future tool support for ea management. In U. Steffens, J.S. Addicks, and N. Streekmann, editors, *MDD, SOA und IT-Management (MSI 2008)*, Berlin, 2008. GITO-Verlag.
- [BEL<sup>+</sup>07] Sabine Buckl, Alexander M. Ernst, Josef Lankes, Kathrin Schneider, and Christian M. Schweda. A pattern based approach for constructing enterprise architecture management information models. In *Wirtschaftsinformatik 2007*, pages 145 – 162, Karlsruhe, Germany, 2007. Universitätsverlag Karlsruhe.
- [BEL<sup>+</sup>08] Sabine Buckl, Alexander Ernst, Josef Lankes, Florian Matthes, and Christian M. Schweda. Enterprise architecture management patterns – exemplifying the approach. In *The 12th IEEE International EDOC Conference (EDOC 2008)*, Munich, 2008. IEEE Computer Society.
- [BELM08] Sabine Buckl, Alexander M. Ernst, Josef Lankes, and Florian Matthes. Enterprise Architecture Management Pattern Catalog (Version 1.0, February 2008). Technical report, Chair for Informatics 19 (sebis), Technische Universität München, Munich, Germany, 2008.
- [BHS07] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages*. Wiley, 2007.



- 
- [Cha09] Chair for Informatics 19 (sebis), Technische Universität München. Eam pattern catalog wiki. <http://eampc-wiki.systemcartography.info> (cited 2009-11-06), 2009.
- [Cop96] James Coplien. *Software Patterns: Management Briefs*. Cambridge University Press, 1996.
- [Der06] Gernot Dern. *Management von IT-Architekturen (Edition CIO)*. Vieweg, Wiesbaden, 2006.
- [EHH<sup>+</sup>08] Gregor Engels, Andreas Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann, and Jan-Peter Richter. *Quasar Enterprise – Anwendungslandschaften serviceorientiert gestalten*. dpunkt.verlag, Heidelberg, Germany, 2008.
- [Ern08] Alexander Ernst. Enterprise architecture management patterns. In *PLoP 08: Proceedings of the Pattern Languages of Programs Conference 2008*, Nashville, USA, 2008.
- [Ern09] Alexander M. Ernst. *A Pattern-Based Approach to Enterprise Architecture Management*. PhD thesis, Technische Universität München, München, Germany, 2009.
- [Gro08] Object Management Group. Business motivation model 1.0, 2008.
- [MJBS09] Christoph Moser, Stefan Junginger, Matthias Brückmann, and Klaus-Manfred Schöne. Some process patterns for enterprise architecture management. In *Software Engineering 2009 – Workshopband*, pages 19–30. Lecture Notes in Informatics (LNI), 2009.
- [Sch08] Jaap Schekkerman. *Enterprise Architecture Good Practices Guide – How to Manage the Enterprise Architecture Practice*. Trafford Publishing, Victoria, BC, Canada, 2008.

# Lead Roles In A Software Project

Andreas Fießer  
patterns@fiesser.de  
January 2010

This paper discusses patterns about the lead roles of a software development team: The Product Manager who is the interface to the customer, the Project Manager who allocates the resources and the Architect who designs the software and leads the developers.

A fourth pattern discusses how these roles act as hubs and how they relate to each other. The view is based on the project management triangle.

## 1 Introduction

Projects are teamwork. When teams reach a certain size they require organizational structure to be effective. Team members need to set different emphases in their work to make sure the project goals are fulfilled.

This paper illustrates how different roles can be used to organize a software development team. When looking at the flow of communication, it becomes apparent that distinct roles communicate differently.

As Coplien and Harrison describe in their pattern HUB, SPOKE, AND RIM some roles act as hubs [Coplien Harrison 2005]. Typical examples of such hubs are the PRODUCT MANAGER, the PROJECT MANAGER and the ARCHITECT. Depending on the situation roles like developers or testers might also act as hubs.

## 2 Audience

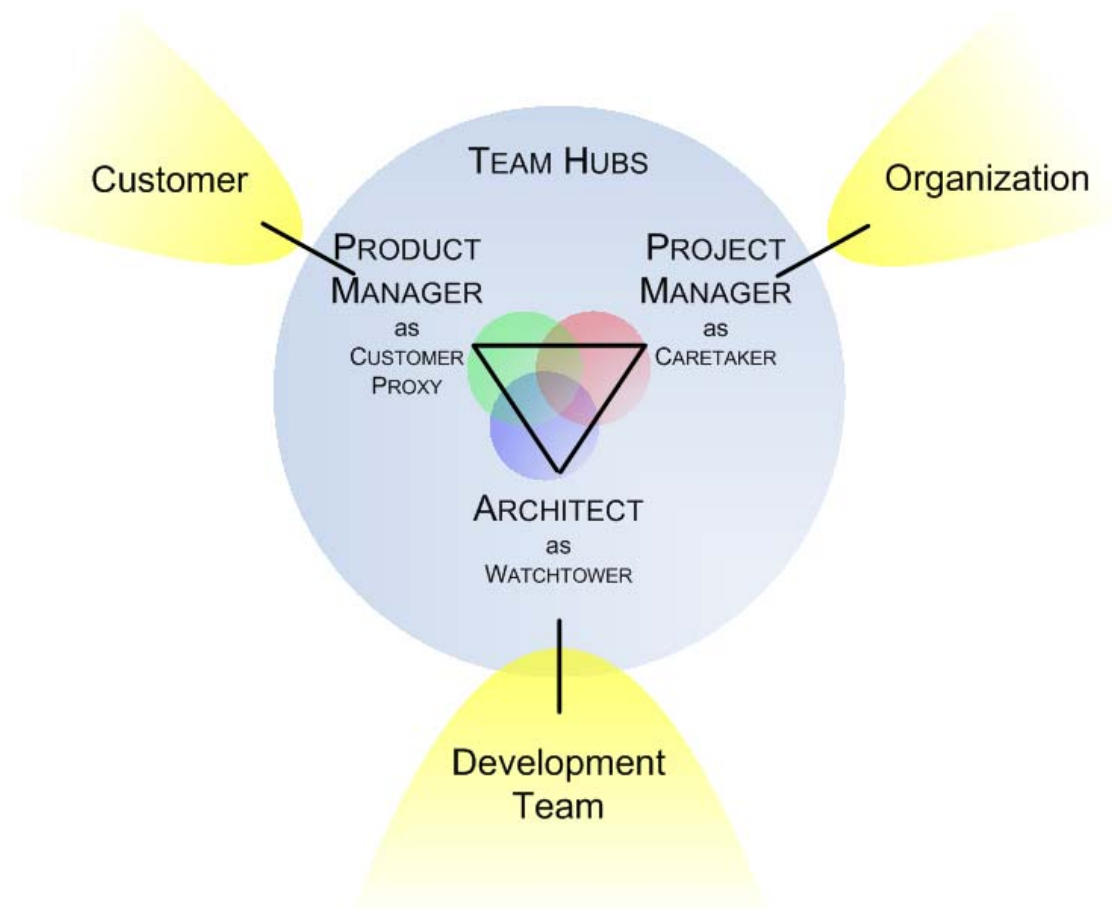
The patterns presented here try to structure communication and responsibilities in a project team. The focus of this paper is planning project teams, not the detailed actions of the respective team members. The patterns might also work in other domains than software development.

The author's experience stems largely from web development in small teams of less than a dozen people. The products created are individually constructed and the development process takes several months. This setup is also the focus of this paper. Nevertheless the patterns also apply to other scenarios. The author welcomes any kind of feedback about the relevance of these patterns in other setups.

For the scope of this paper, creating a mass market product can be treated in the same way as a tailor-made product ordered by a customer. In both cases there are certain needs that are to be met and thus features that have to match the needs.

Copyright retained by author. Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.

### 3 Pattern Map



**Fig. 1: Hubs and Stakeholders**

For a software project stakeholders come together from different directions. Their roads cross inside the project team. The goal of the project has to be to avoid conflicts and turn the different views into an advantage.

After taking a closer look at the use of TEAM HUBS in this context, the three roles of the PRODUCT MANAGER, the PROJECT MANAGER and the ARCHITECT are presented in patterns.

The patterns in this paper focus on the hub aspects of the mentioned roles. Although this is only part of the roles, the patterns are named after the corresponding roles coined by literature and practice for readability. Their hub function is expressed by their alias names: CUSTOMER PROXY, CARETAKER, WATCHTOWER.

## 4 The Patterns

### 4.1 TEAM HUBS

#### Context

A team of several people is working on a software project that has a limited budget and a deadline. The customer expects the finished product to meet his expectations. Some tasks in a project are not closely related to coding.

#### Problem

**How to organize the flow of information in a project?**

#### Forces



**Fig. 2: The Project Management Triangle**

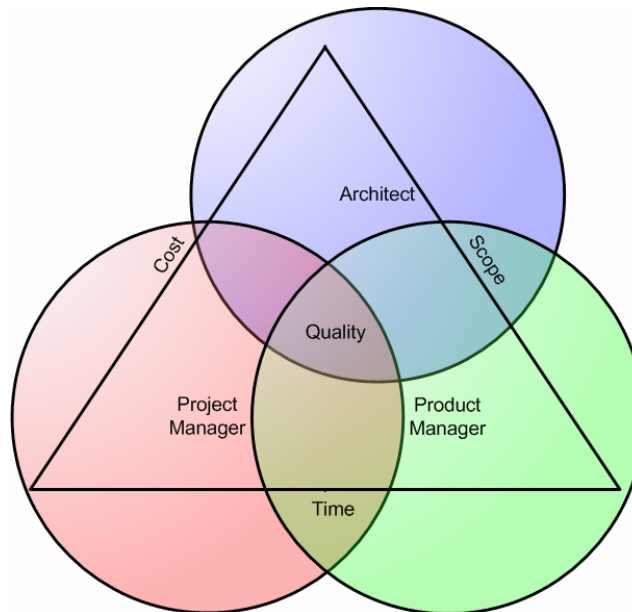
- As illustrated by the Project Management Triangle [Schwanninger Kircher 2009, Kelly 2008, Wikipedia 2009a] projects happen within certain constraints. You cannot change one aspect without touching another. Focussing on all four conflicting aspects at the same time is very difficult for a single person. Letting every team member acquire all skills necessary to carry out a project is not effective either. Management tasks and coding require completely different mindsets [Kelly 2009a].
- As the developer is the one who actually creates the product, he should play a central role in the process. Nevertheless he needs support by others.
- Communication becomes more complex with an increasing number of participants.

## Solution

**Classify the aspects of a project and channel the communication and tasks they implicate over dedicated hubs.**

These hubs perform all the management activities that are not directly connected to coding. They actively look for tasks that fall to their sphere and support the team with their skills.

To formalize the hubs create several roles, each only focussing on a subset of the four aspects. There are many roles in a software project. Typical examples of hub roles include the PRODUCT MANAGER, the PROJECT MANAGER and the ARCHITECT. These roles bundle the interaction between their respective group and the outside. This does of course include other hubs.



**Fig. 3: Aspects covered by roles**

Sometimes hubs emerge naturally from the characters of the team members. A beneficial formal organization structure fosters these momenta instead of blocking them.

The most obvious way to separate the tasks is to assign the roles to different individuals. If there is nobody available full time, make sure that somebody dedicates at least a share of his time to switch views. Although it reduces the necessary communication efforts this option can only be a workaround as it requires considerable discipline from that person to stay objective in both roles [Kelly 2009a, Kelly 2009c].

## Rationale

Assigning an aspect to a role makes sure that every constraint is sufficiently taken into consideration when making decisions. Overlapping the aspects makes it easier to find a compromise of two positions.

In software development the types of quality that the roles care about are different in focus. As Schwanninger and Kircher explain, high quality for the ARCHITECT means developmental quality, i.e. maintainability of the code. For the PRODUCT MANAGER it is the user perceived quality, e.g. usability and performance. The PROJECT MANAGER focuses mainly on regulatory quality. This means he has an eye on the compliance with standards the company or the project have committed to [Schwanninger Kircher 2009].



## **Consequences**

### **Benefits**

- Representing a specific role makes it easy for a team member to have a specific point of view and argue for it.
- It also allows for the roles to be filled not by generalists but by specialists. So a wider range of skills and experience can be considered when making a decision.
- Overlapping spheres might seem like ineffective redundancy at first. But they guarantee that the roles share common interfaces they can use for negotiation.
- Since requirements, planning and implementation are done by different people each of them is more objective as they are less biased by the other side.
- The work of the developers is coordinated and prepared by lead roles with a different focus. This relieves the developers from organizational details and other non-technical issues. They can focus on their strength: Coding. The developers as well as the stakeholders of the project know whom to address with an issue.
- Introducing more than one leading role leads to a division of powers that prevents arbitrariness.
- A team member fills a role with his skillset and experience. If the person leaves the team, it is clear which requirements the replacement person needs to fulfil to be able to take over the role.

### **Liabilities**

- As a person is asked to fill a predefined role, the team member might feel pressed into a role that does not fit well. It is important to allow the freedom that the role adjusts to the person as long as this does not touch other goals.
- When splitting up into different roles, team members have different responsibilities. This tends to create a hierarchy between managers and producers. As the roles described here have no disciplinary rights, this requires more networking and a more sophisticated style of communication.
- The PROJECT MANAGER and the PRODUCT MANAGER are detached from the code. So it is necessary to abstract communication when talking about technical issues.
- Explicitly separating the aspects of the project creates the opportunity to delegate responsibilities for a problem. If the one the problem was delegated to does not take care of the issue properly, there is a risk that nobody feels responsible any more and the problem is neglected.
- Separating the roles creates a management layer. This requires extra manpower and creates extra cost.

### **Example**

A caravan manufacturer contacts a web agency with the blurry vision to add a section to his website that can exclusively accessed by his retailers.

Before the web agency's developers start to work, the PRODUCT MANAGER advises the customer on the set of useful features to support his intentions.

After the PRODUCT MANAGER and the ARCHITECT have jointly estimated the required budget, the customer picks the features he wants to get implemented.

The PROJECT MANAGER now allocates the available resources.

The developers start to work within the given structure of requirements and deadlines.

### **Related Patterns**

The PROJECT MANAGER, the PRODUCT MANAGER, and the ARCHITECT act as TEAM HUBS. The ARCHITECT contributes technical overview. The PRODUCT MANAGER shares his knowledge of the target market and the PROJECT MANAGER contributes monitoring and organization.

Coplien and Harrison have created a number of patterns that give some details on how the TEAM HUBS should work. A productive team is a COMMUNITY OF TRUST in which WORK FLOWS INWARD to the developers so that the DEVELOPER CONTROLS PROCESS (in the team) and the other roles are supporters. Depending on the context this pattern is complemented by HUB, SPOKE, AND RIM. The approach that developers ENGAGE CUSTOMERS finds its limits in a FIREWALL that shields the developing staff from external interruptions and noise and a GATEKEEPER that facilitates the flow of useful information between the team and the outside.

PRODUCT MANAGER, PROJECT MANAGER and ARCHITECT are implementations of these patterns for their domain [Coplien Harrison 2005].

See BALANCE CONSTRAINTS [Schwanninger Kircher 2009] about how to cope with the aspects of the project management triangle.

## 4.2 PRODUCT MANAGER

### Also known as

CUSTOMER PROXY

### Context

The customer has certain needs and a rough idea about how the software should meet them. The customer has limited resources to supervise the project. The ARCHITECT and the development team are focussed on solving technical problems and creating features.

### Problem

**How can you ensure that the product meets the customer's expectations?**

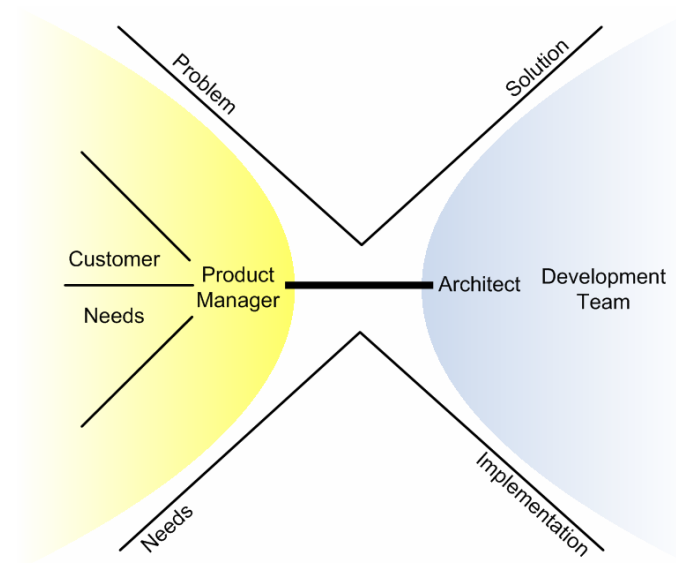
### Forces

- The customer wants the product to meet his expectations but he might not know how to communicate them. The customer might not have a consistent view of his needs and has no overview over the technical possibilities.
- The customer is not interested in the concrete technical implementation. He wants his requirements to be fulfilled.
- The customer is required for feedback but he might not be available when needed or willing to get involved in the development process [Völter Kircher 2008a].
- Developers are not experts in the customer's domain. Because they should be able to relate to the intention of the requirements, they need some understanding of that domain without investing too much time to gather the required knowledge.
- Features that look easy to the customer might turn out to be complicated to implement. This might prolong the development process.
- Taking care of the customer can be a time consuming process [Kelly 2009b].

### Solution

**Bundle communication between the team and the outside.**

Gather market knowledge in one point to create a perspective free from the details of technical implementation inside the development team. This creates the role of a PRODUCT MANAGER who is able to support the customer in specifying his requirements and explain these requirements to the developers. He helps to clean up inconsistencies and point out solutions the customer might not have thought of.



**Fig. 4: The Product Manager at the Focal Point**

The less the actual customer is available, the more the requirements become a base for the PRODUCT MANAGER to act as a SURROGATE CUSTOMER for the team [Coplien Harrison 2005].

In the outward direction the PRODUCT MANAGER lets the customer know when a certain request involves extraordinary technological challenges. This allows them to see why a certain feature creates so much effort or even trade it in for some less complicated features that achieve a similar effect.

Starting from that point and keeping the customer in mind, the PRODUCT MANAGER can turn to the inside of the team and start negotiating with the development team lead by the ARCHITECT about the actual way to implement a feature. In this process the decisions are made, which technology will have to be implemented. The choice might be made in spite of the challenges it brings. Technologies that seem fancy at first may be cancelled because they do not add to the value of the product [Kelly 2009b].

When discussing within the team it is important to make sure that the features planned and their priorities match with those of the customer. Consequently, substantial technical constraints in implementation need to be carried on to the customer to find an efficient solution. It is important to use this option with caution. It is not the sole purpose of the PRODUCT MANAGER to make life easy for the developers. The ARCHITECT must be challenged to look for a solution outside of his comfort zone to prevent bothering the customer unnecessarily.

### **Rationale**

Now all market knowledge comes together in a single point. At the same time the insight into the product's features cumulates in this point. The focal point in which both sides converge is the place where the overview is optimal.

This position is ideal to mediate between the two viewpoints. In the end the goal of all actions is to create a product that is attractive to the customer. The customer does not care about technical details.

## Consequences

### Benefits

- Customer wish lists and techie talk become structured information in the language of the target before reaching the other party.
- By having an outside view, i.e. outside of the code, the PRODUCT MANAGER is not biased by the effort or the complexity of the technology that is necessary to create the desired features. He can come up with ideas inspired by nothing but the customer needs.
- The customer and the development team now have a clear contact that can explain the other side in their language and forwards their issues to the other viewpoint.
- Somebody watches that the intention of the customer is followed inside the executing team without the customer disturbing. Optimally, the customer can limit himself to reacting to the progress reports and does not even have to take any proactive action.
- The PRODUCT MANAGER has a clear vision of the product but at the same time is part of the team. This allows immediate decisions that are nevertheless robust and objective.
- Outlining the customer needs for the team members motivates them because they know the reason behind a request. The given information also serves as a background for the team and by that improves the quality of the feedback.
- Especially in busy times it becomes a benefit that the resources required for customer care are not drawn from development capacity. Taking your time for the customer does therefore not slow down the coding.

### Liabilities

- As the contact between the customer and the development is limited it is a challenge to assure that information reaches the other end without shifting the message.
- Laying the feature strategy in the hands of one person evokes the risk of subjective influence.

### Example

The caravan manufacturer wants his retailers to reduce support requests, order spare parts and stay in close contact with him. The company is an expert in building vehicles. Even in their marketing department there is no expert on online communication. So they hire a web agency to set up an ecommerce system and a blog. The agency's PRODUCT MANAGER discusses the customer's ideas, presents more options and proposes possible solutions. While discussing the details of the required ecommerce system to order spare parts it turns out that the caravan manufacturer rather wants his retailers to communicate with each other to exchange experience than to bother him. Nevertheless he wants to keep an eye on the issues they are concerned with. So the idea of a blog is discarded and replaced by a bulletin board. To detail the solution the PRODUCT MANAGER consults the ARCHITECT that was assigned to the project.

As the field of the wanted features narrows, the PRODUCT MANAGER starts to write down specifications to prepare the quote. He consults the ARCHITECT on open issues to get a more concrete picture of the actions and effort required.

After a set of features has been authorized, the PRODUCT MANAGER gives feedback to the developers about the features they develop and requests that the board allows to attach files to postings. He is told that this requires considerable coding effort because the software package used as a platform does not provide this feature. From discussions he had with the marketing director the PRODUCT MANAGER knows that the retailers need to exchange manuals, sketches and other documents. So in spite of the effort it causes, the PRODUCT MANAGER insists on the feature. He asks the PROJECT MANAGER to block the developer in charge of the board for some more time.

### **Related Patterns**

The PRODUCT MANAGER passes on all necessary information the PROJECT MANAGER needs to set the stage for the project. In turn he receives an organizational framework for the project.

The PRODUCT MANAGER strives for maximum customer value. By that he is a counterpart for the ARCHITECT who seeks technical quality.

Depending on project size the role of the PRODUCT MANAGER might not be executed by a single person but a team of requirements engineers.

In Scrum the PRODUCT MANAGER is replaced by the Product Owner [Kelly 2009b].

This pattern is an implementation of Coplien and Harrison's SURROGATE CUSTOMER [Coplien Harrison 2005]

Unlike a PRODUCT MANAGER, a business analyst does not analyze the needs of various customers but of the single company that employs him. It is a double inward view since this is the same company as the one that conducts the software project to satisfy the needs. [Kelly 2009c]

## 4.3 PROJECT MANAGER

### Also known as

CARETAKER

### Context

A team of several people is to work on a project. There are requirements and limited resources of time, staff and budget to implement them.

### Problem

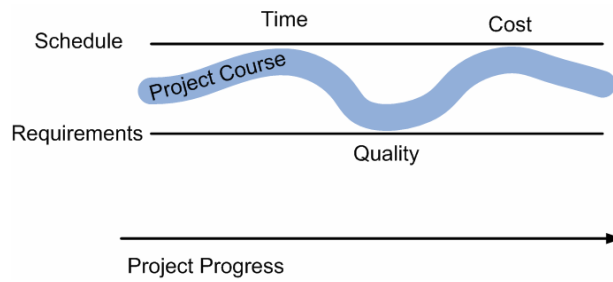
**How can you ensure that a project reaches its goal and stays within its given constraints?**

### Forces

- A feature might turn out to be much more complicated to implement than anticipated. So it will exceed the resources allocated to it. As the main costs in software projects are personnel costs, an increase of project duration commonly means a rise of costs.
- The needs of the customer might change due to changes in his market.
- As time goes by the vision of the product becomes clearer. The customer comes up with ever new ideas how to improve the features. Explicit new features would cause a request for extra budget, so the new feature requests are camouflaged as bugs or minor extensions of existing features. Nevertheless this scope creep requires time and effort to be implemented.
- The customer might be willing to cut features to meet the deadline. But depending on his situation he might instead prefer to either postpone the deadline or even increase the budget to keep scope and quality.
- During the course of the project, unforeseen events and trends might occur.
- There might be a finished third party solution available. Using it instead of developing the module yourself might save time and reduce risk but it might not fulfil the same quality standards as the main project. Licence costs might occur.
- Striving for the perfect product promotes quality but – as the Pareto Principle predicts – the higher the quality requirements are, the less efficient the development process becomes [Wikipedia 2009b].
- Motivated staff is more productive.
- Monitoring and controlling are not contributing directly to the product itself. So the capacities they consume increase the overall costs of a project without having a tangible result.

### Solution

**To ensure project success, have a dedicated person to set the stage within the constraints of the project. While monitoring the project, it is this person's job to create a good working atmosphere to motivate the team.**



**Fig. 5: Steering the project within the project constraints**

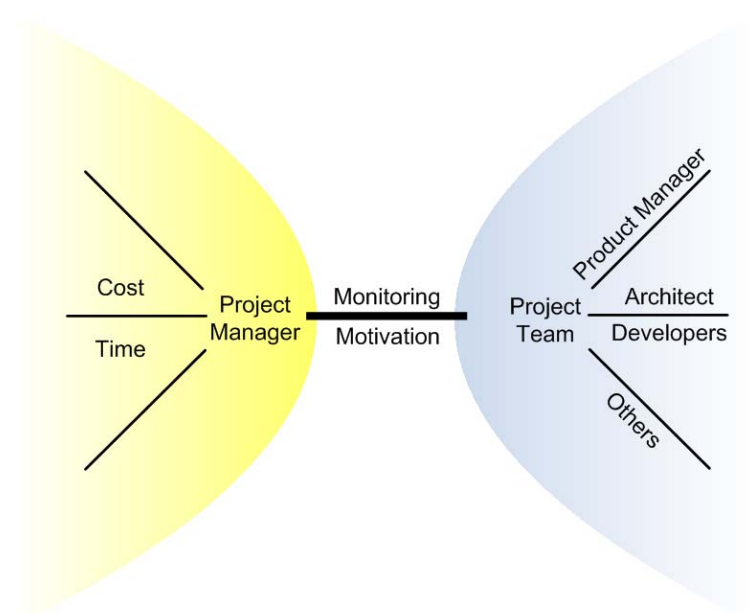
The PROJECT MANAGER creates a plan considering the given resources and defends it. He assigns tasks to the team members and keeps an eye on the project environment. He steers the resources actively, takes decisions when necessary and advocates trade-offs. The PROJECT MANAGER tracks the progress of the project. Whenever he discovers a weak spot, he takes actions to remove it.

Instruments of the PROJECT MANAGER include cutting features, postponing deadlines, extending funds, replacing tailor-made features by third-party-modules and, with caution, adding manpower [Kelly 2008].

When it comes to trade-offs between the project constraints it is important to know the customer's priorities. To further increase the options there should be some buffer in every plan and actions should be taken as early as possible.

To keep the schedule and reach an optimum in quality, the focus needs to shift during the course of the project. First it is implementing new features to perfection. Later in the project the PROJECT MANAGER needs to make sure that the developers concentration shifts towards debugging of the existing features in order to finalize the product.

### Rationale



**Fig. 6: The Project Manager shielding the constraints**



Developers want to concentrate on their work. Their focus is on the details of the code. The same is true for the PRODUCT MANAGER who focuses on customer value. So having a separate role that is not dazzled by such aspects makes sure that the other project constraints are not disregarded. Being outside of the creational process, the PROJECT MANAGER can also take care of social aspects.

## **Consequences**

### **Benefits**

- The stage is set. The developers can concentrate on coding in a productive atmosphere and the PRODUCT MANAGER has room to focus on requirements.
- The resources of a project get enough attention to be managed with overview and foresight.
- A neutral role can moderate between conflicting interests and facilitates trade-offs.
- High motivation and thus productivity in the team takes tension from the project as it saves time and money.
- Monitoring creates transparency. The positive effect active management can have on keeping other actions within the plan makes it a very profitable activity.

### **Liabilities**

- The PROJECT MANAGER is far away from the actual development. Functional quality is not his main concern. He sees features as milestones and judges them by their status.
- It is important that the PROJECT MANAGER as the ever monitoring authority is perceived as a supporter not a threat.

## **Example**

The PRODUCT MANAGER requests more resources for the bulletin board to get the picture attachment feature implemented. The task that would need to be postponed for that is the multi-language support of the parts order tool. The PROJECT MANAGER asks the PRODUCT MANAGER to decide which feature has the higher priority. As not all retailers will use the board but all of them shall use the order tool, and the multi-language support is crucial to the latter, it is prioritized. After evaluating all other options, the file upload feature is scheduled for the buffer time at the end of the development timeline. In parallel the customer is asked whether it would be ok to postpone the feature after the launch if necessary.

## **Related Patterns**

The PROJECT MANAGER stays in close contact with the ARCHITECT to learn about the effort a feature causes, to update project status and discover problems early. He also interacts with the PRODUCT MANAGER. He knows when requirements change and can be asked to cause it actively.

Coplien and Harrison have created a number of patterns on project management.

SIZE THE SCHEDULE explains how a schedule should be dimensioned.

When the COMPLETION HEADROOM gets out of control, there are means for crisis management: SOMEONE ALWAYS MAKES PROGRESS, TEAM PER TASK or SACRIFICE ONE PERSON. If they are not sufficient, the PROJECT MANAGER should better TAKE NO SMALL SLIPS to keep up motivation of customer and team.

The limit of detail to which the PROJECT MANAGER is concerned is set by INFORMAL LABOR PLAN.

To take care of the social aspects in a team, the PROJECT MANAGER should incorporate the MATRON ROLE and foster DEVELOPMENT EPISODES. [Coplien Harrison 2005]

## 4.4 ARCHITECT

### Also known as

WATCHTOWER

### Context

There is a variety of technical options to implement the requirements and a team of developers to do so.

### Problem

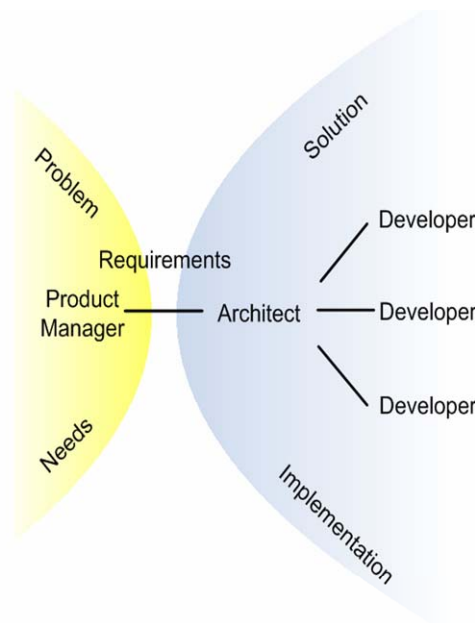
**How should the requirements be transformed into software?**

### Forces

- It is tempting to include as many features as possible in the product. This strains cost and time and it does not even necessarily improve quality.
- The development team concentrates on solving technical problems and creating features.
- A single person can only handle a limited workload. But the more hands are working on a product, the faster it becomes inconsistent, redundant, inflexible and error-prone.
- Established technologies are easy to handle, quick to implement and less buggy. New technologies might offer options that have not been available before. But using new technologies implies risks: Uncertainty, compatibility, security, introduction effort.
- Using it third party features might save but it comes with the risk to lose quality because it is not tailor-made any more.
- Using complex frameworks or third party modules creates a lot of overhead but it may save time.
- Marketing people often do not understand technical details and language.

### Solution

**Have somebody design the big picture. He is the one to check changes in the architecture for consistency with the existing system. He has to keep track of technological innovations and monitor the quality of the code.**



**Fig. 7: The Architect as interface to the outside world**

The ARCHITECT brings all his knowledge into the design. To increase the pool of experience, he orchestrates the ideas of the other developers. The ARCHITECT is to encourage the team to look for solutions off the beaten track as well.

The team has to agree on the technical platforms to use before starting to code. Also to the inside the ARCHITECT makes sure that the features implemented correspond to the requirements and are not just nice to have. He creates coding guidelines to support consistency.

The ARCHITECT's job is to break down complexity and structure issues to make the design decisions accessible for non-technical contacts. He can filter outside requests before they reach the development team.

During the project the ARCHITECT has to abstract on the development to discover when the code drifts away from the goals set and make sure it is corrected. Discussing design decisions before and during development makes sure they are incorporated by the team and not perceived as commands that just need to be obeyed to.

To keep in touch with the code, the ARCHITECT should be a lead programmer or at least at times be involved in coding.

### **Rationale**

The ARCHITECT is a watchtower in two directions. To the outside of his realm his function is to be the first to discover a change and judge whether it might be a chance or a threat. He then reacts accordingly. Only if action is required he informs the rest of the people.

Guidelines provide orientation. He also watches the inside to make sure that everything follows the rules that were set. Knowing that these tasks are done by somebody, the rest of the team is able to concentrate on their main business without fear.

Proper delegation and division of work allow growth without overloading the single person and still create a coherent product.

## Consequences

### Benefits

- Requirements and coding guidelines channel the developers ideas and also give them a basis to reflect on their work.
- New technological options may be a leap in product and development quality. The risks they might bring have been anticipated and are mitigated.
- Keeping the overview for them and providing the developers orientation inside the project gives them peace of mind for development. At the same time it does not confine the developers. Whenever they want to get an overview, they know whom to ask.
- The central hub brings clear communication channels that can be used. This makes it easier to expand the team because adding a new member does not necessarily require a new connection to every single team member.
- Shielding outside communication and introducing a hub for outside communication that understands their language frees the developers from the need to translate their detailed view into a non-technical one themselves.

### Liabilities

- A predefined environment narrows creativity for the single team member. It requires discipline.
- Having a head of development that does the communication with outside requires an extra step in communication.

### Example

Before giving a cost estimate for the parts order tool the ARCHITECT starts an investigation to compare possible third party platforms. He gathers the available experience from the developers and lets them evaluate unknown packages. On this basis the ARCHITECT and his team settle for a solution. The ARCHITECT presents it to the PRODUCT MANAGER and gives a cost estimate for its customization to the PROJECT MANAGER.

### Related Patterns

The ARCHITECT discusses technical feasibility with the PRODUCT MANAGER.

The PROJECT MANAGER requires a plausibility check for his schedules and feedback about progress during development.

The ARCHITECT as a watchtower is an implementation of Coplien and Harrison's ARCHITECT CONTROLS PRODUCT. They also deal with how the role might be split up into an ARCHITECTURE TEAM. They emphasize the necessity that ARCHITECT ALSO IMPLEMENTS to stay up to date [Coplien Harrison 2005].

Depending on the team size the outside view may be supported by a dedicated technologist who researches new technologies. [Völter Kircher 2008b]

## 5 Acknowledgements

My thanks go to my shepherd Michael Kircher for his invaluable domain and pattern expertise and the right hints at the right time. I would also like to thank Allan Kelly for inspiring me to make the first steps towards this paper and the participants of my workshop at EuroPLoP 2009, especially Andreas Rüping, for their incredibly helpful comments.

## 6 References

- Coplien, James O. and Neil B. Harrison 2005. *Organizational Patterns of Agile Software Development*. Upper Saddle River, NJ: Pearson Prentice Hall.
- Kelly, Allan 2008. *On management*.  
<http://www.allankelly.net/static/writing/OnManagement/OnMngm1-Constraints.pdf>.  
2008-07-03 (Last accessed 2010-1-21).
- Kelly, Allan 2009a. *On Management #4: Caveat Emptor*.  
<http://www.allankelly.net/static/writing/OnManagement/OnMngm4-CaveatEmptor.pdf>.  
2009-02-20 (Last accessed 2010-1-21).
- Kelly, Allan 2009b. *On Management #5: The Product Manager role*.  
<http://www.allankelly.net/static/writing/OnManagement/OnMngm5-ProductManager.pdf>. 2009-04 (Last accessed 2010-1-21).
- Kelly, Allan 2009c. *On Management #6: The Business Analyst's role*.  
<http://www.allankelly.net/static/writing/OnManagement/OnMngm4-CaveatEmptor.pdf>.  
2009-06-08 (Last accessed 2010-1-21).
- Schwanninger, Christa, and Michael Kircher 2009. *Patterns for Product Line Engineering*. In: Proceedings of the EuroPLOP 2009 conference, Irsee. In: CEUR-WS online repository, <http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/>.
- Völter, Markus and Michael Kircher 2008a. *Roles in Software Engineering I. Episode 110*. <http://www.se-radio.net/podcast/2008-09/episode-110-roles-software-engineering-i>. 2008-09-18 (Last accessed 2010-1-21).
- Völter, Markus and Michael Kircher 2008b. *Roles in Software Engineering II. Episode 112*. <http://www.se-radio.net/podcast/2008-09/episode-112-roles-software-engineering-ii>. 2008-09-28, (Last accessed on 2010-1-21).
- Wikipedia 2009a. *Project Management Triangle*.  
[http://en.wikipedia.org/wiki/Project\\_management\\_triangle](http://en.wikipedia.org/wiki/Project_management_triangle). 2009-03-22 (Last accessed on 2010-01-21).
- Wikipedia 2009b. *Pareto Principle*. [http://en.wikipedia.org/wiki/Pareto\\_principle](http://en.wikipedia.org/wiki/Pareto_principle) 2009-03-22 (Last accessed on 2010-01-21).

# Patterns for Strategy Management of Technology Entrepreneurship and Innovation MSc Program

---

*Yanka Todorova<sup>1</sup>, Petko Ruskov<sup>2</sup>, Elissaveta Gourova<sup>3</sup>, Mark Harris<sup>4</sup>*

<sup>1</sup>*Sofia University, Faculty of Mathematics and Informatics, Sofia, Bulgaria, [todorova.yana@gmail.com](mailto:todorova.yana@gmail.com)*

<sup>2</sup>*Sofia University, Faculty of Mathematics and Informatics, Sofia, Bulgaria, [petkor@fmi.uni-sofia.bg](mailto:petkor@fmi.uni-sofia.bg)*

<sup>3</sup>*Sofia University, Faculty of Mathematics and Informatics, Sofia, Bulgaria, [elis@fmi.uni-sofia.bg](mailto:elis@fmi.uni-sofia.bg)*

<sup>4</sup>*Intel GmbH, [mark.harris@intel.com](mailto:mark.harris@intel.com)*

## Abstract

The paper introduces three patterns for use by university managers and lecturers for creating of Master of Science (MSc) program in engineering and science faculties, developing entrepreneurial skills:

- TECHNOLOGY ENTREPRENEURSHIP AND INNOVATION (TEI) CURRICULA – provide a valuable and computational set of courses to guarantee the competence of graduate students in the turbulent environment;
- TEI MSc PROGRAM – create the road map for successful implementation of the technology entrepreneurship graduate education;
- TEACHERS TEAM – when the human capital is scarce, focus the stress on attracting and retains the best professionals in the fields.

The authors experimented with the MSc graduate Technology Entrepreneurship curriculum design and implementation of the program in the university master level teaching, and used a mix of methods and observation techniques to assess its feasibility and applicability for developing the winning strategy.

**Keywords:** *Patterns, Technology Entrepreneurship Curricula, Education Strategy*

## 1 Introduction

Providing citizens and employees with a new set of competences turned out to be in the middle of all changes in public life and economy. Therefore, Information and Communication Technology (ICT) subjects were introduced in school curricula, a lot of initiatives were implemented for educating trainers, establishing computer labs in schools and connecting them to the global networks. While the e-skills necessary for all European citizens are generally solved by educational systems, a lot of challenges remain for building the competences required by industry. In the last few years the issue of e-skills was put on the agenda at different fora all around Europe – ICT Skills Monitoring Group, Career Space industry-led initiative, European e-Skills Forum, e-Skills and e-Learning expert groups at EC, ICT Task Force, etc. The general conclusion coming out of these is that present employees need more than just technical skills and knowledge. The need to equip employees with interdisciplinary skills, combining e-skills with entrepreneurial competences, innovativeness and creative thinking, turned out to be among the highest challenges for businesses, educational institutions and policy makers nowadays [5].

The Entrepreneurship and innovation education process is a management process, not an individual characteristic [3]. As a process it can be described, modeled, analyzed, understood and taught. The core elements of the entrepreneurship process are the evolution and identification of a business opportunity and the recruiting and aggregation of the necessary resources (technology rights, people, money) to pursue the opportunity [16].

Some universities tend to be rather conservative institutions, not eager to change. Therefore, developing a MSc program is not something that happens fast and often. The introduction of major changes within a university faculty, as elsewhere in the world, starts by changing the organizational culture and adopting a new way of thinking and philosophy that will guide staff toward new faculty goals [12, 15, 18].

As a result of analysis and observation of the practice in creating an entrepreneurship program, the paper describes TEI Education Program as a set of patterns [11]. This MSc education follows closely the new technologies change and their impact on industrial practice. In order to remain competitive, scientific and engineering faculties should apply innovation in all its types and practices – organizational, technological, process, etc. They need to closely monitor labor market needs and establish close cooperation with the main users of their services in education and research [13].

The paper introduces three patterns for use by university managers and lecturers for creating of Master of Science program in engineering and science faculties.

### **Audience**

Many innovative universities are trying to reengineer their curricula in response to the dynamic global environment and the socio-economic changes. Only some faculty managers have the luxury of stopping or delaying the educational processes at the universities in order to redesign them. In present dynamic world, no time remains for such stop-overs, and the costs would be dreadful.

These patterns systematize the long-term education management experience and observation of the authors and combine it with recent strategic management approaches. These patterns can be useful for scientific and technology faculties' managers for design, development and implementation of entrepreneurship education in engineering MSc program.

## **2 The Patterns**

The benefits of graduate education practice are communicated through the concept known as a pattern. The concept of patterns was widely described in many industries, papers and books [2, 4, 7, 9, 10, 12, 19]. There are many patterns that deal with specific issues in the strategic management loop of the education and training process - creation and execution of a program for technology entrepreneurship and innovation education programs. This paper presents the following patterns (Fig. 1):

- TEI MSc PROGRAM;
- TEI CURRICULA;
- TEACHERS TEAM.



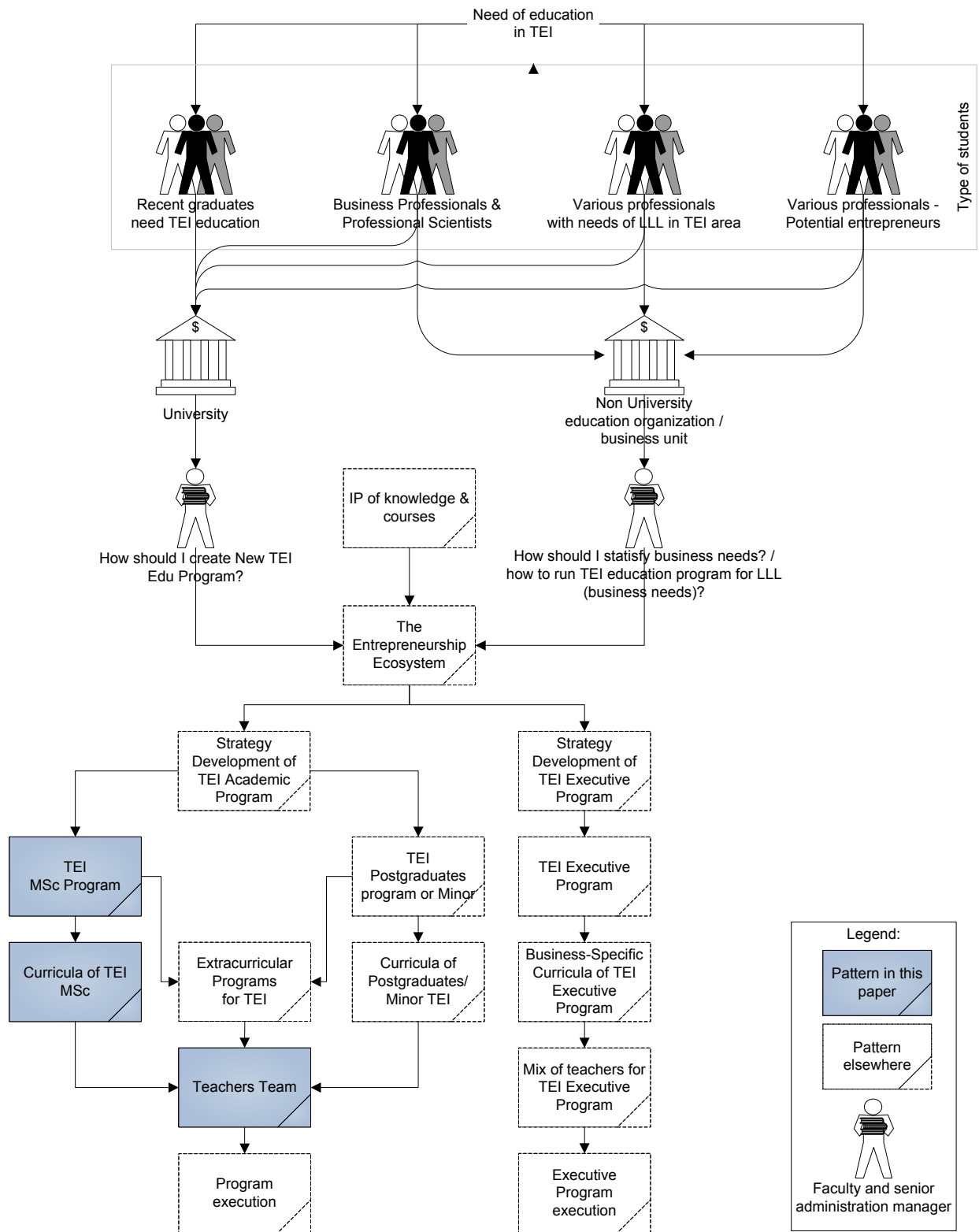


Fig. 1 Development of TEI education program pattern sequence

## 2.1 TECHNOLOGY ENTREPRENEURSHIP AND INNOVATION MSc PROGRAM:

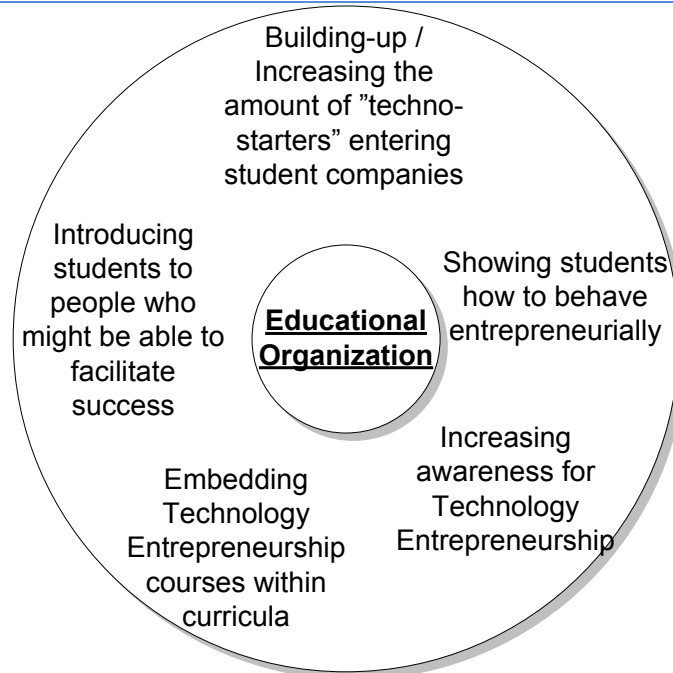


Fig. 2 TEI education program

### 2.1.1 Context

Students are the main target of any educational program. They could be recent graduates, business professionals and professional scientists, professionals with needs of life-long learning (LLL), or various professionals - potential entrepreneurs. They all might need TEI education (Fig.1).

New technologies and organizational innovations affect the transmission and generation of knowledge. It is generally accepted that the collaboration between businesses and universities can accelerate the rate of innovation and the economic growth in key markets around the world and to facilitate overcoming the following difficulties [6]:

- weak links between science and businesses
- not sufficient measures to develop innovation infrastructure, support services, technological brokerage, intermediary services, etc.
- not involvement of university students in scientific and technological activities
- low innovative culture and weak innovative culture of businesses
- low level of investment in new products and processes
- slow implementation of measures and not systematic and transparent evaluation, etc.

At the same time, it is widely discussed that future employees need not only technology skills, but they need also a set of interdisciplinary skills which traditional education programmes do not provide. There is an obvious need for educational programmes focusing on various scientific disciplines and across traditional university faculties [5].

A key reason for describing a TEI MSc pattern is that strategy management, learning design experience and curricula of the MSc program education can actually be reused. Reinventing the wheel for every new MSc program makes only the faculty increasingly unmanageable and slows down the programme delivery.

## 2.1.2 Problem

**What actions should a decision maker undertake in the process of developing a new TEI MSc program in its own university / department?**

## 2.1.3 Business value.

The TEI MSc PROGRAM pattern allows academic managers to improve the process of education and to grow the efficiency and efficacy of the education.

## 2.1.4 Forces

There are many forces in the area. Below are summarized the most important ones.

*Force 1: Meeting labour market and student needs:* On the one hand, universities provide strong education for students with a specific major, e.g. technical/engineering, scientific, etc. and as a result, students are well skilled in technology. But on the other hand, the students need to commercialize their knowledge. Besides, companies are seeking employees not just with technology skills, but most demanded are employees with interdisciplinary skills and knowledge.

*Force 2: Bridging the gaps of the university environment and the Entrepreneurship Ecosystem.* Universities need to establish stronger links with businesses, but also to focus on research and teaching. Unfortunately, their services not always are adequate to business needs. This leads to the fact that very often university environment is not aligned with entrepreneurship ecosystem. Besides, universities experience serious problems closely related to the general socio-economic and research environment [13]:

- Migration to industry of highly skilled professionals
- Insufficient research funding
- Lack of sufficiently stimulating research environment
- Lack of stable and multiple bridges between research, development, education and training
- Lack of traditions in university-industry-government cooperation.

*Force 3: Conservative and separated university environment:* Universities are very conservative institutions, not eager to change. Every faculty has a strong knowledge and experience in a specialized field(s). In order to teach management and entrepreneurial skills to professionals in a technical field, there is a need of strengthening faculty's capacity, and integrate researchers and experts in different fields than its core research and teaching areas. The faculty should undertake several activities in order to determine its future strategy in research, technology development and innovation (RTDI) [13].

## 2.1.5 Solution

**Match the MSc programme to local educational ecosystem and deliver benefits to the stakeholders – students, business, government and community.**

A project for development of TEI MSc program should be launched and conducted by a faculty team leaders focused on the design and implementation of a new academic educational program.

Some efforts are needed to change the stakeholders' mindset and the model of thinking about MSc degree education. There is a need before launching the program to:

- Increase the awareness for TEI ;
- Increase the number of technology entrepreneurship courses for researchers of the Faculty, and for all stakeholders.

Faculty should gain and enhance its knowledge in the area of TEI and participate in research projects in various areas closely linked to the new program core subjects. University/departments should establish stronger links with businesses and government. Students should have a chance for learning by doing on-the-spot.. Here the strong academia-industry collaboration could facilitate the MSc program as a whole.

**Specific objectives** for the TEI educational program can be summarised as follows:

- Provision of a mix of competences needed for future entrepreneurs, including technology, business and personal/organisational competences
- Providing students with practical experience and self-confidence in a risky and dynamic environment
- Showing students how to behave entrepreneurially
- Introducing students to people who might be able to facilitate success

After launching a MSc programme on TEI, it is essential to consider some follow-up activities:

- Embedding TEI Education courses within curricula of all university faculties;
- Building-up / Increasing the amount of "techno-starters" entering student companies;

### 2.1.6 Consequences

Studying entrepreneurship could be an important positive force on students' career development. Whether students are studying entrepreneurship with the definite intent of turning into an entrepreneur or to be a business executive or a more informed stakeholder in general, they will all need entrepreneurial skills, once entering business settings.

Many students will come across of working closely with entrepreneurs in new and small enterprises. These enterprises will have to manage innovation for long-term growth and compete with other similar enterprises.

### 2.1.7 Variations

Depending of the Entrepreneurship ecosystem, specific strategy can be developed and implemented for building TEI education program. The process of particular adoption of patterns in Fig. 1 can lead to variations in adoption of TEI MSc PROGRAM.

The program can propose a teaching curriculum, classroom and external exercises on site, for teaching the entrepreneurship to engineers and scientists, creating innovative business people with cross-disciplinary skills, technical expertise, and the ability to grasp market opportunities.

### 2.1.8 Examples

The awareness on the importance of entrepreneurship and management skills was build within several years work at European RTD projects among researchers of the Faculty of Mathematics and Informatics (FMI). The awareness on the business needs for interdisciplinary competences of their future employees resulted in launching in the autumn of 2007 by FMI a new MSc program on TEI. The program turned out to be very successful as the students in school year 2008/2009 increased more than twice.

The Bulgarian format of TEI graduate program has been created due to the transfer of knowledge and support provided by the Intel and UC Berkeley initiative. Due to that initiative a new way of thinking and working was introduced. The teaching methods changed and became more interactive, applying the Intel-UC Berkeley education patterns. New different courses about TE were introduced, in line with the UC Berkeley curriculum.

Only a year after its launching, the program became the most demanded MSc program at Sofia University. Tangible signs of success of this program are already observed as some of the graduates of this program started their own businesses as a result of it [16].

### 2.1.9 Related work & Sources

Galabova L., Ruskov P., "Analysis of the Status and Opportunities for further Development of Bulgarian Entrepreneurial Education in Universities, Proceedings of the International Conference for Entrepreneurship, Innovation and Regional Development ICEIRD 2008, Skopje&Ohrid, Macedonia 8-12 may 2008, pp. 237-245 [20].

## 2.2 TECHNOLOGY ENTREPRENEURSHIP AND INNOVATION CURRICULA

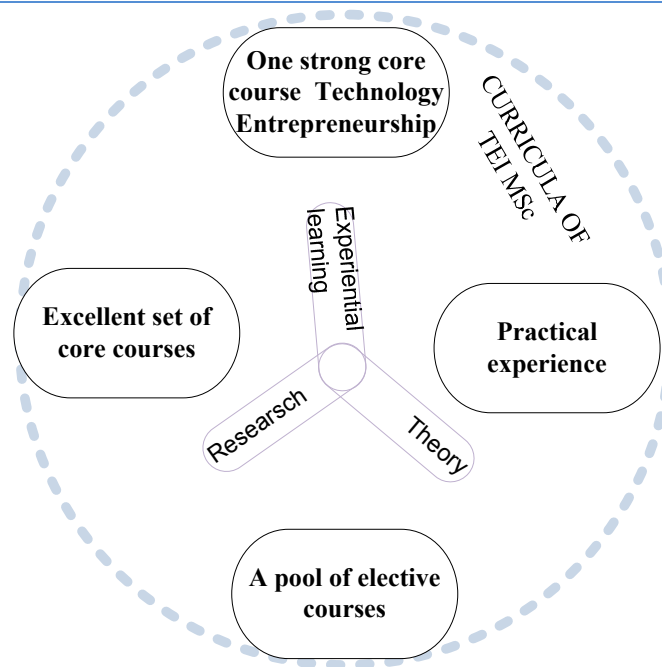


Fig. 3 Curricula of TEI MSc program

### 2.2.1 Context

One of the biggest challenges facing every economy is how to develop more skilled and productive workforce. Entrepreneurship Education is a key part of the solution to several pressing policy challenges, including university dropout rates, workforce readiness, and country's economic competitiveness. Countries assumed entrepreneurship as a means of growing their markets, and creating jobs in order to be more competitive and with stable economy.

As a whole, engineers and scientists do not know enough about entrepreneurship and innovation in order to successfully translate an idea into business. At the same time, current practice needs a mix of technology and interdisciplinary skills. Therefore, the curricula in TEI shall provide to professionals with scientific and engineering background knowledge in economics and innovation. Its goal is to teach potential entrepreneurs how to commercialize their knowledge, and turn it into new technologies and innovations.

## 2.2.2 Problem

**What type of competences need to gain scientific and engineering graduated students and how to ensure them with the TEI curricula?**

## 2.2.3 Business value.

The pattern helps university managers and lecturers to enrich the quality of courses content. The pattern also allows flexibility of the curricula courses. It addresses the time to market of the program and decreases cost of development.

## 2.2.4 Forces

*Force 1: Motivated Leadership:* There are many basic curricula which universities have created in different scientific fields. TEI is an interdisciplinary program and the faculty leaders need best practices in order to be able to create it faster according to the local circumstances. The personal motivation and dedication of the leaders is essential for developing the curriculum and overcoming all possible barriers as well as encourage others to grasp the ideas of the new curriculum.

*Force2: Determining proper mix of courses.* The curriculum should find the appropriate balance between the different courses in order to ensure the required technology, business and personal/organizational competences of the students. At the same time, the courses need to be developed with a focus on harmony between theory and practice.

*Force 3: Ensuring potential teachers.* By designing the curriculum it is of utmost importance to consider potential teachers and bridge the gaps between different faculties. Finding motivated talented teachers is an important factor for the success of the new MSc program and the curriculum should be designed taking the best available expertise.

*Force 4: Involving stakeholders in the education process:* It is well-known that contemporary society and economy need fast implementation of innovation and commercialization of research results and new knowledge, but traditional curricula do not provide the necessary interdisciplinary competences and risk taking abilities. In order to face this problem it is important that the curriculum involves at an early stage stakeholders as guest lecturers in order to share their experience, provide guidance to students as well as opportunities for internship.

*Force 5: Ensuring a critical mass of potential applicants:* Students in technical subjects need new state-of-the-arts competences in the area of TEI, but such curricula exist only in a small number of universities. There is a need by designing a new curriculum to consider how to reach potential students and encourage them to choose it.

*Force 6: Challenges with IP of knowledge and courses.* Universities own the intellectual property rights (IPR) of their knowledge. There is worldwide ranking of the universities and their products. At the same time, there are many issues related to using and teaching entrepreneurial knowledge, inviting guest speakers - professors from other universities and real leaders - entrepreneurs, and reusing their knowledge.

## 2.2.5 Solution

**Investigate the curricula best practices of world leading universities (AS\_IS), benchmark the leaders in implementing the desired solution (TO\_BE) .**

Each new initiative needs strong leadership in order to succeed. This is especially important for a new educational curriculum in order to overcome resistance and administrative barriers and motivate followers.

By designing the curriculum should be considered the following:

- Developing one strong first core course with two parts – Technology Entrepreneurship and Technology Entrepreneurship in IT. An excellent set of four to five core courses focused on provision of technology, business and organizational competences. A pool of elective courses adding specific competences for meeting individual needs of the students;
- Theory, research, and experiential learning should merge into one another within the education process and extra curriculum courses by ensuring a proper mix of theory, interactive and practical content;
- Different and complementary teaching materials should be delivered for the courses;
- The teachers team should be formed following the TEACHERS TEAM pattern.
- Engaging the stakeholders as lecturers – leading managers with practical experience in entrepreneurship .
- The new MSc program and the core courses in its curriculum should be widely advertised, as well as the expected results in competence building and the new approach for more practical orientation and stakeholders’ involvement in the educational process.

Finally, the ENTREPRENEURSHIP ECOSISTEM should be considered and along with its INTELLECTUAL PROPERTIES (IP) OF KNOWLEDGE AND COURSES , the decision-maker (faculty and senior administration manager) should carefully design the curriculum of the MSc program on TEI.

### 2.2.6 Consequences

Faculty leaders have knowledge and best practices for rapid design of new curricula in TEI. Applying the pattern will allow universities to be more competitive.

Graduated students in TEI will fill in the gaps in the fast changing market economy and will speed up new innovative technology products or processes commercialization.

Entrepreneurs educated on this curriculum will have more knowledge and practical experience in decision making and risk taking. This leads to faster and competitive implementation of innovative technologies, products or processes.

The time limitations, due to the large competition at higher students education, require fast reengineering of curricula and creating of new curricula in TEI. But applying the pattern will shorten the time and decrease the efforts needed for development of TEI curricula.

### 2.2.7 Variations

Depending of the needs of stakeholders, the background of the student, and the type of the university (engineering, scientific, economic, etc.), this pattern can be adapted with various specific compulsory or elective courses included in the curricula.

### 2.2.8 Examples

#### **Example 1.**

As an effort of many of the full-time and adjunct faculty members within the Lester Center for Entrepreneurship at UC Berkeley, the Entrepreneur Curriculum was developed.

The Entrepreneur Curriculum is projected to be a full academic year of courses. It can be offered at the last year of an undergraduate program or in a master’s program. It is best implemented where there are local entrepreneurs who can mentor students or be guest speakers and provide insight into the real experiences. This curriculum is intended to target

high technology start-up companies and will appeal to entrepreneurs in both, the engineering and business disciplines.

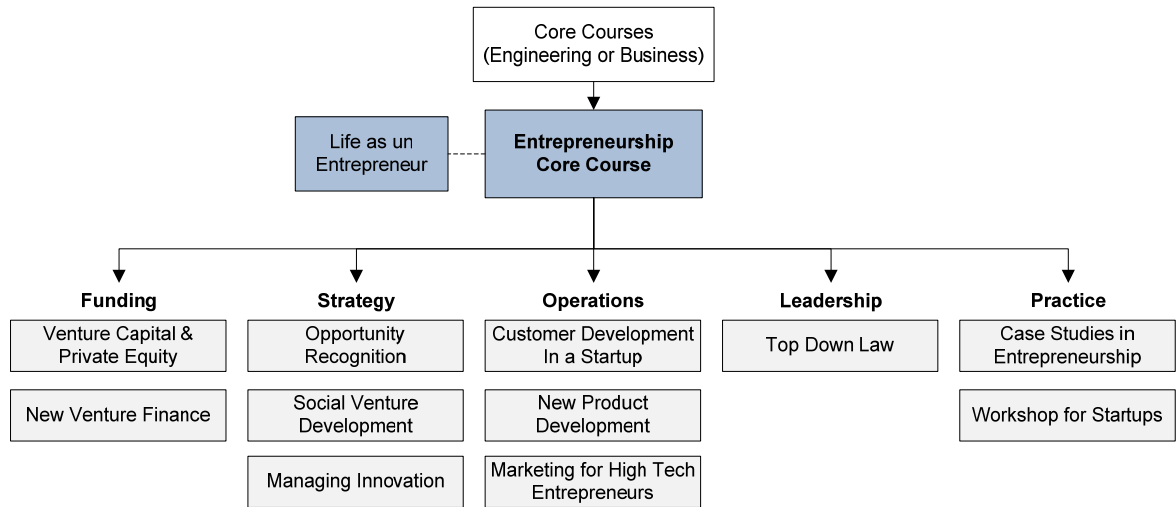


Fig. 4 Entrepreneurship Curriculum at Lester Center for Entrepreneurship and Innovation, Haas School of Business, UC Berkeley

**Example 2**– Adapted curricula for Technology Entrepreneurship and Innovation at Faculty of Mathematics and Informatics, Sofia, Bulgaria

Following many discussions in Sofia concerning preparation of curriculum and structure of the Master of Science program “Innovation and Technology Entrepreneurship”, the main topics were selected [19]. As in the curriculum of the Berkley University entrepreneurship program, it is identified:

- One core course with two parts – Technology Entrepreneurship and Technology Entrepreneurship in IT (10 academic credits, 120 hours lectures and practical seminars), based on the curricula from Intel of which ~30% derived from the original UC Berkeley “core” curricula, and 70% were created based on the needs of Western and Central Eastern Europe.
- Four basic courses – Innovation Management (7,5 academic credits, 60 hours lectures and practical seminars), Financial management and venture capital (7,5 academic credits, 60 hours lectures and practical seminars), Strategic management (5 academic credits, 60hours lectures and practical seminars) and Marketing management (5 academic credits, 60 hours lectures and practical seminars).
- A pool of elective courses each by – 5 academic credits, 60 hours lectures and practical seminars: Commercial legislation in Bulgaria and Internet and law; Organizational Behavior; e-Business, e-Governance; Knowledge Management, Customer Relationship Management; Project management; Project financing of innovation, Entrepreneurship (Student Company - practical simulation of venture creation - localized for Bulgaria), etc.

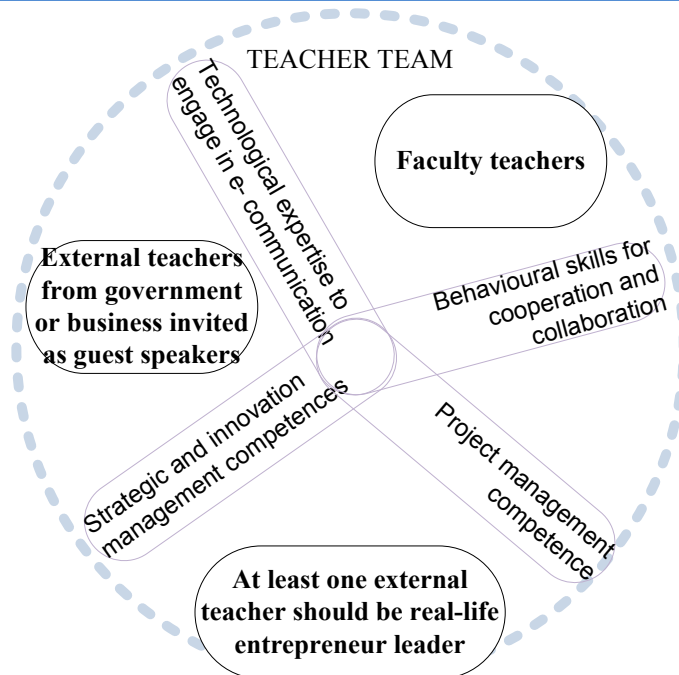
### 2.2.9 Related work & Sources

The curriculum process is described in book:

Engel J. , Charron D. (2006), Technology Entrepreneurship Education, Theory to Practice, Lester Center, UC Berkeley 2006.



## 2.3 TEACHERS TEAM:



Synergy mix of faculty and external teachers

Fig.5 The teachers team

### 2.3.1 Context

Teachers are not just important contributors to the education process, but they have become for their universities the most valuable capital. Capable professors are not just hard to find, but faculties are waging a war for talents. Teaching experts do not just bring their background and experience to their classes, but contribute their human capital. A key reason for describing a TEACHERS TEAM pattern is that curricula design experience and execution of the MSc program can actually be effective and efficient only when it is completed with the most talented professors. Nowadays, the greatest risk faced by teachers, especially those in Technology Entrepreneurship area for which the latest technical knowledge is their most critical human capital, is not the loss of a job, but rather the devaluation of the assets and the hard satisfaction of the smartest students' demands.

It should be taken into account how to motivate and keep employees, and the following human capital value chain provides an insight into this issue: [1]:

“1. In addition to being fairly compensated, people place high value on:

- Being in an environment where they can grow and learn and advance
- The managerial skills/abilities of their immediate supervisor
- Being treated fairly, appreciated and acknowledged
- Doing work that makes a contribution

2. These determinants of employee satisfaction drive employee retention

3. The retention rate among key employees drives customer satisfaction

4. Customer satisfaction drives customer retention

5. Customer retention drives profitability and other measures of financial performance including total stockholder return.”

### 2.3.2 Problem

**What kind of teachers should faculty managers recruit, source and select to form teacher's team for technology entrepreneurship education?**

### 2.3.3 Business value

TEACHERS TEAM pattern allows academic managers as an owner and investor of human capital to effectively manage the professors and the positions, and measure this asset where it will generate the highest return of investments. The pattern also increases the teachers' satisfaction because a more transparent process is easier to share and communicate.

### 2.3.4 Forces

These are common problems typical for academic human capital. The most important driving forces include:

*Force 1: A "critical mass" of talented teachers:* There is a big gap between existing teachers and a need to find the right faculty, to attract top teaching and research talent. On the one hand, there are some candidates for faculty positions, but on the other, universities experience serious problems closely related to capable teachers [13]:

- Lack of sufficiently stimulating research environment;
- Migration to industry of highly skilled professionals and of experts;
- Increase diversity of hiring
- Lack of youth applicants

*Force 2: Team strategic management:* Need of dynamic strategy planning and implementation and real measurement in order to take competitive decisions and to lead changes. The academy is tradition-keeping organization and the academic culture is very conservative. But the dynamic globalization and the market economy open the academic processes and change the capabilities of the professors. The brain drain requires creating and updating a strategy for learning and growing in the universities environment.

*Force 3: Teachers mind set change management:* Fundamental shifts in the way teachers think — and speak — about working life management. The Internet, professional and social networks are strong tools for dissemination of information and practices. At the same time, ICT introduce big changes in the teaching environment – new methodologies, new tools and practices which face teaching staff with enormous challenges to change and adapt rapidly.

*Force 4: Synergy mix of faculty and external teachers:* Every faculty has strong knowledge and experience in a specialized field(s). But in order to teach management and entrepreneurial skills to professionals in a technical field, there is a need of strengthening faculty's capacity. The teachers abilities associated with e-literacy, innovation, and entrepreneurship are expected to be crucial for strategic success, but are considered insufficient in today's professors.

### 2.3.5 Solution

**Model and develop a dynamic strategy loop cycle about human capital development and execute it as quick as possible. Do real measurement to take competitive decisions and to lead changes.**

University should have a synergy mix of faculty and external teachers. External teachers might be invited as guest speakers, part time lecturers etc. At least one external teacher should be real-life entrepreneur leader.

Faculty team members in the new TEI program must have:

- Technological expertise to engage in e-communication;
- Project management competence that extend to handling joint ventures programs and strategic partnerships;
- Strategic and innovation management competences and wide knowledge to understand present technology environment and business demands;
- Behavioural skills that permit successful cross-functional, inter-company, and multi-regional cooperation and collaboration.

### 2.3.6 Consequences

Studying entrepreneurship could be an important positive force on students' career development. Whether students are studying entrepreneurship with the definite intent of turning into an entrepreneur, or of growing to be a business executive or a more informed stakeholder in general, once students enter into the business, they will all need entrepreneurial skills.

Many students will come across of working closely with entrepreneurs in new and small enterprises. These enterprises will have to manage innovation for long-term growth and compete with other similar enterprises.

### 2.3.7 Variations

Depending on the ecosystem, this pattern can be adapted with various specific positions, full or part time. Different forms of cooperation between university, business and governance can be implemented.

### 2.3.8 Example

MSc program TEI in Information Technology at FMI is very successful as the students applying for the second year after launching increased more than twice and new candidates for teaching position appears. The position for PhD education becomes more attractive and applicants grow.

Due to the synergy of the teaching team a new way of thinking and working was established. The TEI team leaders succeed to attract the best teachers from leading Bulgarian universities and well-known experts from business and governance. The teaching methods changed and became more interactive. Teachers wrote many papers and share positive experience between Bulgarian and European Technology Entrepreneurship educators.

### 2.3.9 Related work & Sources

PATTERN FOR GRADUATE STUDENT COMPANY LIFE CYCLE (Petko Ruskov, Milena Stoycheva, Yanka Todorova 2009) [17].

## Conclusion

The pattern thinking and approach are well suited to encourage university managers to expand their perspectives, and also to communicate and to develop the strategic skills and behavior required. Patterns idea is an entirely new approach for TEI education and supporting a 21st century human capital competences. The patterns way of managing the lifecycle and the presented techniques for identifying opportunities and creating action plans in order to ensure education efforts, is proofed to be successful. TEI MSc PROGRAM; TEI CURRICULA; AND TEACHERS TEAM patterns answer many questions concerning graduates' education.

The value of the patterns described in this paper is in integrating knowledge, experience, best practices and tools within one new strategy live-cycle for master program technology entrepreneurship. The described patterns show how to reuse design and implementation of the TEI MSc program.

In the future authors intend to continue the description of more patterns of MSc education in Technology Entrepreneurship and Innovation.

## Acknowledgements

The authors express their deep appreciation to Dr. Michael Weiss for shepherd this paper to EuroPLoP 2009. His contributions were just superb.

The work on this paper has been supported by:

- Intel and UC Berkeley Project <entrepreneurship.berkeley.edu> that is funded by the Intel. <<http://www.intel.com/education/highered>>
- UC 7FP Project SISTER: “Strengthening the IST Research Capacity of Sofia University”, Grant agreement no.: 205030

## References

1. Davenport T., The Human Capital Metaphor: What's in a Name LineZine, <http://www.linezine.com/7.2/articles/tdthcmwian.htm>
2. Elssamadisy A. 2007, Patterns of Agile Practice Adoption, Crafting an Agile Adoption Strategy, InfoQ, 2007. Engel J., Charron D., (2006), Technology Entrepreneurship Education, Theory to Practice, Lester Center, Berkeley 2006.
3. Engel J., Charron D. (2006), Technology Entrepreneurship Education, Theory to Practice, Lester Center, UC Berkeley 2006.
4. Galic M., and all, Academic Edition: Applying Patterns Approaches Patterns for e-business Series, IBM Redbooks publication, 2007, SG24-7466-00.
5. Gourova E., Antonova A. Nikolov R.. Building skills for the knowledge society, *Proc. of Third International scientific conference 'Computer Science'*, Istanbul, 12-15 October 2006, pp.107-113
6. Gourova E., Antonova A., Todorova Y., *Industry-academia collaboration in Bulgaria – the case of Sofia University*, Proc. of International Conference for Entrepreneurship, Innovation and Regional Development ICEIRD 2009, Thessaloniki, 24-25 April 2009, pp. 157-166
7. Kelly A. 2005a, Business Strategy Design Patterns, The Porter Patterns, <http://www.allankelly.net>
8. Kelly A. 2005b, Business Strategy Patterns for Technology Companies, The Porter Patterns, <http://www.allankelly.net>
9. Kelly A. 2005c, Business Strategy Patterns for the Innovative Company, The Porter Patterns, <http://www.allankelly.net>
10. Kelly A. 2008, Business Patterns for Product Development, (EuroPLoP 2008), <http://www.allankelly.net>
11. Masters of Science Degree Program of Sofia University [http://www.fmi.uni-sofia.bg/education/magisters/informatika-07-08/elektr\\_biz\\_el\\_uprav.pdf](http://www.fmi.uni-sofia.bg/education/magisters/informatika-07-08/elektr_biz_el_uprav.pdf)

12. Miles I., Patterns of innovation in service industries, IBM Systems journal, vol.47, No 1, 2008, pp.115-128
13. Nisheva, M., E.Gourova, P.Ruskov, Y.Todorova, A.Antonova, 'Strategic framework for IT education and research at Sofia University', *International Journal of Education and Information Technologies*, Issue 4, Volume 2, 2008, pp. 213-225
14. Radenski A., Patterns for active e-learning in CMS environments. *Serdica Journal of Computing*, Vol. 2, No 3, 2008, pp. 277-294.
15. Robertson B., Sribar A., *The Adaptive Enterprise, IT Infrastructure Strategies to Manage, Change, and Enable Growth*, Intel Press IT Best Practices Series, 2002.
16. Ruskov P., Harris M., Todorova Y., Strategic Model for Master of Science program "Innovation and Technology Entrepreneurship", 3rd Balkan Conference in Informatics (BCI'2007), 27-29 September 2007, Sofia, Bulgaria, ISBN:978-954-9526-41-7, vol.1, pp. 501-512.
17. Ruskov P, Stoycheva M., Todorova Y., Pattern for Graduate Student Company Life Cycle, EUROPLoP 2009, Bavaria, Germany (under review)
18. Strategy as Practice: Research Agenda, <http://www.s-as-p.org/agenda.htm>
19. Ogrin M., *Mashup Patterns*, Addison Wesley, 2009, 9780321579478.
20. Galabova L., Ruskov P., "Analysis of the Status and Opportunities for further Development of Bulgarian Entrepreneurial Education in Universities, Proceedings of the International Conference for Entrepreneurship, Innovation and Regional Development ICEIRD 2008, Skopje&Ohrid, Macedonia 8-12 may 2008, pp. 237-245.

# Performance of Open Source Projects<sup>1</sup>

Michael Weiss

Carleton University, Ottawa, Canada

[weiss@sce.carleton.ca](mailto:weiss@sce.carleton.ca)

## 1 Introduction

The patterns in this paper describe open source development practices from a performance perspective. In product development, performance is measured in terms of the *time* it takes to develop a software product, the resulting *quality* of the software, and the *cost* of development. These dimensions are in tension with one another. Since improving performance has side effects, we also need to include the impact on other dimensions than performance (eg risk, trust) in our discussion of the practices.

The audience for these patterns are developers and project managers, who are thinking about adopting an open source development approach. The practices documented in the patterns are derived both from the literature on open source development, and from the author's experience as contributor to several open source projects. The author also had the opportunity to observe a large open source project.

## 2 Patterns

An open source project must start with a *Credible Promise* (3), otherwise it will fail to attract developers. To encourage other developers to build on it, a project needs to offer sufficient functionality or equivalent.<sup>2</sup> For the open source project to grow fast, it is necessary to build up momentum quickly. Rather than waiting for a polished release with full functionality, the owners of the project should aim to deliver a stream of *Frequent Releases* (5). This allows them to learn from users how the code is used; advanced users will discover and fix problems for you, resulting in higher code quality.

In order to achieve a critical mass of functionality early in the life of a project, the project should *Build On the Shoulders of Others* (7) by integrating components developed by others. Reusing components with proven functionality also allows a project to reach a higher level of quality earlier.

To keep others - users and developers of components - engaged in your project you need to maintain an *Open Dialog* (9). Development of the system needs to be carried out in the open. At the same time, not all developers participating in a project may pursue the same goals. Developers need to be able to explore different ways of evolving the system in parallel. Hence, *Parallel Development* (11) provides a mechanism for coordinating between stable and experimental releases of the same project.

---

1 Copyright is retained by author. Permission is granted to Hillside Europe for including in the CEUR archive of conference proceedings and for the Hillside Europe website.

2 For some projects the biggest attraction for other developers is not the functionality (all that exists may be a specification), but the reputation of the project founders (Fogel, 2006).

A map of the patterns showing their relationships is shown in Figure 1. Links between patterns X and Y should be interpreted as “after pattern X you may also use pattern Y”. Patterns with a star (\*) are planned patterns. Thumbnails of these patterns can be found in Appendix A.

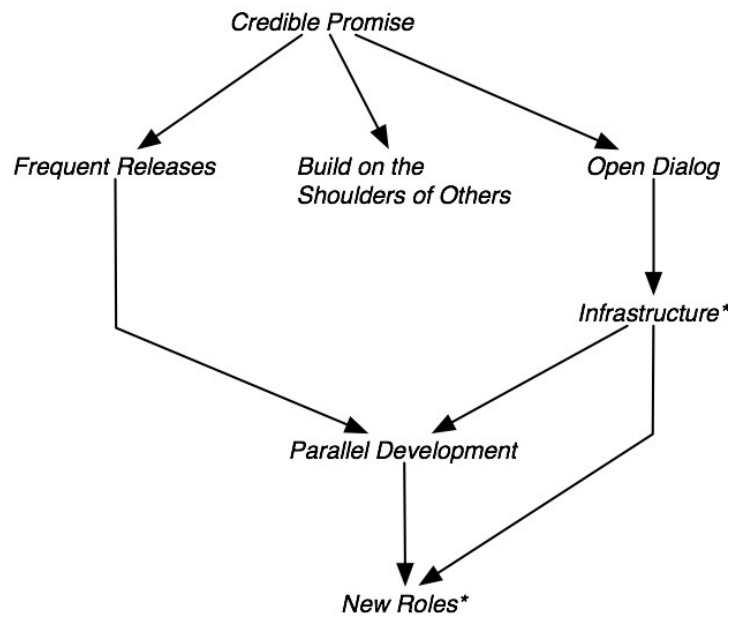


Figure 1: Patterns for the performance of open source development

## 2.1 Credible Promise



*What both projects did have was a handful of enthusiasts and a plausible promise. The promise was partly technical (this code will be wonderful with a little effort) and sociological (if you join our gang, you'll have as much fun as we're having). So what's necessary for a bazaar to develop is that it be credible that the full-blown bazaar will exist! (Raymond, 1998)*

<b>Context</b>	You are starting an open source project. So far you have been working on the project by yourself. You want other developers to join and leverage their contributions to grow the project.
<b>Problem</b>	<b>How do you mobilize developers to contribute to your project?</b>  In economics this problem is known as penguin problem. Hungry penguins are gathered on a floe of ice. However, none of them wants to dive first for fear of being eaten by a predator. No penguin moves until every penguin moves.
<b>Forces</b>	If your project does not provide a core of the intended functionality, developers will not have enough incentives to join your project.  Developing an initial core of functionality takes time.  You don't want to be too far ahead in your implementation, but leave developers with unresolved challenges.  When developers join they need to make an investment in your project, something they lose if your project fails.  Sometimes you already have code from a closed source product that you want to open up. Opening up means that you need to release control. Otherwise, why would external developers work for you for free, if they didn't share ownership with you.  Therefore,
<b>Solution</b>	<b>Build a critical mass of functionality early in your project that demonstrates that the project is doable and has merit.</b>  An exception to this rule is that projects without running code can attract developers when their creators have a high reputation.
<b>Consequences</b>	A critical mass of functionality is valuable for other developers.  Time spent on developing the initial core to attract developers early on in the project is often offset by faster growth later. However, there is also a risk: you may not succeed in growing a critical mass of functionality, and the extra time and effort spent on architecting the project for a community of developers will be “sunk”.



When you focus on your core functionality, other developers will find it challenging to contribute additional features.

One risk is that developers who join your project may benefit more from the project than you gain from their contributions.

Opening up a closed source project can extend its life. However, other developers can now also influence its direction.

**Examples**

Fetchmail and Linux are the two examples Raymond refers to.

BigBlueButton is an open source web conferencing system whose functionality was sufficient for teaching our online courses, but did not yet have the bells and whistles of competing commercial products (eg recording and playback or desktop sharing).

Subversion initially had no running code. Its credibility came from its founders, the main developers of the CVS versioning system.

**Related patterns**

When starting an open source project you want to move fast. Rather than waiting for a polished release with full functionality aim for *Frequent Releases* (5). This allows you to learn from users how they use your code; advanced users will also discover and even fix problems for you, resulting in higher code quality.

To simplify the bootstrap process required to produce a critical mass of functionality *Build On the Shoulders of Others* (7). Reusing components with proven functionality also allows a project to reach a higher level of quality earlier (ie improve quality and time).

Engaging others in your project requires an *Open Dialog* (9).

*Seeding* (Homsy & Raveh, 2007) an online community with content ensures that people will find it worthwhile visiting.

Benefits and risks of developing a product with other companies are described in *In Bed with the Enemy* (Weiss, 2007).

**Sources**

Literature (Lerner & Tirole, 2002; Fogel, 2006; Haefliger et al., 2008) and the author's observation.

## 2.2 Frequent Releases



*Frequent release cycles are both a curse and a blessing. Software developers are creating fixes and patches all the time. The downside is the developer doesn't want to do upgrades all the time. (Klawans, 2007)*

<b>Context</b>	You need to provide a working system early.
<b>Problem</b>	<b>How do you move an open source project along quickly?</b>
<b>Forces</b>	<p>At the start of a project requirements are often unclear. What the user says they want may not be what they really need.</p> <p>Defects can be detected early when releases are frequent. Each release provides an opportunity for feedback.</p> <p>Upgrading to a new release can be disruptive to some users.</p> <p>The larger a change between two versions of a project, the more challenging the new code is to integrate.</p> <p>Releases may be incomplete in terms of functionality or unstable. Users have different risk comfort levels for upgrades.</p> <p>When developers share their changes infrequently, they may duplicate each others' efforts by solving the same problem.</p> <p>It takes time away from other tasks to keep up with changes.</p> <p>Therefore,</p>
<b>Solution</b>	<p><b>Release code in small, quick increments.</b></p> <p>Don't hold off a release until the functionality is complete, but make changes to the code available as soon as the code is complete, ie when it compiles and executes. Each incremental release also brings the system closer to functional completeness.</p>
<b>Consequences</b>	<p>Releasing a system in frequent increments protects you against overdesign. Each release is an opportunity to receive feedback from users that helps you shape the direction of the project.</p> <p>The quality of the project increases when defects are removed early. Higher quality means that you spend less time fixing bugs.</p> <p>User expectations need to be carefully managed. Releases need to be labelled as stable or development releases.</p> <p>Frequent releases allow continuous integration. Each integration step is small, allowing problems to be quickly localized.</p> <p>Developers can reuse intermediate releases, which can significantly shorten development time for individual features.</p> <p>As changes are shared earlier, developers can lever partial solutions</p>

by other developers, making them more efficient.

Integrating frequent releases can be difficult and time consuming. You may want to skip some incremental changes.

**Examples**

Linux, Apache, and Mozilla all have frequent releases.

While BigBlueButton initially did not create releases frequently enough so as to maximize development velocity, automated builds are now created after each successful commit (that passes tests).

**Related patterns**

Working with external developers, who may pursue different goals while they share an interest in your project, requires a mechanism for coordinating between stable and the experimental releases made by different developers, as described in *Parallel Development (11)*.

*Incremental Integration* (Harrison & Coplien, 2006) ensures that subsystems work well together. It encourages developers to check regularly for incompatibilities with other subsystems.

The technical implications of frequent releases (out of scope for us) are described in *Continuous Integration* (Elssamadisy, 2007).

**Sources**

Literature (Lui & Chan, 2008; Fogel, 2006; Harrison & Coplien, 2006; Goldman & Gabriel, 2005), and the author's observation.

## 2.3 Build On the Shoulders of Others



*We used open source extensively in the creation of the Nokia 770. We favored components that were developed by active communities and already used by many. [...] We created the product in shorter time and with fewer resources, compared to other products utilizing proprietary software. In essence, open source offers time and cost savings in a form of readily available components and subsystems, available developers, and effective development model. (Jaaski, 2006)*

<b>Context</b>	You need to build a critical mass of functionality. You need to provide a working system early.
<b>Problem</b>	<b>How do you grow a critical mass of functionality quickly?</b>
<b>Forces</b>	The faster you can deliver the core functionality, the more and the sooner you will be able to attract other developers. Reuse allows to leverage the code and experience of others, but it takes time to select and to understand external code. Not invented here (NIH) prevents us from looking outside. Personal pride can be in the way of reuse. Code that you reuse is not streamlined to your project goals. You only own the intellectual property (IP) on code that you write yourself, or that you paid somebody external to develop. Maintaining your own code is hard enough. Maintaining external code of potentially variable quality can be much harder. Not all pieces of your system are equally valuable. Building all pieces yourself means that you will need to spend time on building functionality that you would rather spend elsewhere. Therefore,
<b>Solution</b>	<b>Integrate assets from other open source projects.</b> Your development time is reduced by leveraging the functionality provided by existing code and building on existing designs. Your main task becomes one of writing “glue” code that links these code assets. Much of your leverage will come from building on code

developed by others in the form of libraries, components, or tools. However, there are other opportunities for reuse: APIs (a new implementation of an existing API), exchange formats (writing to and reading from existing formats), services (invoking code that is hosted elsewhere), requirements (cloning another design), and test suites (compliance with an existing specification).

<b>Consequences</b>	<p>You shorten the time to deliver a critical mass of functionality, which provides an incentive for others to join your project.</p> <p>By reusing proven code, you can also produce code of higher quality, sooner. However, this introduces a new problem: your project now becomes dependent on other projects you don't control. Whenever one of the projects you depend on changes, you need to incorporate them back into your project, otherwise you will not be able to benefit from bug fixes made to those projects.</p> <p>Your ability to reuse depends on how well you have adopted a culture of reuse. Reusing existing open source components is not necessarily “second nature” to your developers. It may help to make reuse part of refactoring by putting a process in place that ensures that quick initial solutions are replaced by existing components for longer-term stability of the project.</p> <p>You inherit “baggage” from existing assets that you don't need, and which may lead to duplication and to maintenance problems.</p> <p>Also, as you reuse code created by others, you need to ensure that you comply with the licenses associated with this code.</p> <p>You need to maintain a stack of software artefacts developed by others. But this also creates opportunities for third parties who take on the task of managing the stack (companies like RedHat).</p> <p>Reuse allows you to focus on the parts you like to work on, or which add most value by incorporating existing components for the parts that are necessary, but less interesting or valuable to you.</p>
<b>Examples</b>	<p>The initial version of BigBlueButton was built in a few months by combining many existing pieces of functionality, including Red5, an open source version of the Flash Media Server; Asterisk, an open source PBX; OpenOffice to convert PowerPoint slides to PDF; and Apache ActiveMQ, an open source message broker. The actual code of the initial version was only around 10K lines of code.</p> <p>Many projects (both small and large) build on the Eclipse platform by developing their application as an Eclipse plugin. There are more than a 1000 plugins listed on the Eclipse Plugin Central site. The code common to the plugins is provided by Eclipse.</p> <p>OpenOffice is both an example of requirements cloning and reuse of exchange formats. OpenOffice implements most of the feature set of Microsoft Office, and can read and write Word documents.</p>
<b>Related patterns</b>	<p>Leveraging existing code to speed up development is suggested by <i>Prototype a First-Pass Design</i> (Foote &amp; Opdyke, 1994).</p> <p>There are different ways of achieving license compliance. Patterns have been documented for licensing, but not for compliance.</p>
<b>Sources</b>	<p>Literature on reuse in open source (Haefliger et al., 2008), software cloning (Lui &amp; Chan, 2008), and the author's observation.</p>

## 2.4 Open Dialog



*The open-source approach is new to Autodesk, especially publicly discussing new features. It has taken considerable effort to get used to this, but we have taken it to heart, and all new features for MapGuide Open Source are debated in the public mailing list before we start development.” (B. Dechant, MapGuide technical architect), quoted in Birch (2007)*

<b>Context</b>	You want users and other developers to contribute to your project.
<b>Problem</b>	<b>How do you engage others in your project?</b>
<b>Forces</b>	<p>You need to listen to your users; the dialog must be two-way.</p> <p>You don't want to appear weak by sharing your ongoing decision process, including the wrong turns and mistakes.</p> <p>It is difficult to guess what users really want. If you ask them, they are often unable to articulate their needs.</p> <p>Users and external developers need to feel valued.</p> <p>You can't build everything yourself.</p> <p>Other users and developers must benefit from participating.</p> <p>Therefore,</p>
<b>Solution</b>	<p><b>Conduct the project in the open, maintaining a two-way dialog with project participants (users and external developers).</b></p> <p>Give users and external developers access to your source code. Artefacts to share include the code, installation instructions and binaries (as applicable), and test plans. Allow outsiders to participate in your project decisions by discussing project plans on a wiki or a project mailing list. Create opportunities for others to participate by allowing them to fix bugs and add features.</p>
<b>Consequences</b>	<p>Users will provide you with valuable feedback. This allows you to learn from your users and improve the “fit” with their needs.<sup>3</sup></p> <p>Making your decision process transparent builds trust between you and other project participants.</p> <p>When you share your decisions with your users, they can tell you that what you propose to do is not what they want.</p> <p>Openness encourages reciprocal behavior. Sharing with others will</p>

<sup>3</sup> Quality depends as much on building the right system (meeting explicitly stated requirements) and building the system right (meeting the unstated requirements as well, ie fit).

cause them to contribute back to your project.

You will be able to achieve more by enlisting outside contributions.

The result of open participation is greater value than what any individual contributor could have achieved on their own.

### **Examples**

The contributors to the BigBlueButton project are developers at the company co-founded by the original developer, students, faculty of the university (who are also lead users of the system), and members of other businesses who are developing complementary products. Code, project plans, as well as bugs and feature requests are shared by hosting the project on Google Code. Currently, any registered contributor can contribute equally to the project.

MapGuide open source project by Autodesk quoted above.

Hosting sites such as SourceForge or Google Code provide a set of tools for publicly sharing the output of an open source project.

### **Related patterns**

Maintaining an open dialog requires a supporting *Infrastructure* (13) of tools such as wikis, mailing lists, bug trackers, or repositories.

As a project grows, tasks such as moderating a mailing list require dedicated resources. The answer is to create *New Roles* (13).

Canned hosting sites like Google Code provide you with a technical *Infrastructure* to maintain an open dialog. These sites typically include code repositories, wikis, mailing lists, and bug tracking tools. They also maintain developer profiles and project statistics.

*Engage Customer* (Harrison & Coplien, 2006) ensures that there is a continual exchanges between developers and customers.

*Gatekeeper* and *Firewall* (Harrison & Coplien, 2006) deal with the issues of translating and filtering external interactions.

### **Sources**

Literature on open participation in open source projects (Goldman & Gabriel, 2005) and the author's observation.

## 2.4 Parallel Development



*The current production versions are Python 2.6.2 and Python 3.0.1. You should start here if you want to learn Python or if you want the most stable versions. Note that both Python 2.6 and 3.0 are considered stable production releases, but if you don't know which version to use, start with Python 2.6 since more existing third party software is compatible with Python 2 than Python 3 right now. (Python Software Foundation, 2009)*

<b>Context</b>	You need to manage the expectations of your users.
<b>Problem</b>	<b>How do you balance the need of users for stability with the need to explore new directions for your project?</b>
<b>Forces</b>	<p>While developing the current release of your system, you also need a way of working on new features for future releases.</p> <p>External developers may pursue different goals from you when they participate. You need to give them a mechanism for pursuing their interests, while benefiting from their contributions.</p> <p>Therefore,</p>
<b>Solution</b>	<p><b>Maintain separate release streams, those with the official stable releases, and others for experimental development.</b></p> <p>There can be multiple levels of stability (such as nightly builds, weekly releases, and scheduled milestones).</p>
<b>Consequences</b>	<p>This solution addresses the needs of your internal development (feature roadmap), as well as those of external developers. The stable versions of your project can be used for ship products, and experimental versions allow you to explore future products. As you overlap maintenance and new feature development, you also shorten the time to introducing those new features.</p> <p>However, developers need to allocate time to coordinate between the different versions, which they cannot use to write new code. For example, when a bug is fixed in the stable version, it needs to be applied to all the experimental versions as well.</p>
<b>Examples</b>	<p>The BigBlueButton project has separate streams for the production version, which is used to teach online classes and therefore requires the behavior of the system to be stable, and streams for new feature development that will eventually be rolled into the stable stream.</p> <p>The Eclipse project has a yearly release train. Projects to be contained in the release train need to meet a set of well-defined</p>



requirements (such as signed-off milestones).

The Python project currently has two stable releases (see quote).

**Related  
patterns**

In a large project managing the different releases requires dedicated release managers, and example of *New Roles*.

*Named Stable Releases* (Harrison & Coplien, 2006) provide a handle for communicating changes to developers.

There are many patterns such as Berczuk (2003) about the technical aspects of configuration management (out of scope for us).

**Sources**

Literature on parallel development in open source (Fogel, 2006; Muffatto, 2006; Davies, 2009), and the author's observation.

### 3 Conclusion

In this paper we presented a first set of patterns on open source development that will form the core of a larger pattern language. In this set we focused on open source practices that improve the performance of a project, that is, how adopting these practices helps reduce the time, improve the quality, and reduce the cost of an open source project.

There are several parallels between open source and agile development practices as noted by Lui & Chan (2008) and Goldman & Gabriel (2005). This paper does not claim that practices like *Frequent Releases (5)* and *Parallel Development (11)* are unique to open source development, but it emphasizes their key position in the open source paradigm. Other practices related to making development transparent and creating a credible promise are more germane to open source development. However, every practice described has aspects unique to the context of open source development.

Future papers will document patterns for other perspectives on open source development. Overall, the author envisions the collection of patterns to contain patterns on the strategic use of open source (why adopt an open source approach), open source product development (of which these patterns are a part), technical architecture (how are open source systems structured to encourage contributions), licensing aspects (impact of license choices), and governance of open source projects (organization).

### Acknowledgements

I thank Cecilia Haskins for shepherding this paper. I especially thank her for her deep insights into patterns and her patience with a slow author.

In formatting these patterns I owe a tremendous amount to the format Allan Kelly has used in his papers, which I tried to emulate.

### Appendix A - Planned patterns

Here are short forms of the patterns not described in this paper.

Infrastructure	How do you share project decisions effectively? Leverage coordination tools that can be accessed by all project participants. These include mailing lists, messaging, code repositories, bug tracking tools, and wikis.
New Roles	When a project gets too large, how do you coordinate contributions? Define formal roles for contributors (such as issue manager) and the interaction between them.

### Appendix B - Contributions of the patterns

This table summarizes how the patterns described in this paper affect the different dimensions of performance (time, cost, and quality). These relationships could provide the basis for testable hypotheses for a future study, which may empirically establish the impact of the patterns.

	Minimize Time	Minimize Cost	Maximize Quality
Credible Promise	+		
Frequent Releases	-	(-)	+
Build on the Shoulders of Others	-	(+)	+
Open Dialog		(-)	+
Parallel Development	-	(-)	+

## References

I tried to limit the number of references, but the ones below are needed to give proper attribution. Key references are highlighted with a (\*).

Baldwin, C., and Clark, K. (2006), Architecture of participation: does code architecture mitigate free riding in the open source development model?, *Management Science*, 52(7), 1116-1127.

Berczuk, S. (2003), *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, Addison Wesley.

BigBlueButton (2009), <http://code.google.com/p/bigbluebutton>.

Birch, J. (2007), *MapGuide Open Source: Project Insights and Practical Applications*, August, Geoplace.com

Davies, T. (2007), On branching and frequent releases, <http://twmdesign.co.uk/theblog/?p=37>

Elssamadisy, A. (2007), *Patterns of Agile Practice Adoption*, InfoQ.

\* Fogel, K. (2006), *Producing Open Source Software: How to Run a Successful Free Software Project*, O'Reilly.

Foote, B., & Opdyke, W. (1994), Lifecycle and refactoring patterns that support evolution and reuse, *PloP*, and in Coplien, J., & Schmidt, D. (1995), *Pattern Languages of Program Design 1*, Addison Wesley, <http://www.laputan.org/lifecycle/lifecycle.html>.

\* Goldman, R., & Gabriel, R. (2005), *Innovation Happens Elsewhere: Open Source as Business Strategy*, Morgan Kaufmann.

\* Haefliger, S., von Krogh, G., & Spaeth, S. (2008), Code reuse in open source software, *Management Science*, 54(1): 180-193.

\* Harrison, N., & Coplien, J. (2006), *Organizational Patterns of Agile Software Development*, Addison Wesley.

Jaaski, A. (2006), Building consumer products with open source, *Linux Devices*, Dec, <http://www.linuxdevices.com/articles/AT7621761066.html>.

Johnson, R. (2007), Is it a tomcat, or the elephant in the room, Spring Source blog, posted on Dec 24, 2007, <http://blog.springsource.com/2007/12/24/is-it-a-tomcat-or-the-elephant-in-the-room>.

Lerner, J., & Tirole, J. (2002), Some simple economics of open source, *Journal of Industrial Economics*, 50(2): 197-234.

\* Lui, K.M., & Chan, K., Software Development Rhythms: Harnessing Agile Practices for Synergy, Wiley.

Milinkovich, M. (2008), A practitioner's guide to ecosystem development, Open Source Business Review, October, [www.osbr.ca](http://www.osbr.ca).

\* Muffatto, M. (2006), Open Source: A Multidisciplinary Approach, Imperial College Press.

Raymond, E. (1998), interviewed by F. Cavalier from Mib Software.

Homsky, O., & Raveh, A. (2007), Pattern language for online communities, EuroPLoP.

Weiss, M. (2007), In bed with the enemy, EuroPLoP.

## Photo credits

Penguins on an ice flow, by T. Ellis and shared under a CC-BY-NC license, [http://www.flickr.com/photos/tim\\_ellis/26360944](http://www.flickr.com/photos/tim_ellis/26360944)

Human pyramid, by somerandomsequence, shared under a CC-BY-SA license, <http://www.flickr.com/photos/somerandomsequence/3898247588>

Synchronize, by sammiji, need to obtain rights or replace, <http://www.flickr.com/photos/sammiji/3417689614>

Open door, by D. Seagers and shared under a CC-BY-NC-SA license, <http://www.flickr.com/photos/seagers/1805045379>

II, by Lori B. and shared under a CC-BY license, <http://www.flickr.com/photos/13025462@N08/2905698081>

# The Role of Analysis Patterns in Systems Analysis

Anna Bobkowska, Jakub Grabowski  
Gdansk University of Technology, Poland  
Contact e-mail: annab@eti.pg.gda.pl

## Abstract

Analysis patterns seem to be a promising approach supporting software analysis. However, the usefulness of this approach has not been proved. This paper presents a 'big picture' of issues related to application of analysis patterns in software development process from practitioner's perspective. It examines several aspects of analysis patterns and their application in software development. Then, it provides a method of system analysis with analysis patterns. And finally, it presents a case study of analysis with analysis patterns for a system supporting a small real medical centre.

## 1. Introduction

Analysis patterns are conceptual structures which represent patterns for object-oriented analysis. The first book about analysis patterns was written by Martin Fowler [F97], who defines pattern as an idea that has been useful in one practical context and will probably be useful in others. For example, no matter what we plan, we consider some actions, resources, parties, time periods and locations. Analysis patterns capture the experience of analysts and provide reusable fragments of object-oriented analysis models. It is believed that application of analysis patterns can improve quality of analysis and reduce time and cost of system analysis. Fowler's book contains patterns for the areas of accountability, observations and measurements, inventory and accounting, planning and trading. They were developed on the basis of author's experience in industrial projects. Another source of analysis patterns are the proceedings of PLoP workshops [P]. With the reference to the more general idea of 'pattern', the meaning of analysis pattern has been enriched by the notion of 'template' - a universal structure which is valid regardless of the domain. Several analysis patterns can be found in the proceedings of the PLoP workshops, e.g. [F00, FY99, FY00, FY01, HF04a, S00].

Analysis patterns are promising, but not a very popular practice in software development nowadays. They are not supported by general purpose UML tools although some of the UML tools contain design patterns. In the area of application of analysis patterns it is worth to mention the following pieces of work: a proposition of template for analysis patterns descriptions made to facilitate their application [H01]; an observation that analysis patterns can be applied by specialization when one has abstract patterns or by analogy when one transports models from another project [FY00b]; an attempt to integrate analysis patterns with MDA approach [FN05] and the idea of 'stable analysis patterns' (which were designed to satisfy criteria of traceability and generality) together with a method for applying them in opposition to applying analysis patterns by analogy [HF04b]. These papers suggest ways of applying analysis patterns, but they do not pose questions related to effectiveness of this application. Thus, more systematic research related to application of analysis patterns is needed.

*Copyright retain by authors.*

*Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.*

The goal of this paper is to investigate the application of analysis patterns. But we do not attempt to find a possible application of analysis patterns. We take the perspective of analysts who is interested in the following: How analysis patterns can really support the process of analysis and writing documentation of analysis phase? What are their strengths and their weaknesses? What is their role in analysis? Which systematic process of system analysis with analysis patterns could be applied?

These questions are motivated by the fact that software projects usually have no resources for waste on application of 'yet another' technique. All activities must have their justification in increase of quality or efficiency. Analysis is a very important phase in software development process and it can benefit from analysis patterns.

This paper provides a framework for dealing with very detailed aspects of analysis patterns application. It constitutes a 'big picture' of all identified important issues which should be taken into account as the context in more detailed research.

The target audience are researchers and practitioners who have got already an idea of analysis patterns, but they would like to explore them in more details including the role of analysis patterns in analysis, their strengths and weaknesses, the process of their application and the effectiveness and efficiency aspects of their use. This paper may be interested for professionals with a skeptical attitude towards patterns and analysis patterns in particular. They might find out conditions which must be met in order to succeed with patterns.

The paper is structured as follows. Second section examines analysis patterns for their features and possibilities of application in system analysis. Section three takes a practitioner's perspective and presents a method of systematic application of analysis patterns in system analysis. Section four presents a case study of applying analysis patterns in analysis of a small medical centre. Section five presents conclusions.

The structure does not entirely represent the order of performing research. In the course of work many iterations took place and results of some pieces of work described later in this paper were the input to aspects described earlier. The case study was done (by Jakub Grabowski) in the parallel to theoretical investigations (by Anna Bobkowska). The study described in section 2.7 was made earlier and, in fact, it has inspired posing some of the questions. The proposition of the process described in section 3 is the answer to the problems identified in section 2. The authors hope that use of this process allows to eliminate some problems with application of analysis patterns.

## **2. Analysis Patterns under Examination**

While searching for the role of analysis patterns in software development, we discuss the following aspects:

- Analysis patterns from the perspective of analysis phase;
- Different aspects of the notion of analysis patterns;
- Generality of analysis patterns with respect to domain;
- Fit between the context of discovery and the context of application;
- Quality of analysis patterns;
- Efficiency of the application of analysis patterns;
- Perception of analysis patterns by MSc students.

## **2.1. Analysis Patterns from the Perspective of Analysis Phase**

Analysis is a creative part of software development process. It is concerned with creating the vision of software product, exploring details of business processes, learning about domain and checking whether the proposed solution is really what is needed in the customer's enterprise. The fundamental assumption of analysis is that different companies have different needs even if they operate in the same domain. Some analytical activities can be made according to some guidelines, e.g. describing existing business processes, but others require visionary skills, e.g. creating innovative systems or re-organizing the business process. Analysts must also combine different points of view and discover tacit knowledge. Thus the process of analysis cannot be replaced by automatic insertions of the analysis patterns. However, it can be supported by providing some knowledge about domain, vocabulary and solutions. Analysis patterns are one of many sources of knowledge about systems solutions.

Analysis patterns cover only a few aspects which are important for analysis in software development. With the increasing size of software systems and the increasing role of economical aspects and business modeling when developing systems, modern approaches to system analysis (IIBA [iiba], RUP [RUP]) focus more and more attention on relationship between business and system, as well as the relationship between system deployment and economical factors in software projects.

A proven area of software technology related to systems analysis is requirements engineering. In some cases, requirements engineering techniques are more useful than system analysis, e.g. testing similar systems, ethnography, surveys, communication with customers and future users of the system. Potentially analysis patterns can facilitate communication of analysts and customers by delivering to analyst the knowledge about the domain and vocabulary of customers. However, when customers have different mental representations than these described by analysis patterns, they can even make the analysis more difficult.

One of the reasons of a small popularity of analysis patterns might be the fact that not always in software development a systematic object-oriented analysis is made. Analysis patterns are strictly related to object-oriented analysis. Furthermore, they are a more advanced technique and they are even more seldom in use. In practice, analysts focus mainly on the scope of the system and their functional requirements, fit in context and fit to business. These aspects are seldom covered by analysis patterns. Different approaches to object-oriented analysis can be found in literature, e.g. there are differences between OMT [OMT], Fowler [F97] and Rational Unified Process [RUP]. Some approaches focus more on business while others on key classes of the system. There is no standard of object-oriented analysis since UML [UML] defines only the language and not the application. Additionally, there are some problems with analysis patterns themselves. Fowler doesn't use UML diagrams and structured descriptions. While asking the representatives of software companies or even universities whether they know and use analysis patterns the answer is usually 'no'. The issue of whether analysis patterns are not used because of the lack of knowledge about them, or because of uselessness of the idea of analysis patterns, or maybe because of low maturity of analysis patterns nowadays, requires more investigation.

## **2.2. Notion of Analysis Pattern**

The notion of analysis pattern combines the notion of analyst experience and the notion of a kind of universal template of solution which can be reused in several projects. However, experience from one project does not necessarily mean the template for all other projects.

According to the rules of induction, if one wants to have a general solution one should work out independently several specific cases and make their generalization. Analysis patterns presented by Fowler were developed on the basis of a few industrial projects and most of analysis patterns presented at PLoP workshops were developed in academic environment and their usefulness in real industrial projects is uncertain.

There is a diversity of analysis patterns. Factors, other than their origin, include:

- Different scope of problem - analysis patterns range from a single class, e.g. quantity or time stamp patterns, to fragments of solutions which contain about 20 classes;
- Level of maturity - some patterns were very precisely elaborated while others represent just a 'first approach' to analysis with a list of aspects they do not cover;
- Level of abstraction - some patterns were generalized on the basis of several solutions and present an abstract pattern while others are just description of someone's experience and de facto are fragments of concrete projects.

The diversity of patterns has impact on differences of their potential application.

Additionally, patterns might provide terminology at a higher level of abstraction and resulting benefits for communication similar to the application of design patterns. With design patterns the developers can use just one word 'bridge' or 'mediator' for explaining the entire fragment of solution. However, in order to fulfill that goal analysis patterns should be general and they should have unique and representative names.

### **2.3. Generality of Domain and Progress in Systems Development**

Generality of domain influences the commonality of potential applications of analysis patterns. Some patterns represent domains which are popular and others deal with very specific domains. For example, planning is popular in almost every field of activity while the application of trading is limited to banks and advanced exchange of goods.

Some domains can be characterized by a closer similarity among the cases whilst others are different from case to case, e.g. accounting in a given country is standardized and all accountants must do it similar way while Customer Relation Management (CRM) can be based on a diversity of possible strategies and it can be done in several ways.

Some domains are independent of the context whilst others are dependent on it, e.g. local legal regulations, local customs, local values, etc.

Software systems evolve in time. They become larger and they cover wider and wider scope of activities. The progress in technology and the state-of-the-art of developing systems means that patterns made decades ago might not be useful nowadays. They need to be updated.

### **2.4. Fit between Context of Discovery and Context of Application**

For better understanding of the application of analysis patterns, it is important to distinguish a context of discovery and a context of application. The context of discovery represents the circumstances in which analysis patterns were discovered. The areas include: country, company, projects, methodology, mentality of the authors, time, etc. The context of application represents the circumstances in which analysis patterns are applied. It might happen that an analysis pattern is to be applied in a country in which different regulations or customs obey, or in a project with different focus, needs or methodology. The bigger gap between these two contexts, the lower probability that patterns will fit to the problem at hand.



The analysis patterns need customizations during their application. Let's consider an example of a small shop and a supermarket. Although both of them sell goods, some analysis patterns made in the context of supermarket might be useless in the small shop because of different problems and processes. Another example can be planning. It seems to be a very general analysis pattern. However, plans in different domains cover specific artifacts, e.g. planning software project (additionally to what planning pattern contains) requires phases, iterations and distinction to project-level and iteration-level plans.

## **2.5. Quality of Analysis Patterns**

It is believed that analysis patterns will allow to increase quality of the analysis. The argument is they are verified in successful applications. However, in order to satisfy the goal of increasing the quality of analysis, the quality of analysis patterns must be high. Analysis patterns support elaborating more detailed solutions in the scope covered by the pattern. However there are also the following risks:

- solution contains redundant model while a simpler model is needed;
- the fragments of model which are not covered by the pattern are skipped although they are needed;
- analysts-beginners cannot evaluate quality of the applied patterns and they believe they have good solutions while actually they have poor models.

An interesting and fundamental issue is the meaning of 'quality of the analysis patterns'. Assuming that the goal we want to achieve is ease of searching for useful patterns, ease of their application in terms of customizations and traceability as well as increase in quality of solution, a provisional list of the quality characteristics includes:

- precision of the content - analysis patterns which contain more details are considered as more useful;
- generality regarding domain - more general analysis patterns can find application in more projects;
- level of abstraction - more abstract patterns will fit to more problems, but on the other hand they will require more customizations;
- structure of description - structured description facilitates applications;
- aspects of description - some descriptions of analysis patterns do not cover functionality which is an essential aspect of software analysis; the patterns which cover all essential aspects will be more useful.

Apart from these aspects of quality which are application-specific, analysis patterns should satisfy the criteria of object-oriented analysis models, such as completeness in a given scope, correctness, consistency as well as clear element names on the diagram, clear descriptions, clear layout of diagram etc.

## **2.6. Efficiency of Analysis Patterns Application**

Statements about analysis patterns efficiency claim it is possible to achieve reduction of time and cost of software development with analysis patterns. They usually use the idea of inserting some fragments of solutions instead of making system analysis from scratch.

The idea of savings assumes the most optimistic case: high quality patterns, easy access to patterns, fit of the patterns to problem at hand and easy integration of patterns with other models. In reality, the following risks and their consequences might appear:

- when customizations of patterns are necessary - the savings of time might not be so large;
- when the context of application is very different from the context of discovery - the analysis patterns might not fit at all or their application is not efficient;
- when wrong patterns have been chosen for application - the analysis patterns might not fit at all or their application is not efficient;
- when there are no patterns to support a given part of the solution - the time is wasted for searching for analysis patterns and traditional analysis must be made anyway;
- when the quality of patterns is poor - a poor solution is achieved and analysis must be redone;
- a hidden use of resources is the work on integration of analysis patterns with other models.

In order to avoid the problems described above, one should evaluate usefulness of the patterns before applying them.

When searching for a reference model to estimate savings one encounters a large set of variables. The results of object-oriented analysis depend on analyst's personality, skills and experience, domain and novelty of project, methodology in use and its fit to the analyst style. Most of them can't be controlled. The only variable which can be universal is the methodology (micro-process). A different methodology should be used when making analysis without analysis patterns, and a different one is required to make analysis with analysis patterns.

## **2.7. Perception of Analysis Patterns by MSc Students**

A voice in discussion about analysis patterns can be the perception of analysis patterns by MSc students in specialization of Software Engineering. We make an exercise related to application of analysis patterns which covers the following tasks:

- describe a group of analysis patterns including Fowler's patterns as well as selected PLoP patterns,
- find for them as many applications as possible,
- try to apply them in some cases by customizing the patterns for this application,
- provide comments on their application.

The results of the exercise are presented and discussed on the forum of the group.

About sixty students took part in this exercise. The following results were achieved. The students perform the descriptions very well. The applications they find depend more on imagination of students than on the group of patterns. The analysis with the customization of models seldom is done well. The comments depend on which group of patterns they were working with and what were their achievements.

We have collected the following comments and opinions. The students appreciate industrial origin of the Fowler's patterns. They consider them as advanced, complicated, poorly described (unstructured description of text with diagrams in other notation than UML), sometimes too specific for a universal pattern. Fowler's patterns are considered as potentially useful description of how someone have done the analysis. The patterns from PLoP proceedings are more controversial. Some students consider them useful in practice or useful for learning as academic examples (according to their context of discovery), but others have totally negative attitude. They argue they are useless because they present obvious and simplified models and that they sometimes contain defects. These patterns are considered as easy to understand, well-structured, immature, good for a start but not sufficient for solving real industrial problems. We suspect that the reason of the critics is the difference between

expectations and the content of patterns or preferences of models made in different modeling style, e.g. regarding the use of association class related to association class. Students with industrial experience claim that the analysis patterns sometimes do not cover all essential aspects. For example, in case of stock pattern - it presents only a basic structure for goods in the stock while it does not consider documents, nor calculation of the value of goods in the store, nor different kinds of sub-stores; or in the area of negotiations - there is no representation for offers nor indicators of negotiations which must exist in Negotiation Support Systems.

To conclude, analysis patterns appear as a valuable description of analyst experience with the potential to be used in other projects, but in current state of the art they shouldn't be treated as general, universal templates. Their application cannot replace analysis. Students usually have had an industrial experience and the level of knowledge about analysis they will have when leaving university. Thus, the results generalize well to software developers who have graduated from university but do not have experience with application of analysis patterns. But their opinions about analysis patterns mean that if they do not consider analysis patterns as a mature technology they will not use it in practice.

### **3. System Analysis with Analysis Patterns**

In this section, we are looking for a realistic methodology how to make the best use of analysis patterns in system analysis. After the examination of the analysis patterns, the fundamental assumption of the approach is that they might be helpful in system analysis but the choice and integration of analysis patterns will not replace the process of analyzing software requirements.

#### **3.1. Macro-process for Reuse of Analysis Patterns**

Analysis patterns can be considered as reusable fragments of solutions and general rules of reuse paradigm can be applied. Reuse paradigm defines two processes:

- development 'for reuse', which produces reusable assets and stores them in a repository;
- development 'with reuse', which develops applications with the use of reusable assets; it contains also other activities e.g. acquiring or verification of requirements.

By analogy, system analysis with analysis patterns should also have these two processes. They are presented in Fig. 1. The process 'for reuse' contains creation and collection of analysis patterns. It can be realized in a company by creating specific patterns or by acquiring (and customizing) the patterns which have been published. The outcome of this process is a repository with structured descriptions of analysis patterns. The process 'with reuse' contains software development which includes system analysis with analysis patterns.

One of the tasks made during the application of analysis patterns is searching for relevant patterns. The repository of analysis patterns plays the key role in this activity. In order to assess efficient application of analysis patterns the repository should contain appropriate number of right patterns and the patterns should be described with a template which facilitates searching. By analogy to other reuse techniques one can expect much better improvement in quality with the use of company-made analysis patterns than with general ones.

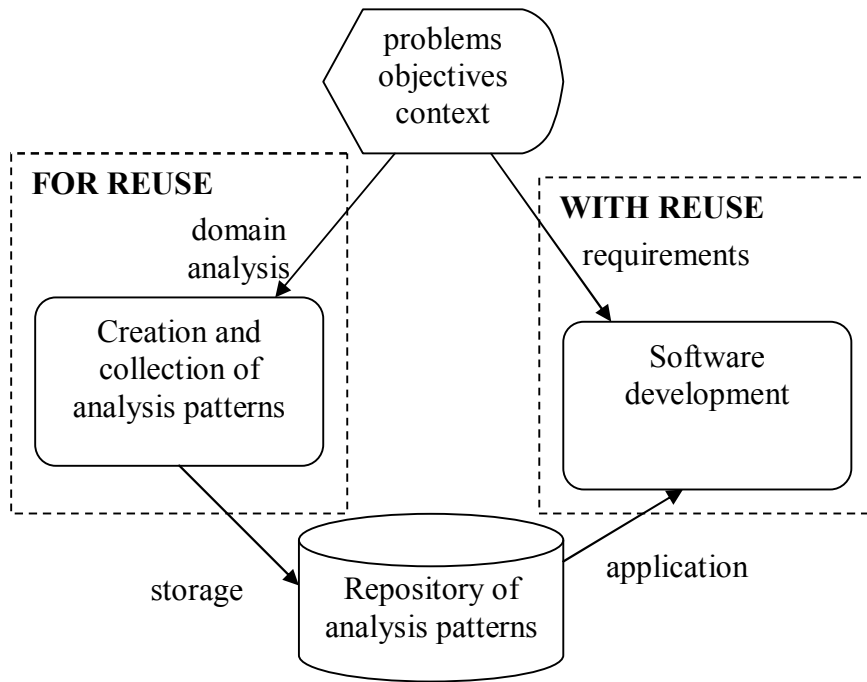


Fig. 1. Analysis patterns in the perspective of development 'for' and 'with' reuse.

### 3.2 Micro-process of Analysis with Analysis Patterns

The process of system analysis with analysis patterns which is a part of the process "with reuse" is presented in Fig. 2 and activities are described in Table 1. The main role which performs all the activities is analyst. Sometimes the analyst must communicate with the customer, especially when preparing the vision and validating the model. The product of this process is documentation of the analysis which contains both textual specifications and models of the system.

Some people prefer using analysis patterns after they make models. In this case, analysis patterns are useful for extensions, improvements and verification of the model. When referring to the process presented in Fig. 2., draft models made during the task 'identify areas and make draft models' are more detailed and 'customization to the problem' is specialized to extensions and improvements with analysis patterns.

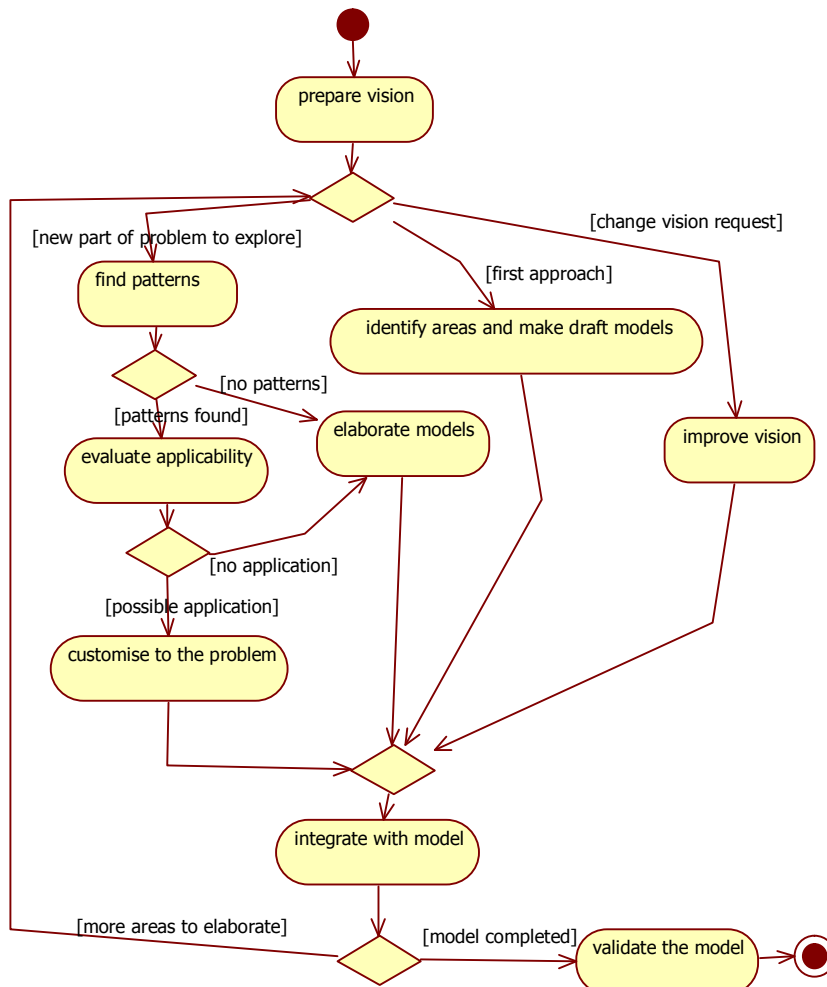


Fig. 2. Activity diagram with a sketch of system analysis with analysis patterns

Table 1. Description of activities in the process of system analysis with analysis patterns.

ACTIVITY	DESCRIPTION
prepare vision	the activity of making initial analysis in order to determine scope of the system in context as well as objectives, priorities, users and the scope of functionality
identify areas and make draft models	the activity of defining key areas, problems and classes before acquiring the patterns; with this activity it should be possible to avoid the risk of skipping the areas uncovered by the patterns, it might contain use case modeling as well
improve vision	the activity performed when analysis leads to the discovery of a better solution than one stated in vision or the analysis causes inconsistency with vision, in this case the vision should be modified
find patterns	the activity of searching for the patterns which could be useful in the analysis of the area under exploration
evaluate applicability	the activity of evaluating the patterns which have been found for their quality, fit to project and possibility of application
customize to the problem	a group of activities which make the patterns fit to the problem at hand; it can include specialization, analogy, simplification, enhancements, replacements of some element with others etc.
elaborate model	the activity performed when no patterns can be applied in a given area, in this case traditional methods of analysis should be used
integrate with model	the activity of integrating the elaborated part of problem with the fragments which have been developed before
validate the model	the activity of checking whether the model expresses what has been expected, e.g. with the participation of a customer

### **3.3 Specific Context of Use of Analysis Patterns**

Analysis patterns are one of the sources of knowledge about the system under analysis. They will not replace the activities of meeting customers and other stakeholders, reading their documents, business process modeling, observations of their work and making decisions about the objectives of the system under development and its business value. Thus, a question for another piece of research is how analysis patterns can be integrated with other methods of system analysis and requirements engineering.

So far, we have presented a proposition of system analysis with analysis patterns. But in several contexts they can gain a new application. In the context of Model Driven Architecture (MDA), the application of patterns is one of the types of transformations. Although MDA deals mainly with transformations of Platform Independent Models (PIM) to Platform Specific Models (PSM) which correspond to design level, analysis patterns could be used in refinements of Computation Independent Models (CIM). A more important role can be played by analysis patterns in Model Driven Software Development (MDSD) approach. It assumes that the models play the central role in software development and in more mature forms that all organizational knowledge is represented in models. In a company where several similar systems are developed, some analysis patterns can be applied in their development. In this case, many company 'home-made' analysis patterns can be created and applied both in UML or domain specific languages (DSL).

## **4. Case Study**

The goal of this case study was to demonstrate the application of analysis patterns. We were interested in the following aspects:

- benefits and problems related to the use of analysis patterns,
- the impact of analysis patterns on the process of analysis,
- efficiency-related aspects.

The analysis with analysis patterns has been performed for a small real medical centre. It was a non-commercial project made as a part of Master Thesis. The choice of the organization was motivated by the fact that there are many medical analysis patterns. Additionally, we were in contact with a doctor employed in this medical centre which allowed for realistic business model of this organization as well as for acquisition and verification of actual requirements for the system.

### **4.1. Description of the Medical Centre**

The medical centre employs about six doctors, a few nurses and a receptionists. It is located in a one-storey building which contains five consulting rooms, a registration office, a waiting room and a few small storage rooms. Doctors have their weekly schedule which contains information on their working hours and planned holidays. Patients can register at the registration office for an appointment on a given day. Medical records are kept in a large paper archive placed in the registration office. Each day a receptionist checks out (looking into doctors' registers) which patients have appointments with a given doctor and carries their medical records to doctor's consulting room before the beginning of work. At the end of the working day, a receptionist takes all the medical records back from doctors' rooms to the archive in the registration office.

## 4.2. Scope of Analysis

The analysis phase lasted for over a month and involved over a dozen of meetings with the doctor and the receptionist employed in the medical centre. The full version of this case study takes over a hundred of pages. The main products of the analysis are:

- Vision document - which describes the details of organization and focuses on the key areas of the analyzed system. These are: electronic medical records, patient registration for the appointment with doctor, stock and resources management, orders and shipments, and contracts for rendering medical services by external health care organizations;
- Use cases - three diagrams with five actors and forty-three use cases with appropriate descriptions;
- Static model - a class diagram with forty-nine classes and their descriptions.

## 4.3. Application of Analysis Patterns

Four analysis patterns has been applied in the case study. These are: the analysis pattern for reservation and use of reusable entities [FY99], the patterns for observations and measurements [F97], the analysis patterns for the order and shipment of a product [FY00] and the analysis pattern for inventories [F00]. Analysis patterns application can be summarized by the following metrics. Twenty-five classes of the full class diagram (49 classes) were adopted from analysis patterns. There were nine major modifications made to the original patterns' classes. The modifications included specialization, removal of some original classes because they were redundant and inadequate for the model, changes in class or attribute labels as well as addition of a few new attributes.

As an example, we present customization of the analysis pattern for reservation and use of reusable entities. The original class diagram of this pattern contains the following classes:

- Client - with attributes of name, address, phone\_No;
- Entity\_Type - with attribute Type;
- Entity - with attributes entity\_id and entity\_layout;
- Availability - with attributes capacity and number\_avail;
- Reservation - an association class between Client and Entity\_Type with attributes start\_date/time, end\_date/time and confirmation\_No;
- UseRecord - an assocoation class between Client and Entity with attributes contract\_No, act\_start\_date/time, plan\_end\_date/time, act\_end\_date/time and payment\_type.

Fig. 3 presents a fragment of the customized class diagram. The following modifications has been made during the customization of the pattern:

- The original class “Reservation” was used by analogy and resulted in a class “Registration”. It was also simplified since reservation is made for a given time and date whilst registration requires only a given date in this medical centre.
- “Doctor” corresponds to the “Entity” from the original pattern diagram. This analogy may seem quite odd, but in fact it fits well in this application. Patient literally 'uses' not the doctor but the advantage of doctor's services for a given period of time.
- “Visit” is an equivalent of the original class “UseRecord”, because it confirms the fact that the patient was examined by the doctor on a specific day and that the examination lasted for a certain period of time. Visit class contains the results of the medical examination (as a part of that patient's medical record). Every visit should have a corresponding registration.

- The “Patient” refines the “Client” class. Patient reserves a particular entity, what means that he registers for an appointment with a given doctor for a given day and this fact finds its confirmation in a “Registration” written to the doctor's register of visits. Every Patient is a beneficent of doctor's services in the same way as client uses a certain entity.
- “Doctor's register of visits” represents the general “Availability” class from the original pattern diagram. It contains all registrations to given doctor and can be used to determine his availability on a certain day.
- Class “Entity\_Type” was considered unnecessary ('doctor's specialty' was modeled with 'type') and it was removed from the model.

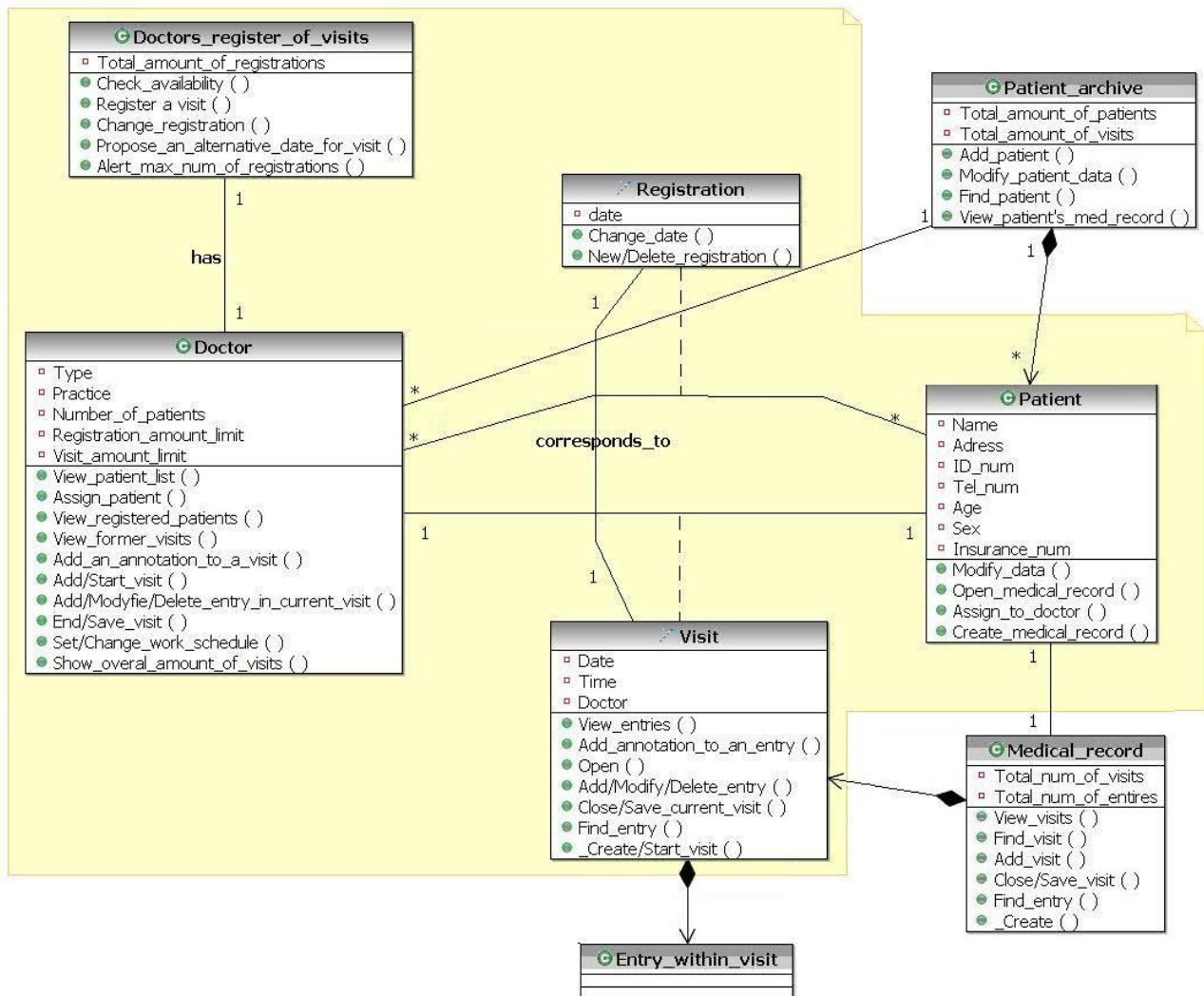


Fig. 3. Fragment of customized class diagram

#### 4.4. Reflections on the Application

This case study was made independently from the work on the method described in section 3. Thus, this method was not used but, in fact, all its activities took place. First, a rich and exhaustive vision of the system was prepared. It precisely describes the scope of the system in the context. The objectives of the system were formulated and priorities for them were assigned. The vision also included the initial requirements for the system and a brief description of its users and their characteristics. Then, use case model was developed. The task of 'identifying areas and making draft models' was limited to defining the areas for



elaboration. Some draft models were created at the stage of evaluating analysis patterns applicability. Finding patterns activity was performed during the earlier part of work on the diploma. The search for useful patterns for the area of exploration was not a necessary task since the author prepared earlier a catalog of analysis patterns and he had a good knowledge of patterns. A brief evaluation of the acquired pattern applicability was performed. The main task at this stage was to determine whether they fit to project and whether their application was possible and reasonable. The context of discovery was compared with the situation and needs of the medical centre. This task allowed to estimate the amount of modifications required to fit the patterns to the project. It provided justification for their use in this project. The activity of customizing the patterns took place at the beginning of static model creation, just after the use case diagram was made. Some modifications of the patterns were made to fit them to the projects requirements. They mainly included simplification (removing unnecessary elements) and specialization (changing the names of a few classes and attributes). In general, this activity was performed surprisingly smoothly and it did not bring larger difficulties. The patterns made foundations for the model and the base for further elaboration and enhancements which involved traditional methods of analysis. After the analysis was finished the model was validated. This process brought a positive result which means that the model expressed what was expected. The major difference with respect to the proposed method was the focus on use cases after preparing the vision and before applying analysis patterns.

The acquisition of analysis patterns started early, at the stage of developing use cases and it has influenced to some extent the meetings with the representatives of the medical centre. Analysis patterns were a kind of reference points for the discussions. They directed the questions which were asked while searching for elements of patterns and concepts in the organization's description. The fact that patterns' concepts may sometimes differ from those of the analyzed organization had to be taken into account to avoid possible misunderstandings.

In this application of analysis patterns we have faced the problems related to the differences between the context of discovery and the context of application. Each of the patterns used in the analytical process had to be adapted to the analyzed environment. In order to make this task, the essential similarities and differences between the mentioned context had to be identified. Then, necessary adjustments and modifications in the model were made.

Application of analysis patterns has brought the following benefits. The greatest advantage was a feeling of shortening of the time spend to understand the rules, principles and mechanisms in the domain. This approach simplified the whole process as the patterns brought not only a general skeleton structure but also many ready elements useful for covering some areas of the medical centre. The last important advantage of analysis patterns is that they introduce flexibility into the model. It may affect not only the resulting analysis models, but also the design and implementation phase in the software development project.

One of the main disadvantages of using analysis patterns were some certain difficulties in adapting them to the set of gathered requirements. Definitely, the main reason was their high level of generality and sometimes a lack of good examples of their application. Unfortunately, the feature of generality can not be completely eliminated because its removal can make the patterns stop fulfilling their major function i.e. being templates of solutions for common, recurring problems. That is the reason why the system analysts, who intends to use analysis patterns, should be aware that they will have to spent a considerable amount of time on making all the necessary modifications and extensions. In some cases, the time spent on seeking patterns and then adapting them to match system requirements will equalize or even exceed the time needed to perform the whole analysis without using the patterns at all. This

matter might get one to wonder if it is reasonable to make use of patterns and benefit from the unquestionable flexibility they introduce.

We would like to finish this section with a 'post-mortem' analysis. The model appeared to have some defects. For example, it doesn't contain doctor's first name and surname. When making traditional analysis the doctor's name probably would be the first attribute we would add to the class of doctor. As it was used here by analogy to 'entity' and entities do not usually have names but identifiers, this attribute is missing. Another example of mistake is 'entry\_within\_visit' which does not have attributes although it should have had. This is a new class and the author forgot to elaborate it. Yet another example of mistake is multiplicity of 'visit' which should not be 1 on both sides. These mistakes prove that application of analysis patterns does not necessarily improve the quality of the entire solution. Analysis must be made with or without analysis patterns. After making this case study, we were informed that more analysis patterns for medical applications were published [SF04, SFL05]. We did not ignore them intentionally but we did not find them in the large archives of the PLOP workshops. This observation confirms the important role of repositories of analysis patterns.

This case study generalizes well to the situation in industry when inexperienced analyst performs analysis with general-purpose analysis patterns. With the increasing experience analysts could learn how to identify and avoid the problems. The use of the method from section 3 could facilitate effective application. In case of using repository of company-made analysis patterns the effectiveness and efficiency are expected to increase significantly.

## **5. Conclusions**

In order to clarify the role of analysis patterns in software development we have examined several aspects of analysis patterns and their application. Their strengths result from the idea of reusable fragments of models, from the experience of analysts and from the verification of the solution in real software projects. Their weaknesses are related to problems with generality, abstraction, precision and description of analysis patterns. During the application of analysis patterns, analysts should take into account the difference between the context of discovery and the context of application. Analysis patterns can facilitate system analysis but it is unlikely they eliminate the need of using other techniques related to system analysis and requirements engineering. They can support creating object-oriented class diagrams and help in understanding the domain. They do not support uncovering specifics of the system, business modeling and, in most cases, analysis of system's functionality.

A systematic process of system analysis with analysis patterns has been proposed. It suggests elaboration of the vision of system under development and then using the available and useful patterns in elaboration of the areas of the problem. In order to avoid unnecessary problems each analysis pattern should be evaluated before using in the project and then customized and integrated with the remainder of solution. The entire model should be validated. It is worth to mention that intention of using of analysis patterns should result in changes of the process. In order to eliminate the risk of leaving uncovered the parts which are not elaborated by the patterns, identification of the 'areas for elaboration' at the beginning of the process is suggested. This process was developed with the consideration of realistic industrial assumptions. The acceptance by practitioners requires verification in further studies.

The case study has shown that it is possible to apply analysis patterns in system analysis. It has also indicated for some benefits and problems of this application. The main benefits are: help in discovering details about the area, flexibility of the model, and possibility of using the

knowledge about the domain in communication with customers. The main problems are: lack of coverage of the functional perspective by analysis patterns, redundant models or lacking details caused by the patterns use, time spent on customizations, misunderstandings when patterns correspond to different mental representations than these owned by customers.

From this perspective we can additionally formulate some requirements for analysis patterns. In order to increase their usefulness, they should have structured descriptions which would facilitate searching for needed analysis patterns, they should cover functionality, e.g. use cases, they should have unique names, and they should be verified and improved to be characterized by really good quality. Catalogues of analysis patterns would facilitate the process of searching for needed patterns.

Some of the discoveries can be valid also for other kinds of patterns. For example, the difference between the context of discovery and the context of application seems to be general enough to exist also in other kinds of patterns. The aspects which were used to examine differences between patterns might be useful for examining other kinds of patterns as well. Finally, the method of analysis patterns application with a small change of 'vision of the system' to 'goal one wants to achieve' and a change of 'model' to 'solution' seems to be applicable also for other kinds of patterns. However, these are only suppositions which should be validated by the designers of other kinds of patterns.

### **Acknowledgements**

Authors would like to thank Eduardo Fernandez for his comments on this paper during the process of shepherding at EuroPLOP. We would like to express appreciation to his work on analysis patterns despite of some critical statements about analysis patterns in the paper.

### **References:**

- [F00] Fernandez E.B., Stock Manager: An Analysis Pattern for Inventories, *Proceedings of PloP 2000*
- [FY00] Fernandez, E. B., Yuan, X., Brey, S. Analysis Patterns for the Order and Shipment of a Product, *Proceedings of PLoP 2000*.
- [FY99] Fernandez E.B., Yuan X. An analysis pattern for reservation and Use of Reusable Entities, *Proceedings of PloP 1999*.
- [FY01] Fernandez E.B., Yuan X. An analysis pattern for the repair of an entity, *Proceedings of PloP 2001*
- [FY00b] E.B. Fernandez and X. Yuan, Semantic analysis patterns, Procs. of 19th Int. Conf. on Conceptual Modeling, ER2000, 183-195.
- [FN05] Filkorn R., Navrat P., An approach for integrating analysis patterns and feature diagrams into Model Driven Architecture, LNCS 2281, 2005.
- [F97] Fowler M., Analysis Patterns: Reusable Object Models, Addison-Wesley, 1997.
- [HF04a] Hamza H.S., Fayad M.E., The Negotiation Analysis Pattern, *Proceedings of PloP 2004*
- [HF04b] Hamza H., Fayad M.E., Applying Analysis Patterns Through Analogy: Problems and Solutions: in Journal of Object Technology, 3,4 April 2004
- [H01] Hahsler M., Software Engineering with Analysis Patterns, Technical Report of WU-Wien, 2001
- [IIBA] International Institute of Business Analysis, <http://www.theiiba.org/am/>
- [RUP] IBM Rational Unified Process, Academic Initiative Repository.
- [UML] OMG Unified Modeling Language, v 2.1.1., [www.uml.org](http://www.uml.org).
- [P] Pattern Languages of Programs (PLOP), <http://hillside.net/plop/>

- [OMT] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W., "Object-oriented modeling and design.", Prentice Hall Int., 1991.
- [S00] Sesera L., A Recurring Fulfilments Analysis Pattern, *Proceedings of PloP 2000*.
- [SF04] Sorgente T., Fernandez E.B., "Analysis patterns for patient treatment, *Proceedings of PLoP 2004*.
- [SFL05] Sorgente T., Fernandez E.B., Larrondo-Petrie M.M., "The SOAP pattern for medical charts", in *Proceedings of the 12th Pattern Languages of Programs Conference (PLoP2005)*.

# Patterns 2.0: a Service for Searching Patterns\*

Aliaksandr Birukou<sup>†</sup>

Michael Weiss

DISI - University of Trento, Italy

SCE - Carleton University, Canada

January 22, 2010

## Abstract

With ever-increasing number of patterns in the literature and online repositories, it can be hard for non-experts to know about new patterns and select patterns appropriate to their needs. We argue that a systematic way for searching patterns is required and we present the Patterns 2.0 service, a composite software service for facilitating pattern search and selection. The service combines several pattern-related services with a recommendation service that allows users to share their experiences in using patterns. The contributions of the paper are: the overview of existing services related to the problem of pattern selection, the definition of Patterns 2.0 service, and description of its possible uses.

## 1 Introduction

Given the steadily growing number of patterns in the literature and online repositories, it can be hard for non-experts to select patterns appropriate to their needs, or even to be aware of the patterns that exist. In this paper, we present an overview of existing software services related to pattern selection and propose a composite software service that facilitates pattern selection. The service can combine existing pattern retrieval services with a recommendation service that allows users to share their experiences in using patterns. We also provide different usage scenarios of that composite service.

Almost fifteen years ago, the GoF stated the problem of selecting patterns: “With more than 20 design patterns in the catalog to choose from, it might be hard to find the one that addresses a particular design problem, especially if the catalog is new and unfamiliar to you” [8]. As time passed, patterns have become an integral part of many development approaches. However, the problem of selecting patterns still exists. If anything, it has become more critical, as the number of documented patterns has continually increased: for instance, Rising’s *Pattern Almanac* [17]

---

\*Copyright retain by authors. Permission granted to Hillside Europe for inclusion in the CEUR archive of proceedings and for Hillside Europe website

<sup>†</sup>This research is partly supported by the EU FP7 projects COMPAS and LiquidPub

lists more than 1200 patterns. In the past nine years since the publication of the almanac, many new patterns and books on patterns have been published. The domains containing more than ten patterns, the problem of choosing the appropriate pattern is particularly hard to solve for inexperienced programmers [18]:

Only experienced software engineers who have a deep knowledge of patterns can use them effectively. These developers can recognize generic situations where a pattern can be applied. Inexperienced programmers, even if they have read the pattern books, will always find it hard to decide whether they can reuse a pattern or need to develop a special-purpose solution.

This quote also suggests that experienced software engineers rely on their knowledge to select patterns to apply in a given context. Over time, they build up a good understanding of which patterns apply to their domain. However, they also tend to be less aware of more recently documented patterns. (This becomes very clear when we consider that for many developers the notion of patterns still stops at the GoF book.) Developers with less experience may also ask for advice from friends or colleagues. However, such interactions are highly personalized and rarely documented, that is, this knowledge remains tacit. May [14] observes that patterns have made design knowledge explicit, the process of applying patterns has become itself new tacit knowledge. Several tools for assisting in the process of pattern selection have been developed to make the knowledge underlying the application of patterns explicit.

Although the problem of pattern selection can be considered a particular instance of the general problem of retrieval of relevant information from large document collections [5], it requires specialized tools for a number of reasons: (i) patterns are structured documents where different parts express very different types of information; (ii) they are often linked to each other in a pattern language; and (iii) design patterns accumulate the experience of developers in dealing with design problems. Therefore, besides search engines for patterns such as PatternSeer<sup>1</sup>, tools for managing pattern catalogs (see Deng [4] for an overview) and wikis such as PatternForge<sup>2</sup> and Planet<sup>3</sup>, existing approaches for supporting pattern selection include case tools [9], expert systems [13], recommendation systems [3], and formal frameworks that help reuse knowledge about patterns (see Weiss [19] for an overview of several such systems).

However, existing approaches that support pattern users in the selection of patterns have several shortcomings: (i) they usually require additional effort during the authoring and selection process (e.g. authors need specify metadata about their patterns); (ii) pattern repositories require effort in maintaining and updating information; (iii) they often targeted at developers, helping them to select architectural or design patterns, while there are also patterns on organizing conferences or meetings, computer-mediated interaction patterns, which are used by non-developers<sup>4</sup>; (iv) they rarely support collaboration and personalized recommendations.

---

<sup>1</sup><http://doc-it.fe.up.pt/aaguiar/space/Projects/PatternSeer>

<sup>2</sup><http://www.patternforge.net/wiki>

<sup>3</sup><http://patternlanguagenetwork.org>

<sup>4</sup>Therefore in the following we use a more general term “pattern user”

Thus, as May [14] says, much of the information how patterns are selected by users remains tacit despite the existence of these tools.

In this paper we present an overview of existing services for pattern search and selection and propose the Patterns 2.0<sup>5</sup> service, a composite service for facilitating pattern selection. The service combines existing pattern retrieval, tagging, and recommendation services. The core contribution is improving pattern search and providing assistance for pattern selection by combining the services and integrating the recommendation service for tracking pattern usage history. The latter provides support for social factors (tacit knowledge about how patterns are used within an organization), collaboration (potential for linking users) and personalization (who prefers which patterns or domains).

The proposed service address the outlined shortcomings in existing solutions for pattern selection in the following ways: (i) it improves searching by using tagging, usage history; (ii) using community-generated content allows for minimizing the effort in maintaining and updating information in pattern repositories; (iii) the service is orthogonal to the format and domain of patterns, and can be used by different communities either collaboratively (sharing usage data between different communities), or in isolation (each community consumes recommendations based on its own usage data).

The primary audience for this paper are developers of pattern repositories and pattern retrieval systems, as well as researchers on the application of patterns.

The paper has the following structure: in Section 2 we review existing approaches for pattern search and selection. In Section 3 we describe the proposed service and requirements on the services it invokes, while in Section 4 we discuss limitations and possible extensions of our approach. We conclude the paper in Section 5.

## 2 Approaches to pattern selection

Recently, there have been several efforts in making patterns available in online pattern repositories, where they can be browsed and searched by various criteria. An early example was the Pattern Almanac [17], which is also available in electronic form<sup>6</sup>. More recent examples are the PatternShare<sup>7</sup> site hosted by Microsoft between 2006-2007, Yahoo Design Pattern Library<sup>8</sup>, Sun collection of J2EE patterns<sup>9</sup>, and computer-mediated interaction patterns<sup>10</sup>. In this section, we review existing approaches for selecting patterns stored in such repositories.

In order to store patterns in a repository, a structured pattern representation must be adopted. There have been several proposals for structural pattern representation, most notably the Pattern Language Markup Language (PLML) [6] and Entity Meta-Specification Language (EML) [20].

---

<sup>5</sup>2.0 in the name is from Web 2.0, because the service uses tagging and other community-generated content

<sup>6</sup>[www.smallmemory.com/almanac](http://www.smallmemory.com/almanac)

<sup>7</sup>[patternshare.org](http://patternshare.org)

<sup>8</sup><http://developer.yahoo.com/ypatterns/>

<sup>9</sup><http://java.sun.com/blueprints/patterns/>

<sup>10</sup><http://www.cmi-patterns.org/>

Existing online repositories rarely contain personalized features, although they can provide customizable pattern properties for enhancing search [10]. To the best of our knowledge, most of them remain oblivious to the advent of Web 2.0 and list content defined by the repository creator and provide no tagging, bookmarking and other social features. The sad thing about this is no matter how heavily the repository is used for searching patterns, it does not change and improve over time, if not maintained. However, several wiki-based repositories such as PatternForge were created recently trying to overcome such shortcomings and to use the power of the community in order to enrich repositories with tags, links and other user-generated content.

There are several search engines for patterns. PatternSeer is an ongoing project that aims at delivering a system that crawls and indexes pattern descriptions on the Internet and makes them accessible to users via keyword-based search. The problem with current solutions is their limited coverage of patterns. This reminds one of the problems early Internet had – just eleven years ago it was better to use several search engines to get more different results for a query.

Several approaches exploit past user experience in order to suggest suitable patterns. The ReBuilder [9] framework adopts a case-based reasoning approach, where cases represent situations in which a pattern was applied in the past to a software design. ReBuilder supports the retrieval and adaptation of patterns. Cases are described in terms of class diagrams. Cases are retrieved based on a combination of structural similarity between the current design and a pattern, as well as the semantic distance between class names and role names in the pattern.

The authors developed a recommendation system for pattern selection [3] which is complementary to systems like ReBuilder: in this system, patterns are selected on the basis of previous actions by other users. Also, while the use of the relations in a class diagram provides additional information about the desired pattern, such diagrams are not always available. However, since our system uses textual descriptions, and does not require an object model, it has a wider range of potential applications. However, it probably cannot compete with ReBuilder in domains where class diagrams are available. Finally, our system implements a collaborative approach to pattern selection, facilitating experience sharing among users.

Several approaches propose adding formal semantics to pattern descriptions (see Weiss [19] for an overview). As most patterns are organized in pattern languages, some approaches target the selection of pattern(s) from such languages, thus handling relations between patterns, not only individual pattern descriptions. Zdun [21] proposes an approach for pattern selection based on desired quality attributes and pattern relations. The approach requires formalizing the pattern relationships in a pattern language grammar and annotation of the patterns with their effects on quality goals. As a result, the search space is narrowed down and the time spent evaluating alternatives is decreased. Mussbacher et al. [15] present a goal-oriented requirement language that formalizes the forces of patterns and relations between patterns.

Most of the existing approaches require additional efforts, such as specifying additional information about patterns or their relations, creating a knowledge base, or organizing the collection in a specific way. On the contrary, our system can handle different repositories and pattern engines and provide recommendations and other social features. As we show in the



sections below, our pattern recommendation service addresses some of the shortcomings of existing approaches for pattern selection by combining several pattern-related services and enhancing them with social features such as recommendations and tagging. The only additional (and optional) effort required is that of providing feedback, but, as we show, in some cases this can be automated.

### 3 Patterns 2.0: a composite service for pattern selection

The Patterns 2.0 service composes other existing services for delivering pattern-related content to the user. It is intended to be used by pattern users (including developers) and pattern writers. In the following subsections, we present the architecture and use cases of the Patterns 2.0 service.

#### 3.1 Architecture

The Patterns 2.0 service is essentially a combination of existing services related to pattern selection. It takes as input user queries, forwards them to the appropriate services and integrates the results. The aim of the service is to improve search of patterns and to provide assistance for pattern selection. The nature and the purpose of the services composing the Patterns 2.0 is explained in this section.

The Patterns 2.0 service combines the following services as shown in Figure 1:

- *Pattern retrieval service.* A service that provides such functionalities as searching and retrieving patterns from a *pattern repository* that stores pattern descriptions. Planet is an example of such a service. It is wiki that allows authors to contribute patterns.
- *Pattern engine service.* A service with functionalities similar to those of a pattern retrieval service, but with the key difference that the system only maintains a *meta-index* to patterns described in full elsewhere. PatternSeer is an example of such a service.
- *Pattern recommendation service.* A service that provides recommendations<sup>11</sup> about patterns using a database of *pattern usage history* collected from past user interactions with the service. An example of such service is IC-Service, a general-purpose recommendation service [2], whose application to the problem of pattern selection is described in [3]. Other examples are described in [9, 13].
- *Pattern tagging service.* A service with functionalities similar to those existing on many pattern wikis: users can annotate patterns via tags, and patterns can be retrieved by an external service based on tags. Examples include PatternForge and Planet.

---

<sup>11</sup>By recommendations we mean hints on patterns that may be of interest to the user, considering the submitted query, something the user may (as opposed to must) find relevant

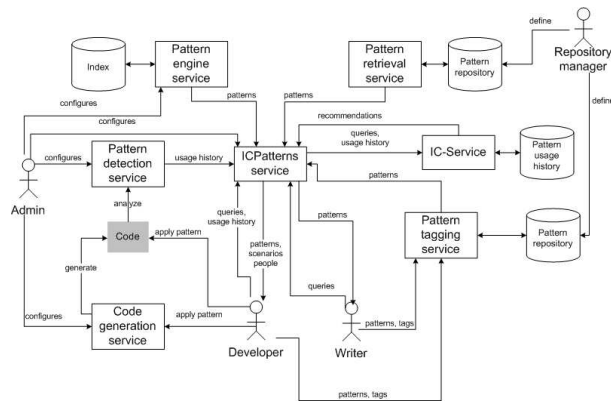


Figure 1: Architecture of the Patterns 2.0 service. The arrows denote the invocation flow and the labels denote information passed

Note, that we do not restrict pattern repository to GoF patterns. Instead, we assume that several repositories that use different pattern representation formats (PLML, etc.) and domains (patterns about security, organizing meetings, architectural patterns [12]).

Let us define possible API for the described services, in order to give more details about what is expected from them. The API are defined using pseudo code.

**Patterns 2.0 service:**

`Pattern[] getPatternsByProblem(problemDescription:String, context:Context)` - this function returns patterns relevant to the specified `problemDescription` possibly in a given context.

`Pattern[] getPatternsByKeywords(keywords:String, context:Context)` - this function returns patterns relevant to the specific keywords, possibly in a given context.

`saveFeedback(pattern:Pattern, keywords:String, accepted:Boolean)` - this function saves feedback: which pattern was accepted/rejected for a set of keywords.

**Patterns retrieval service:**

`Pattern[] getPatterns(keywords:String)` - this function returns patterns relevant to the specified keywords.

**Patterns engine service:**

`URL[] getPatterns(keywords:String)` - this function returns URLs to patterns relevant to the specified keywords.

**Patterns recommendation service:**

`Pattern[] getPattern(keywords:String)` - this function returns URLs to patterns relevant to the specified keywords.

`saveFeedback(pattern:Pattern, keywords:String, action:Action)` - this function saves feedback: which pattern was found relevant for a set of keywords. Here `action` refers to the feedback action, which depends on the type of user. For instance, in case of a pattern user, such actions can be applying pattern or marking it as not relevant. In case of writer, an example of feedback action could be rating the pattern.

**Patterns tagging service:**

`Pattern[] getPatternsByTags(tags:String)` - this function returns tagged with all specified tags.

The goal of the Patterns 2.0 service is to process the query submitted by a pattern user or a pattern writer

and transform it into several ad-hoc queries that will be forwarded to the appropriate service. For instance, to get results from the pattern tagging service, the query should be transformed into relevant tags and the `getPatternsByTag` function should be invoked. Furthermore, the feedback provided by the user is propagated to the recommendation service, to be stored in the usage history.

A query user submits to the Patterns 2.0 service includes a description of the problem and an optional context. The problem is described by a set of keywords, optionally restricted to specific elements of the pattern description, e.g. context or problem statement. An example of such a description could be “improve access control”, or, in case of restriction: “improve access control, CONTEXT, SOLUTION”. An example of the context could be the set of patterns already deployed in the project where the problem is encountered, e.g. “Authorization, Authenticator, TrustedProxy”. The use of context can be used by the recommendation service in order to improve recommendations. For instance, to find in the usage history the situation, which is the most similar to the current search, the IC-Service calculates the similarity between users in terms of their past actions (queries and feedback). We believe that the personalization and contextualization of the query should allow for providing more relevant results than when only the simple keyword-based search already supported by pattern engine and pattern retrieval services is used. However, since the recommendation service does not host pattern descriptions, but only actions users performed on patterns, one or several other services (pattern retrieval service, pattern engine service, and pattern tagging service) are required for answering user queries.

There are three types of recommendations supported by the recommendation service:

- **Recommending patterns.** Recommending patterns suitable for solving a specific problem. Patterns matching a specific problem are returned in response to a query.
- **Recommending key patterns in a specific area.** Suggesting a list of patterns essential to a certain class of problems, or to the understanding of a particular repository of patterns (i.e. what is the best order to read the patterns in order to learn to use them).
- **Recommending pattern sequences.** Similar to recommending patterns, but recommendations consist of sequences of patterns to apply in a given situation. This takes relations between patterns into account. Sequences can also be mined from pattern usage.

The algorithms used for producing the recommendations are outside the scope of this paper and can be found elsewhere [3]. The purpose of this paper is to describe an architecture that embeds the recommendation service as part of an integrated system for pattern selection and application.

## 3.2 Use cases

The Patterns 2.0 service can be used in different ways: as a component of an ad-hoc pattern management system within an organization, on online pattern sites, as a plug-in into an IDE for developers, and so on. In the following, we describe general use of Patterns 2.0, classify potential users of the Patterns 2.0 service, and discuss example scenarios.

### 3.2.1 General description of use of Patterns 2.0

Figure 1 depict the invocation flow, described in this subsection. A user accesses the service by submitting a query via the user interface. We assume that the Patterns 2.0 service can be accessed in a number of ways: from a browser, from a plug-in to an IDE, or similar. After the Patterns 2.0 service receives a query, the service forwards it to the recommendation service. The problem description part of the query is forwarded

to the pattern engine and pattern retrieval services. Tags extracted from the query are forwarded to the pattern tagging service. Each invoked service responds to the Patterns 2.0 service with a list of patterns, which the Patterns 2.0 service combines in the results sent back to the user. In case of a pattern user, i.e. is searching for patterns to solve a specific problem, they can also get descriptions of situations where other users have used the pattern or a list of those users so that it is possible to discuss the problem with them.

At some point after getting the results, the pattern user applies one or several patterns. We provide the possibility for submitting this information to the Patterns 2.0 service as part of the usage history, i.e. feedback actions in connection with the previously submitted query. Such feedback is passed to the pattern recommendation service. In case a writer performs a search, the feedback contains relevance of the results, i.e. whether returned patterns are indeed related to the queried topic. In case of pattern user, the feedback contains information about which patterns were applied to which queries. Obviously, the key problem lies in the “observability” of the users’ feedback actions, i.e. actions of using a pattern for a problem, or finding it relevant to the topic. In case of the pattern user, they can explicitly indicate that the pattern X has been selected for the problem A, where the problem corresponds to a search in the history of searches. The key challenge is providing them with motivation of doing this additional action. However, learning from Web 2.0 lessons, there should be a way for convincing users to provide feedback if they see value in recommendations they get as the result of this collaborative effort.

In case of the developer, the following two options can be also considered for observing feedback actions:

1. A pattern detection service (such as [1]) processes project documents and code, but in this case it is still non-trivial to link the detected patterns back to the query.
2. A case tool which supports (semi-)automatic implementation of patterns (such as [16]) could be used. The actions of the tool would provide the required trace. However, this case is limited to widely known patterns, such as GoF patterns.

In case of the writer, implicit relevance feedback (i.e. the feedback not requiring additional actions from the writer), such as clickthrough rate, time spent reading pattern description, and how the search session ended (see [7] for an overview) can be used.

### 3.2.2 Users of Patterns 2.0

Potential users of the Patterns 2.0 service can be classified in several groups:

- Pattern users who lack experience in applying patterns such as students, trainees or interns, people who are on their first architect/developer job, or have rarely used patterns before.
- People who have experience in using patterns, and, for instance, know how and when to apply the GoF patterns, but are unfamiliar with a specific pattern collection.
- Pattern writers who would like to find patterns related to patterns they are authoring.

Concerning the first and second group of users, the proposed tool could be very effective for organizations who maintain a pattern repository and infer which patterns are most useful under which conditions from the users’ interaction with the repository.

With respect to the general architecture, users can play several roles when interacting with the various services:

- **Admin.** Configures services for a specific group of users.
- **Repository manager.** Defines the collection of patterns in a pattern repository.

- **Pattern user.** Interacts with the Patterns 2.0 service to get recommendations.
- **Pattern writer.** Interacts with the service to find patterns on a specific topic.

### 3.2.3 Possible uses of Patterns 2.0

To clarify the main uses of the Patterns 2.0 service, we consider the following specific “profiles” of the proposed architecture:

- **Poogle**<sup>12</sup>. In this profile, Patterns 2.0 is used to search patterns using pattern engine and pattern retrieval services.
- **Plickr**<sup>13</sup>. The use of Patterns 2.0 for tagging patterns and searching them using tags.
- **PatternLens**<sup>14</sup>. The use of Patterns 2.0 as a recommendation system, where users get recommendations about patterns to apply for solving a specific problem.

## 3.3 Example

In this section, we use an example to illustrate a possible use of the Patterns 2.0 service.

Let us assume that the Patterns 2.0 service uses a repository of security patterns, as configured by the Admin. The patterns in the repository defined by Repository Manager and are from the collection of security patterns previously hosted in [patternshare.org](http://patternshare.org) [11]. Let us consider a developer who needs to improve access control in a system that offers multiple services. Suppose that for an experienced developer it is apparent to use the Single Access Point pattern.

However, our user does not know this, and therefore submits a query with the following problem description: “complex security control”. The Patterns 2.0 service obtains the Single Access Point and Role Based Access Control patterns as results from the other services, discovers that other users previously used Single Access Point for similar problems and sends this information to the user. The developer then submits the feedback on the recommended patterns to the system.

Let us now consider a pattern Writer, who is preparing a pattern language on security in mobile applications, and would like to find related patterns and pattern languages. The user can submit a query on “secure mobile applications” to the Patterns 2.0 service and it will search for related patterns in pattern repositories and pattern search engines. The developer can browse through the list of results to see if there are related patterns to cite in his language. One can imagine an extension of this scenario, where query consists of patterns already present in the language, similarly to query-by-example approach.

## 4 Discussion

Since the Patterns 2.0 service does not store pattern descriptions, there is no copyright issues involved.

---

<sup>12</sup>This name combines “patterns” and “Google”, one of the most popular search engines

<sup>13</sup>This name combines “patterns” and “Flickr”, one of the most popular services for sharing (and tagging) photos

<sup>14</sup>One of the first systems for recommending movies was named “MovieLens”, followed by several systems with similar names in different domains

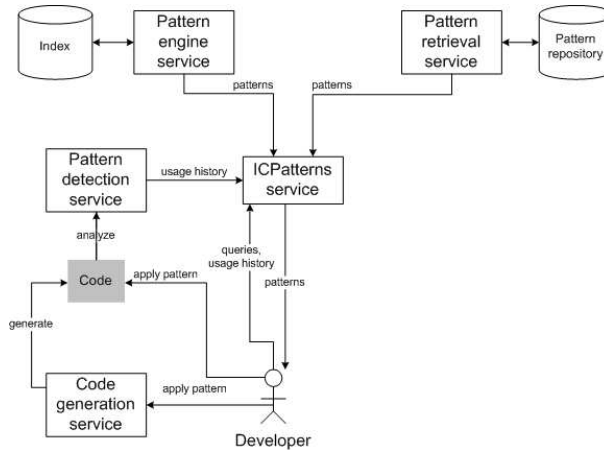


Figure 2: Using Patterns 2.0 for applying and detecting patterns in code.

The quality of Patterns 2.0 results obviously depends on the quality of its component services. One of possible future work directions can be investigation if Patterns 2.0 can perform better than the component services.

We can consider one additional use case for Patterns 2.0, shown in Figure 2. In this case, ad-hoc tools are adopted for automation of applying patterns and detecting them in the code. This Figure introduces two new services:

- *Code generation service.* A service that can generate code templates implementing the selected pattern. An example is described by O’Cinneide and Nixon [16].
- *Pattern detection service.* A service that analyses code to determine instances of patterns that occur in the code. The PTIDEJ tool [1] provides such functionality.

Such services can be also used by the Patterns 2.0 service for gathering feedback on chosen patterns and for providing pattern usage history, correspondingly. However, in the architecture of the Patterns 2.0 service, described in this paper, we leave out the description of tools that automatically generate code implementing patterns (code generation services) and of tools that automatically detect patterns in the code. The reasons why we do not include these tools are: (i) Even though several approaches for automatic detection and code generation have been proposed, the tools remain at the stage of prototypes and were not taken by the mainstream of IDE tools; (ii) such tools are usually limited to GoF patterns, while our solution is more general and does not depend on the repository; (iii) we would like to beyond software patterns, recommending also other types of patterns (e.g., patterns for organizing meetings or conferences), and such patterns are not related to code.

In this work we do not consider pattern languages that organize patterns in collections. The use of such languages can provide additional information for improving pattern selection, especially in the case of recommending sequences of patterns. The inclusion of pattern languages can be one of future work directions.

## 5 Conclusion

We presented the architecture and use cases of a composite service for facilitating the selection of patterns. A unique aspect of this service is its use of different types of services and sharing experiences among pattern users. The service composes other existing services (such as existing pattern repositories or pattern tagging services) for delivering pattern-related content to the user. Future work will include the definition of standard APIs for the services composing Patterns 2.0 and implementation of the service. This goal requires a collaborative effort of the creators of pattern management tools, and progress towards this end has been made over the last year. Information about the further development of this work will be available at [http://disi.unitn.it/~birukou/pattern\\_selection](http://disi.unitn.it/~birukou/pattern_selection).

## Acknowledgement

We would like to thank our shepherd Klaus Marquardt for his many insightful comments.

## References

- [1] H. Albin-Amiot, P. Cointe, Y.-G. Gueheneuc, and N. Jussien. Instantiating and detecting design patterns: Putting bits and pieces together. In *International Conference on Automated Software Engineering*, pages 166–173, 2001.
- [2] A. Birukou, E. Blanzieri, V. D’Andrea, P. Giorgini, N. Kokash, and A. Modena. IC-Service: A service-oriented approach to the development of recommendation systems. In *Proceedings of ACM Symposium on Applied Computing. Special Track on Web Technologies*, 2007.
- [3] A. Birukou, E. Blanzieri, P. Giorgini, and M. Weiss. A multi-agent system for choosing software patterns. Technical Report DIT-06-065, University of Trento, 2006.
- [4] J. Deng, E. Kemp, and E. G. Todd. Managing UI pattern collections. In *ACM SIGCHI New Zealand Chapter’s International Conference on Computer-Human Interaction*, pages 31–38, New York, NY, USA, 2005. ACM.
- [5] W. Fan, M. Gordon, and P. Pathak. On linear mixture of expert approaches to information retrieval. *Decision Support Systems*, 42(2):975–987, November 2006.
- [6] S. Fincher. PLML: Pattern language markup language. report of workshop held at CHI, Interfaces, 56 (pp. 26-28). Technical report, 2003.
- [7] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *ACM Trans. Inf. Syst.*, 23(2):147–168, 2005.

- [8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1995.
- [9] P. Gomes, F. C. Pereira, P. Paiva, N. Seco, P. Carreiro, J. L. Ferreira, and C. Bento. Using CBR for automation of software design patterns. In *European Conference on Advances in Case-Based Reasoning*, pages 534–548, London, UK, 2002. Springer-Verlag.
- [10] S. L. Greene, P. M. Matchen, L. Jones, J. C. Thomas, and M. Callery. Tool-based decision support for pattern assisted development. In *CHI 2003 workshop on HCI Patterns: Concepts and Tools*, 2003.
- [11] M. Hafiz, P. Adamczyk, and R. E. Johnson. Organizing security patterns. *Software, IEEE*, 24(4):52–60, 2007.
- [12] N. Harrison, P. Avgeriou, and U. Zdun. Architecture patterns as mechanisms for capturing architectural decisions. *IEEE Software*, July-August 2007.
- [13] D. C. Kung, H. Bhambhani, R. Shah, and G. Pancholi. An expert system for suggesting design patterns: a methodology and a prototype. In T. M. Khoshgoftaar, editor, *Software Engineering With Computational Intelligence*, Series in Engineering and Computer Science. Kluwer International, 2003.
- [14] D. May and P. Taylor. Knowledge management with patterns. *Communications of the ACM*, 46:94–99, 2003.
- [15] G. Mussbacher, M. Weiss, and D. Amyot. Formalizing architectural patterns with the Goal-oriented Requirement Language. In *Nordic Pattern Languages of Programs Conference*, September 2006.
- [16] M. O’Cinneide and P. Nixon. Automated software evolution towards design patterns. In *International Workshop on Principles of Software Evolution*, pages 162–165. ACM, 2001.
- [17] L. Rising. *The Pattern Almanac*. Addison-Wesley Longman Publishing Co., Inc., 2000.
- [18] I. Sommerville. *Software engineering*. Addison-Wesley, Boston, MA, USA, 7th edition, 2004.
- [19] M. Weiss. Patterns and their impact on system concerns. In *European Conference on Pattern Languages of Programs*, 2008.
- [20] L. Welicki, J. M. C. Lovelle, and L. J. Aguilar. Patterns meta-specification and cataloging: Towards a more dynamic patterns life cycle. In *International Workshop on Software Patterns: Addressing Challenges at COMPSAC 2007*, 2007.
- [21] U. Zdun. Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience*, 37(9):983–1016, 2007.



---

# Applying Architectural Patterns for Parallel Programming

## Solving the One-dimensional Heat Equation

**Jorge L. Ortega Arjona\***

Departamento de Matemáticas

Facultad de Ciencias, UNAM.

jloa@ciencias.unam.mx

### Abstract

The Architectural Patterns for Parallel Programming is a collection of patterns related with a method for developing the coordination of parallel software systems. These architectural patterns take as input information (a) the available parallel hardware platform, (b) the parallel programming language of this platform, and (c) the analysis of the problem to solve, in terms of an algorithm and data.

In this paper, it is presented the application of the architectural patterns along with the method for developing a coordination for solving the One-dimensional Heat Equation. The method used here takes the information from the Problem Analysis, proposes an architectural pattern for the coordination, and provides elements about its implementation.

## 1 Introduction

A parallel program is *the specification of a set of processes executing simultaneously, and communicating among themselves in order to achieve a common objective* [16]. This definition is obtained from the original research work in parallel programming provided by E.W. Dijkstra [4], C.A.R. Hoare [7], P. Brinch-Hansen [2], and many others, who have established the main basis for parallel programming today. Practitioners in the area of parallel programming recognize that the success of a parallel program is able to achieve –commonly, in terms of performance– is affected by three main factors: (a) the hardware platform, (b) the programming language, and (c) the problem to solve.

Nevertheless, parallel programming still represents a hard problem to the software designer and programmer: we do not yet know how to solve an arbi-

---

\*Copyright retain by author. Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.

---

trary problem efficiently on a parallel system of arbitrary size. Hence, parallel programming, at its actual stage of development, does not (cannot) offer universal solutions, but tries to provide some simple ways to get started. By sticking with some common parallel *coordinations*, it is possible to avoid a lot of errors and aggravation. Many approaches have been presented up to date, proposing descriptions of top-level coordinations observed in parallel programs. Some of these descriptions are: *Outlines of the Program* [3], *Programming Paradigms* [8], *Parallel Algorithms* [5], *High-level Design Strategies* [9], and *Paradigms for Process Interaction* [1]. These descriptions provide common overall coordinations such as, for example, “master-slave”, “pipeline”, “work-pile”, and others. They represent assemblies of parallel software components which are allowed to simultaneously execute and communicate. Furthermore, these descriptions are expected to support the design of parallel programs, since all of them introduce common forms that such assemblies exhibit.

The *Architectural Patterns for Parallel Programming* [10, 11, 12, 13, 14, 15] represent a Software Patterns approach for designing the coordination of parallel programs. These Architectural Patterns attempt to save the transformation “jump” between algorithm and program. They are defined as *fundamental organizational descriptions of common top-level structures observed in parallel software systems* [10], specifying properties and responsibilities of their sub-systems, and the particular form in which they are assembled together into a coordination.

Architectural patterns allow software designers and developers to understand complex software systems in larger conceptual blocks and their relations, thus reducing the cognitive burden. Furthermore, architectural patterns provide several “forms” in which software components of a parallel software system can be structured or arranged, so the overall coordination of such a software system arises. Architectural patterns also provide a vocabulary that may be used when designing the overall coordination of a parallel software system, to talk about such a structure, and feasible implementation techniques. As such, the Architectural Patterns for Parallel Programming refer to concepts that have formed the basis of previous successful parallel software systems.

The most important step in designing a parallel program is to think carefully about its overall coordination. The Architectural Patterns for Parallel Programming provide descriptions about how to organize a parallel program, having the following advantages [10, 11, 12, 13, 14, 15]:

- The Architectural Patterns for Parallel Programming (as any Software Pattern) provide a description that links a problem statement (in terms of an algorithm and the data to be operated on) with a solution statement (in terms of an organization or coordination of communicating software components).
- The partition of the problem is a key for the success or failure of a parallel

---

program. Hence, the Architectural Patterns for Parallel Programming have been developed and classified based on the kind of partition applied to the algorithm and/or the data present in the problem statement.

- As a consequence of the previous two points, the Architectural Patterns for Parallel Programming can be selected depending on characteristics found in the algorithm and/or data, which drive the selection of a potential parallel structure by observing and studying the characteristics of order and dependence among instructions and/or datum.
- The Architectural Patterns for Parallel Programming introduce parallel structures or coordinations as forms in which software components can be assembled or arranged together, considering the different partitioning ways of the algorithm and/or data.

Nevertheless, even though the Architectural Patterns for Parallel Programming have these advantages, they also present the disadvantage of not describing, representing, or producing a complete parallel program in detail. Other Software Patterns are still needed for achieving this. Anyway, the Architectural Patterns for Parallel Programming are proposed as a way of helping a software designer to select a parallel structure as a starting point when designing a parallel program. For a complete exposition of the Architectural Patterns for Parallel Programming, refer to [10], and further work on each particular architectural pattern in [11, 12, 13, 14, 15].

## 2 Problem Analysis – The One-dimensional Heat Equation

The present paper attempts to demonstrate the use of the Architectural Patterns for Parallel Programming for designing a coordination that solves the One-dimensional Heat Equation. The objective is to show how an architectural pattern can be selected and applied so it deals with the functionality and requirements present in this problem.

### 2.1 Problem Statement

Partial differential equations are commonly used to describe physical phenomena that continuously change in space and time. One of the most studied and well known of such equations is the Heat Equation, which mathematically models the steady-state heat flow in a region that exposes certain dimensionality, with certain fixed temperatures on its boundaries. In the present example, the region is represented by a one-dimensional entity, for example, a wire of homogeneous material and uniform thickness. The surroundings of the wire are perfectly insulated, and on the extremes, each point keeps a known, fixed temperature. As heat flows through the wire, the temperature of each point eventually reaches a

---

value or state in which such a point has a steady, time-independent temperature maintained by the heat flow. Thus, the problem of solving the One-dimensional Heat Equation is to define the equilibrium temperature  $u(x)$  for each point  $x$  on the one-dimensional wire. Normally, the heat is studied as a flow through an elementary piece of the wire, a finite element. This element is represented as a small, one-dimensional segment of the wire, with a length of  $\Delta x$  (Figure 1).

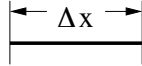


Figure 1: A small one-dimensional element.

Given the insulation surrounding the wire, there could only be a flow through its only dimension. At every point  $x$ , the velocity of the heat flow is considered to have a horizontal flow component,  $v_x$ , which is represented in terms of its temperature  $u(x)$  by the equation:

$$v_x = -k \frac{\partial u}{\partial x}$$

This equation means that heat flow is proportional to the temperature gradient, towards decreasing temperatures. Moreover, in equilibrium, the element holds a constant amount of heat, making its temperature  $u(x)$  a constant. Thus, in the steady-state, this is expressed as:

$$\frac{\partial v_x}{\partial x} = 0$$

Combining this equation with the previous equation for the velocity of flow, thus Laplace's law for equilibrium temperatures arises:

$$\frac{\partial^2 u}{\partial x^2} = 0$$

Known as the one-dimensional heat equation or equilibrium equation, this equation is abbreviated and expressed in general terms (and dimensions) as:

$$\nabla^2 u = 0$$

A function  $u(x)$  that satisfies this equation is known as a "potential function", and it is determined by boundary conditions. By now, for the actual purposes, the One-dimensional Heat Equation allows to mathematically model

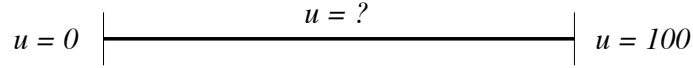


Figure 2: A wire with fixed temperatures at each extreme.

the heat flow through a wire. Nevertheless, in order to develop a program that numerically solves this equation, it is still required to perform a series of further considerations. Let us consider by now a thin wire, for which temperatures are considered fixed at each extreme (Figure 2).

In order to develop a program that models the Heat Equation, first it is necessary to obtain its discrete form. So, the wire in Figure 2 is divided into segments, each segment with a size of  $h$ . This size is relatively very small regarding the size of the whole wire, so the segment can be considered as a single point within the wire. So, this results on a segmented wire, in which two types of segments can be considered (Figure 3).

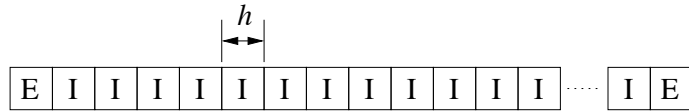


Figure 3: A segmented wire with two types of elements: interior (I) and extreme (E).

1. Interior segments, which require computing their temperatures, each one having to satisfy the heat equation.
2. Extreme segments, which have fixed and given temperatures.

The discrete solution of the Heat Equation is based on the idea that the heat flow through interior elements is due to the temperature differences between an elements and all its neighbors. Let us suppose the temperature of a single interior element  $u(i)$ , whose two adjacent neighboring elements are  $u(i - 1)$  and  $u(i + 1)$  (Figure 4).

Notice that for the case,  $h$  should be small enough so each neighboring element's temperature can be approximated in terms of a Taylor expansion. So, the discrete heat equation is reduced to a difference equation. Rearranging it, it is noticeable that for thermal equilibrium, the temperature of a single element  $u(i)$  in time, from one thermal state to another, is:

$$u(t + 1, i) \approx u(t, i) + \frac{1}{h^2}(u(t, i - 1) + u(t, i + 1) - 2u(t, i))$$

---

This is the discrete equation to be used in order to obtain a parallel numerical solution for the One-dimensional Heat Equation.

## 2.2 Specification of the Problem

From the previous section, it is noticeable that using a wire segmented into  $n$  segments, the discrete form of the Heat Equation implies a computation for each discrete segment of the wire. Moreover, taking into consideration the time as another dimension so the evolution of temperatures through time can be observed, and solving it using a direct method on a sequential computer, requires something like  $O(n^3)$  units of time. Suppose a numerical example: for a wire with, for example,  $n = 65536$ , it is required to solve about the same number of average operations, involving floating point coefficients. Using a sequential computer with a clock frequency of about 1MHz, it would take about eight years for the computation. Furthermore, notice that naive changes to the requirements (which are normally requested when performing this kind of simulations) produce drastic (exponential) increments of the number of operations required, which at the same time affects the time required to calculate this numerical solution.

- *Problem Statement.* The One-dimensional Heat Equation, in its discrete representation, and for a relatively large number of segments in which a wire is divided, can be computed in a more efficient way by:
  1. using a group of software components that exploit the one-dimensional logical structure of the wire, and
  2. allowing each software component to simultaneously calculate the temperature value for all segments of the wire at a given time step.

The objective is to obtain a result in the best possible time-efficient way.

- *Descriptions of the data and the algorithm.* The relatively large number of segments in which a wire is divided and the discrete representation of the One-dimensional Heat Equation is described in terms of data and an algorithm. The divided region is normally represented as a long wire in terms of a  $(n+2)$  array of segments which represent every discrete element of the wire, and encapsulate some floating point data which represents

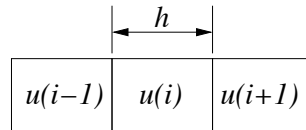


Figure 4: An element  $u(i)$  and its two neighboring elements.

---

temperature, as shown as follows. Thus, a whole wire consists of  $n$  interior segments and 2 extreme segments.

```
class Segment implements Runnable{
    ...
    private int i = -1;
    ...
    private Segment(int i){
        this.i = i;
        new Thread(this).start();
    }
    ...
}
```

Each **Segment** object is able to compute a local discrete heat equation as a single thread. Thus, it exchanges messages with its neighboring segments (whether interior or extreme) and computes its local temperature, as follows:

```
class Segment implements Runnable{
    ...
    private int i = -1;
    ...
    public void run(){
        double temperature, received, total;
        for (int i = 0; i < iterations; i++) {
            // Here the actual segment exchanges data with
            // its neighboring elements
            total = 0.0;
            for (i = 0; i < 2; i++) {
                // Receive from neighboring elements
                // and put it in the variable 'received'
                total += received;
            }
            temperature += temperature + (1/h^2)*(total - 2*temperature);
        }
    }
    ...
}
```

Each time step, a new temperature for the local **Segment** object is obtained from the previous temperature and the temperatures of the neighboring segments (whether interior or extreme). Notice that the term “time step” implies an iterative method in which the operation requires four coefficients. The algorithm described takes into consideration an iterative solution of operations, known as *relaxation*. The simplest relaxation method is the Jacobi relaxation, in which the temperature of each and every interior segment is simultaneously approximated using its local temperature

---

and the temperatures of its neighbors (and it is the one presented here). Other relaxation methods include the Gauss-Seidel relaxation and the successive over-relaxation (SOR). Iterative methods tend to be more efficient than direct methods.

- *Information about parallel platform and programming language.* The parallel system available for this example is a SUN SPARC Enterprise T5120 Server. This is a multi-core, shared memory parallel hardware platform, with  $1 \times 8$ -Core UltraSPARC T2, 1.2 GHz processors (capable of running 64 threads), 32 Gbytes RAM, and Solaris 10 as operating system [17]. Applications for this parallel platform can be programmed using the Java programming language [5, 6].
- *Quantified requirements about performance and cost.* This application example has been developed in order to test the parallel system described in the previous point. The idea is to experiment with the platform, testing its functionality in time, and how it maps with a domain parallel application. So, the main objective is simply to test and characterize performance (in terms of execution time) regarding the number of processes/processors involved in solving a fixed size problem. Thus, it is important to retrieve information about the execution time considering several configurations, changing the number of processes on this parallel, shared memory platform.

### 3 Coordination Design

In this section, the *Architectural Patterns for Parallel Programming* [10] are used along with the the information from the Problem Analysis, in order to select an architectural pattern for developing a coordination that solves the One-dimensional Heat Equation.

#### 3.1 Specification of the System

- **The scope.** This section aims to describe the basic operation of the parallel software system, considering the information presented in the Problem Analysis step about the parallel system and its programming environment. Based on the problem description and algorithmic solution presented in the previous section, the procedure for selecting an architectural pattern for a parallel solution to the One-dimensional Heat Equation problem is presented as follows [10]:
  1. *Analyze the design problem and obtain its specification.* Analyzing the problem description and the algorithmic solution provided, it is noticeable that the calculation of the One-dimensional Heat Equation is a step-by-step, iterative process. Such a process is based on calculating the next temperature of each segment of the wire through



---

each time step. The calculation uses as input the previous temperature, and the temperatures of the two neighbor segments of the wire, and provides the temperature at the next time step.

2. *Select the category of parallelism.* Observing the form in which the algorithmic solution partitions the problem, it is clear that the wire is divided into segments, and computations should be executed simultaneously on different segments. Hence, the algorithmic solution description implies the category of **Domain Parallelism**.
3. *Select the category of the nature of the processing components.* Also, from the algorithmic description of the solution, it is clear that the temperature of each segment of the wire is obtained using exactly the same calculations. Thus, the nature of the processing components of a probable solution for the One-dimensional Heat Equation, using the algorithm proposed, is certainly a **Homogeneous** one.
4. *Compare the problem specification with the architectural pattern's Problem section.* An Architectural Pattern that directly copes with the categories of domain parallelism and the homogeneous nature [10] of processing components is the **Communicating Sequential Elements (CSE) pattern** [11]. In order to verify that this architectural pattern actually copes with the One-dimensional Heat Equation problem, let us compare the problem description with the Problem section of the CSE pattern. From the CSE pattern description, the problem is defined as [11]:

*“A parallel computation is required that can be performed as a set of operations on regular data. Results cannot be constrained to a one-way flow among processing stages, but each component executes its operations influenced by data values from its neighboring components. Because of this, components are expected to intermittently exchange data. Communications between components follow fixed and predictable paths”.*

Observing the algorithmic solution for the One-dimensional Heat Equation, it can be defined in terms of calculating the next temperature of the wire segments as ordered data. Each segment is operated almost autonomously. The exchange of data or communication should be between neighboring segments of the wire. So, the CSE is chosen as an adequate solution for the One-dimensional Heat Equation, and the architectural pattern selection is completed. The design of the parallel software system should continue, based on the Solution section of the CSE pattern.

- **Structure and dynamics.** Based on the information of the Communicating Sequential Elements architectural pattern, it is used here to describe the solution to the Heat Equation in terms of this architectural pattern's structure and behavior.

- 
1. *Structure.* Using the Communicating Sequential Elements architectural pattern for the One-dimensional Heat Equation, the same operation is applied simultaneously to obtain the next temperature values of each segment. However, this operation depends on the partial results in its neighboring segments. Hence, the structure of the actual solution involves a regular, one-dimensional, logical structure, conceived from the wire of the original problem. Therefore, the solution is presented as a one-dimensional network of segments that follows the shape of the wire. Identical components simultaneously exist and process during the execution time. An Object Diagram, representing the network of segments that follows the one-dimensional shape of the wire and its division into segments, is shown in Figure 5.

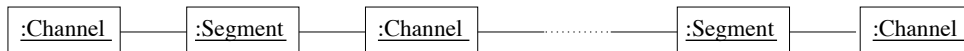


Figure 5: Object Diagram of Communicating Sequential Elements for the solution to the One-dimensional Heat Equation.

2. *Dynamics.* A scenario to describe the basic run-time behavior of the Communicating Sequential Elements pattern for solving the One-dimensional Heat Equation is shown as follows. Notice that all the segments, as basic processing software components, are active at the same time. Every segment performs the same temperature operation, as a piece of a processing network. However, for the one-dimensional case here, each segment object communicates with its previous and next neighbors as shown in Figure 6.

The processing and communicating scenario is as follows:

- Initially, consider only a single **Segment** object, **segment(i)**. At first, it exchanges its local temperature value with its neighbors **segment(i-1)** and **segment(i+1)** through the adequate communication **Channel** components. After this, **segment(i)** counts with the different temperatures from its neighbors.
  - The temperature operation is simultaneously started by the **segment(i)** component and all the other components of the wire.
  - In order to continue, all components iterate as many times as required, exchanging their partial temperature values through the available communication channels.
  - The process repeats until each component has finished iterating, and thus, finishing the whole One-dimensional Heat Equation computation.
3. *Functional description of components.* This section describes each processing and communicating software components as participants

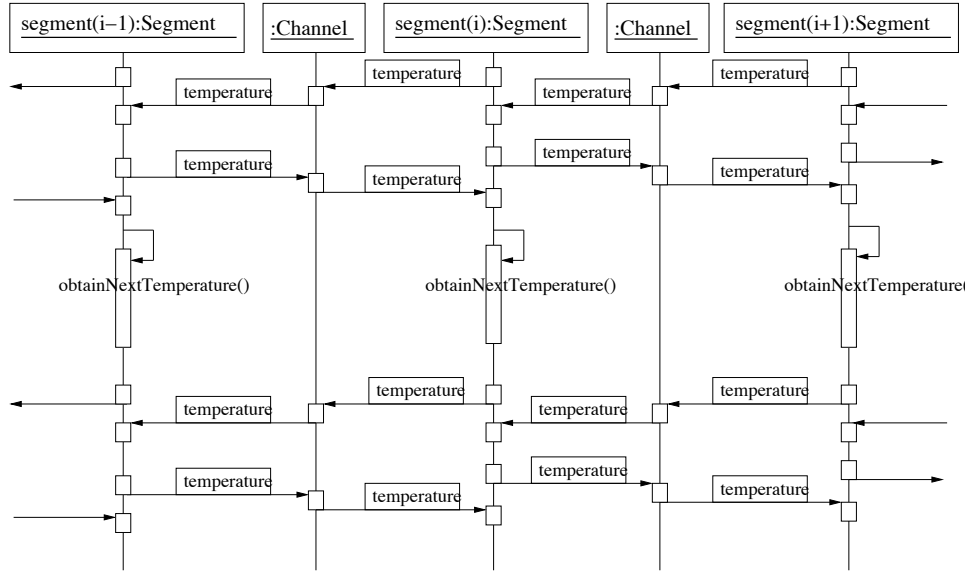


Figure 6: Sequence Diagram of the Communicating Sequential Elements for communicating temperatures through channel components for the One-dimensional Heat Equation.

of the Communicating Sequential Elements architectural pattern, establishing its responsibilities, input and output for solving the One-dimensional Heat Equation.

- **Segment.** The responsibilities of a segment, as a processing component, are to obtain the next temperature from the temperature values it receives, and make available its own temperature value so its neighboring components are able to proceed.
  - **Channel.** The responsibilities of every channel, as a communication component, are to allow sending and receiving temperature values, synchronizing the communication activity between neighboring sequential elements. Channel components are developed as the main design objective of a following step, called “Communication Design”, which is not addressed in this paper.
4. *Description of the coordination.* The Communicating Sequential Elements pattern describes a coordination in which multiple **Segment** objects act as concurrent processing software components, each one applying the same temperature operation, whereas **Channel** objects act as communication software component which allow exchanging temperature values between sequential components. No temperature values are directly shared among **Segment** objects, but each one may access only its own private temperature values. Every **Seg-**

---

**ment** object communicates by sending its temperature value from its local space to its neighboring **Segment** objects, and receiving in exchange their temperature values. This communication is normally asynchronous, considering the exchange of a single temperature value, in a one to one fashion. Therefore, the data representing the whole one-dimensional wire represents the regular logical structure in which data of the problem is arranged. The solution, in terms of a segmented wire, is presented as a network that actually reflects this logical structure in the most transparent and natural form [11].

5. *Coordination analysis.* The use of the Communicating Sequential Elements patterns as a base for organizing the coordination of a parallel software system for solving the One-dimensional Heat Equation has the following advantages and disadvantages:

– **Advantages**

- (a) The order and integrity of temperature results is granted because each **Segment** object accesses only its own local temperature value, and no other data is directly shared among components.
- (b) All **Segment** objects have the same structure and behavior, which normally can be modified or changed without excessive effort.
- (c) The solution is easily structured in a transparent and natural form as a one-dimensional array of components, reflecting the logical structure of the one-dimensional wire in the problem.
- (d) All **Segment** objects perform the same temperature operation, and thus, granularity is independent of functionality, depending only on the size and number of the elements in which the one-dimensional wire is divided. Changing the granularity is normally easy, by just adjusting the number of **Segment** objects in which the wire is divided, thus obtaining a better resolution or precision.
- (e) The Communication Sequential Elements pattern can be easily mapped into the shared memory structure of the parallel platform available.

– **Liabilities**

- (a) The performance of a parallel application for solving the One-dimensional Heat Equation based on the Communicating Sequential Elements pattern is heavily impacted by the communication strategy used. For the present example, the threads available in the parallel platform have to take care of a large number of **Segment** objects, so each thread has to operate on a subset of the data rather than on a single value. Due to this, dependencies between data, expressed as

---

communication exchanges, could be a cause of a slow down in the program execution.

- (b) For this example, load balancing is kept by allowing only a fixed number of **Segment** objects per thread, which tends to be larger than the number of threads available. Nevertheless, if data would not be easily divided into same-size subsets, then the computational intensity varies on different processors. Even though every processor is virtually equal to the others, maintaining the synchronization of the parallel application means that any thread that slows down should eventually catch up before the computation can proceed to the next step. This builds up as the computations proceeds, and could impacts strongly on the overall performance.
- (c) Using synchronous communications implies a significant amount of effort required to get a minimal increment in performance. On the other hand, if the communications are kept asynchronous, it is more likely that delays would be avoided. This is taken into consideration in the next step, “Communication Design” (not described here).

## 4 Implementation

In this section, all the software components described in the Coordination Design step are considered for their implementation using the Java programming language. Once programmed, the whole system is evaluated by executing it on the available hardware platform, measuring and observing its execution through time, and considering some variations regarding the granularity.

Here, it is only presented the implementation of the coordination structure, in which the processing components are introduced, implementing the actual computation that is to be executed in parallel. Further design work is required for developing the channel as communication and synchronization components. Nevertheless, this design and implementation goes beyond the actual purposes of the present paper.

The distinction between coordination and processing components is important, since it means that, with not a great effort, the coordination structure may be modified to deal with other problems whose algorithmic and data descriptions are similar to the One-dimensional Heat Equation, such as the Wave Equation or the Poisson Equation.

### 4.1 Coordination

Considering the existence of a class **Channel** for defining the communications between **Segment** objects, the Communicating Sequential Elements architectural

---

pattern is used here to implement the main Java class of the parallel software system that solves the One-dimensional Heat Equation. The class `Segment` is presented as follows. This class represents the Communicating Sequential Elements coordination for the One-dimensional Heat Equation example.

```

class Segment implements Runnable{
    private static int M = 65536, iterations = 10;
    private static Channel[] [] segment = null;
    private int i = -1;
    public Segment(int i){
        this.i = i;
        new Thread(this).start();
    }
    public void run(){
        double temperature, received, total;
        temperature = random(10*M);
        for (int iter = 0; iter < iterations; iter++) {
            // Send local temperature to neighbors
            if (i < M-2) send(segment[i+1][0], temperature);
            if (i > 1) send(segment[i-1][1], temperature);
            total = 0.0;
            // Receive temperature from neighbors
            if(i > 0 && i < M-1){
                received = receive(segment[i][0]);
                total += received;
                received = receive(segment[i][1]);
                total += received;
            }
            // Insert processing here
        }
    }
    public static void main(String[] args){
        segment = new Channel[M][2];
        for(int m = 0; m < M; m++){
            for(int i = 0; i < 2; i++){
                segment[m][i] = new Channel();
            }
        }
        for(int m = 0; m < M; m++){
            new Segment(m);
        }
        System.exit(0);
    }
}

```

This class only creates two adjacent, one-dimensional arrays of `Channel` components and `Segment` components, which represents the coordination structure of the whole parallel software system, developed for executing on the available parallel hardware platform. `Channel` components are used for exchanging temperature values between neighboring `Segment` components, each one first

---

sending its own temperature value (which is an asynchronous, non-blocking operation), and later retrieving the temperature values of the two neighboring wire components. Using this data, now it is possible to sequentially process to obtain the new temperature of the present component. This communication-processing activity repeats as many times as iterations defined.

The utility of the coordination presented here goes beyond the One-dimensional Heat Equation application. By modifying the sequential processing section, each wire component is capable of computing the discrete versions of other one-dimensional differential equations, such as the Wave Equation or the Poisson Equation.

## 4.2 Processing components

At this point, all what properly could be considered “parallel design and implementation” has finished: data is initialized (here, randomly, but it can be initialized with particular temperature values) and distributed among a collection of **Segment** components. It is now the moment to insert the sequential processing which corresponds to the algorithm and data description found in the Problem Analysis, This is done in the class **Segment**, where it is commented **Insert processing here**, by simply adding the following code, and considering the particular declarations for its computation:

```
temperature += temperature + (1/h^2)*(total - 2*temperature);
```

The simple, sequential Java code allows that each **Segment** component obtains a local temperature based on the One-dimensional Heat Equation. Modifying this code implies modifying the processing behavior of the whole parallel software system, so the class **Segment** can be used for other parallel applications, as long as they are one-dimensional and execute on a shared memory parallel computer.

## 5 Summary

The Architectural Patterns for Parallel Programming are applied here along with a method for selecting them, in order to show how to select an architectural pattern that copes with the requirements of order of data and algorithm present in the One-dimensional Heat Equation problem. The main objective of this paper is to demonstrate, with a particular example, the detailed design and implementation that may be guided by a selected architectural pattern. Moreover, the application of the Architectural Patterns for Parallel Programming and the method for selecting them is proposed to be used during the Coordination Design and Implementation for other similar problems that involve the calculation of differential equations for a one-dimensional problem, executing on a shared memory parallel platform.

## 6 Acknowledgements

The author wishes to thank Neil Harrison, my shepherd for Euro Plop 2009, for his encouraging comments about the present paper.

## References

- [1] G.R. Andrews *Foundation of Multithreaded, Parallel and Distributed Programming.*, Addison-Wesley Longman, Inc., 2000.
- [2] P. Brinch-Hansen *Distributed Processes: A Concurrent Programming Concept.*, Communications of the ACM, Vol.21, No. 11, 1978.
- [3] K.M. Chandy, and S. Taylor *An Introduction to Parallel Programming.* Jones and Bartlett Publishers, Inc., Boston, 1992.
- [4] E.W. Dijkstra *Co-operating Sequential Processes*, In Programming Languages (ed. Genuys), pp.43-112, Academic Press, 1968.
- [5] S. Hartley *Concurrent Programming. The Java Programming Language.*, Oxford University Press Inc., 1998.
- [6] Herlihy, M., and Shavit, N., *The Art of Multiprocessor Programming.* Morgan Kaufmann Publishers. Elsevier, 2008.
- [7] C.A.R. Hoare *Communicating Sequential Processes.* Communications of the ACM, Vol.21, No. 8, August 1978.
- [8] S. Kleiman, D. Shah, and B. Smaalders *Programming with Threads*, 3rd ed. SunSoft Press, 1996.
- [9] B. Lewis and D.J.. Berg *Multithreaded Programming with Java Technology*, Sun Microsystems, Inc., 2000.
- [10] J.L. Ortega-Arjona and G.R. Roberts *Architectural Patterns for Parallel Programming*, Proceedings of the 3rd European Conference on Pattern Languages of Programming and Computing (EuroPLoP98), Kloster Irsee, Germany, 1998.
- [11] J.L. Ortega-Arjona *The Communicating Sequential Elements Pattern. An Architectural Pattern for Domain Parallelism*, Proceedings of the 7th Conference on Pattern Languages of Programming (PLoP2000), Allerton Park, Illinois, USA, 2000.
- [12] J.L. Ortega-Arjona *The Shared Resource Pattern. An Activity Parallelism Architectural Pattern for Parallel Programming*, Proceedings of the 3rd European Conference on Pattern Languages of Programming and Computing (EuroPLoP98), Kloster Irsee, Germany, 1998.



- 
- [13] J.L. Ortega-Arjona *The Manager-Workers Pattern. An Activity Parallelism Architectural Pattern for Parallel Programming*, Proceedings of the 9th European Conference on Pattern Languages of Programming and Computing (EuroPLoP2004), Kloster Irsee, Germany, 2004.
- [14] J.L. Ortega-Arjona *The Parallel Pipes and Filters Pattern. A Functional Parallelism Architectural Pattern for Parallel Programming*, Proceedings of the 10th European Conference on Pattern Languages of Programming and Computing (EuroPLoP2005), Kloster Irsee, Germany, 2005.
- [15] J.L. Ortega-Arjona *The Parallel Layers Pattern. A Functional Parallelism Architectural Pattern for Parallel Programming*, Proceedings of the 6th Latin American Conference on Pattern Languages of Programming and Computing (SugarLoafPLoP2007), Porto de Galinhas, Pernambuco, Brasil, 2007.
- [16] J.L. Ortega-Arjona *Architectural Patterns for Parallel Programming: Models for Performance Evaluation*, PhD Thesis, Department of Computer Science, University College London, UK, 2007. <http://www.sigsoft.org/phdDissertations/theses/JorgeOrtega.pdf>
- [17] Sun Microsystems. *Sun SPARC Enterprise T5120 Server*. <http://www.sun.com/servers/coolthreads/t5120/>.

# Towards Formalized Adaptation Patterns for Adaptive Interactive Systems

Matthias Bezold

<sup>1</sup> University of Ulm, Institute for Information Technology, Ulm, Germany

<sup>2</sup> Elektrobit Automotive Software, Erlangen, Germany

`matthias.bezold@uni-ulm.de`

**Abstract.** A design pattern provides a general and proven solution for a recurring problem. Design patterns are an established approach in the domain of software engineering. Collections of such patterns also exist for graphical interfaces and adaptive hypertext. However, a collection of patterns for adaptive interactive systems does not exist. This paper presents such a collection to provide structured knowledge about applying adaptations to interactive systems. In addition, a formalization of these patterns using semantic technologies is presented as well as the application of these formalizations in an adaptation framework.

## 1 Introduction

Adaptive interactive systems describe user interfaces that change based on the user-system interaction to better reflect the requirements of an individual user. Adaptation has been recognized as a means for improving the usability of user interfaces and studied accordingly [13]. For instance, one possible adaptation highlights frequently used values in a list or another one emphasizes interface elements that might be of increased interest to the user. However, there is no structured work so far that lists and categorizes different kinds of adaptations for adaptive interactive systems. At the same time, tool support and frameworks facilitate a wide-spread use of adaptations in interactive systems.

This paper proposes an approach for adapting multimodal interactive systems, such as automotive dashboard systems, personal navigation devices, or home entertainment systems, which can also be speech-enabled. Two contributions are presented. First, an adaptation architecture employs an abstract definition of adaptation patterns to define adaptations for interactive systems. These definitions can be reused between different systems, but at the same time adjusted to the requirements of specific systems. The approach is based on a semantic description of the interactive system and the adaptations are integrated into a model-based development process. Second, we describe a set of adaptation patterns for interactive adaptive systems. These patterns define successful adaptations that have been used in different systems.

This paper is organized as follows. After a review of related work in Section 2, the application of formalized adaptation patterns in the development of

interactive systems in demonstrated in Section 3. Section 4 introduces a set of adaptation patterns that can be used with the presented framework. After an overview of a prototype implementation and a use case in Section 5, Section 6 concludes this paper.

## 2 Related Work

This section introduces the concept of patterns, reviews patterns from related work, e.g. interface patterns, and presents approaches for the formalization of design patterns. Design patterns are an established method in software engineering that collects solutions for recurring problems. A design pattern consists of a proven solution and a discussion of a problem it solves. Moreover, the context of the pattern further refines the circumstances under which this pattern is applicable. One problem can be addressed by different patterns and the context determines which pattern is used best.

The most well-known use of design patterns are the software design patterns by Gamma et al. [7]. These patterns are widely adopted and taught in classes. Similarly, Buschmann et al. [4] present a set of patterns for software architecture, which deal with a more high-level view on software design.

Originally, patterns only existed in textual, narrative form. However, research on the formalization of patterns aims to increase their utility. A formalized description is a representation using a well-defined structure and vocabulary, thus providing a standardized and machine-processable representation. Different levels of formalization exist for patterns. The first level is to write down the narrative pattern descriptions in a formalized pattern format, which enforces special markings to label the different sections (e.g. motivation or solution) of a pattern description. This ensures consistency and allows referencing between different pattern collections by providing a machine-readable structure in which the patterns are filled in. The Pattern Language Markup Language (PLML) [6] follows this approach and provides an XML document type definition (DTD) for specifying patterns. PLML is however a very high-level definition that aims at describing pattern collections in a uniform way and the semantics of the patterns are not formalized.

A more formal notation of patterns can serve as a basis for intelligent tool support, for instance by providing support when refactoring existing projects to patterns (e.g. Zannier and Maurer [20]). Other approaches even formalize the semantics of the adaptation patterns. Mikkonen [16] presents an approach for formalizing patterns based on a custom notation for defining objects formally, with the focus being on the temporal behavior of design patterns. Hallstrom and Soundarajan [9] present an approach that enables validation and reasoning with patterns. Another example is the work by Henninger [11], who proposes a meta-model for software patterns based on an Web Ontology Language (OWL) infrastructure for applying the patterns in the software development process. This model conceptually builds on PLML, but extends it considerably by including a more formalized representation of the patterns using description logic

to add further semantic knowledge about the patterns. Henninger presents interface patterns as an example of this approach. Our approach provides tool support for adaptation patterns and includes a semantic description of parts of the patterns, but does not fully formalize the precondition for patterns. Therefore, our approach is located between simple tool support and fully formal pattern systems.

Whereas design patterns are mostly used in the domain of software engineering, patterns were also applied by different researchers to the domain of user interface design. For instance, van Welie and van der Veer [19] and Borchers [3] have compiled extensive pattern collections of reusable interface design knowledge, which can be used by designers and system developers in creating graphical interfaces. Tidwell [18] presents an extensive structured catalog of interface design patterns, which covers a wide range of topics, such as the general structure of a graphical application, form input, and aesthetics. However, these patterns do not cover adaptive user interfaces.

A basic set of abstract adaptation patterns, which are descriptions of proven adaptations, was presented for adaptive hypertext systems by Danculovic et al. [5], introducing Link Personalization, Content Personalization, Structure Personalization, and Remote Personalization, which all are very general. These patterns were extended by Koch and Rossi [14] by adding more detailed patterns such as Adaptive Anchor Selection or Adaptive Sorting of Anchors. However, the adaptation of hypertext is focused on content and the linking of different documents rather than the user interface, as required for adaptive (graphical) interfaces. Moreover, more information can be extracted from the user-system interaction of an interactive system, since the observation by an interactive system is richer than tracking the list of visited pages in hypertext systems. Therefore, adaptive interactive systems have their own adaptation patterns and defining such patterns can aid developers in deciding under which circumstances to apply which adaptations to improve an interactive system.

### 3 Formalization and Execution of Adaptation Patterns

This section describes how a description logic-based, semantic definition of adaptation patterns is used to enable adaptivity in interactive systems. In order to apply these patterns to an interactive system, we introduce a semantic model of the interactive system.

Whereas the domain of the design patterns by Gamma et al. [7] is source code, the domain of these adaptation patterns are interactive systems. Therefore, the abstract description of interactive systems used in this work corresponds to an abstract description of source code that is required by tool support for design patterns. For this purpose, the interactive system is described by a semantic layer. This layer is based on an ontology defined in the Web Ontology Language (OWL) [17] format and consists of a number of classes and individuals of these classes. For instance, each graphical element or speech output prompt in the interactive system is represented by an instance of a “button” or “prompt” class

and further described by a set of properties, such as color and size in case of the button. The semantic layer is derived automatically from a description of the interactive system, but annotation contributes further information. The advantage of this approach is that all parts of the interactive system, e.g. the interactive system, information about the user, and a description of the adaptations, are represented using the same formalism. In addition, OWL allows for an automatic inference of additional information (cf. Horrocks et al. [12]).

An adaptation pattern is a description of a common problem of interactive systems and a solution of this problem that is based on adapting the interactive system to the behavior of a user. The formalized definition of adaptation patterns in this work consists of two parts: a declarative description that can be reused between different systems and a functional description of the adaptations that defines how an adaptation is performed in a certain system. The formalization facilitates an inclusion in the tool chain, thus enabling a tool-supported development of adaptive interfaces.

### 3.1 Declarative Description of Adaptation Patterns

The declarative description of patterns consists of three parts: a trigger of the pattern, a selection that determines which part of the interface the adaptation should be applied to, and the name of an abstract adaptation. The declarative description is called *adaptation selector* in the adaptation framework, because it selects an interface element and an adaptation. Adaptation selectors can be reused between different systems, since they only rely on abstracted information in the semantic layer to define the declarative part of the adaptation. For instance, graphical elements are addressed as “graphical button” or “list”, which can support a variety of different implementation and flavors of actual interface elements. The declarative description of an adaptation pattern consists of a number of adaptation selectors.

The adaptation trigger is connected to the user modeling component [2], which observes the user and derives information from the user-system interface. For instance, the user modeling component can predict future user actions (e.g. opening a certain sub-menu) or the user’s favorite values (e.g. favorite names in an address book). When the user modeling component provides new information, the respective adaptation selectors, which are triggered by this information, are activated.

Interface elements are selected by an adaptation executor through a query that returns all elements from the semantic layer that match the given query. For instance, if the user modeling predicts that a certain action called “A” will be performed next by the user, the selection could load all elements from the semantic layer that trigger action “A”. A simplified version of SPARQL<sup>3</sup> is used for the notation for the queries.

---

<sup>3</sup> SPARQL Protocol and RDF Query Language: <http://www.w3.org/TR/rdf-sparql-query/>

<b>Adaptation selector “ButtonEmphasisSelector”</b>	
Trigger:	Prediction of user action “A” by the user modeling component
Element:	Select a graphical button that trigger the predicted action “A” (in an OWL-based notation)
Adaptation:	ButtonEmphasis

**Fig. 1.** An adaptation selector selects all graphical buttons that trigger an action that was predicted. In addition, an abstract adaptation (“Component Emphasis”) is chosen for the selected elements.

Finally, an abstract adaptation is recommended for the selected elements. For example, the selected button that triggers action “A” could be emphasised to draw the user’s attention to it by enabling the “ButtonEmphasis” adaptation, which is an instance of the “Component Emphasis” pattern (to be introduced in Section 4).

Fig. 1 shows an adaptation selector that was used as an example throughout this section: Based on a prediction of a user action “A” by the user modeling component, the adaptation selector picks all graphical buttons from the semantic layer that trigger action “A” and recommends the “Component Emphasis” adaptation for these elements.

### 3.2 Functional Description of Adaptation Patterns

In order to enable the interactive system to apply an abstract adaptation recommended by an adaptation selector, a functional description of the pattern is required, i.e., instructions on how to execute this pattern on a specific interface element. Adaptation executors can be system-specific and therefore not be reused between different systems in every case. The reason is that the execution of adaptations depends on the specific implementation of the interface element. However, default implementations have been defined that can be used on a wide range of elements by changing only basic properties (such as the position or the size).

The functional definition of patterns specifies the effects of an adaptations by defining which properties of the individuals are changed. Fig.2 shows an adaptation executor of the “Component Emphasis” pattern that changes the color or the size of a graphical button, which are represented by properties of the corresponding individual. One adaptation pattern is not represented by a single adaptation executor, but by a set of adaptation executors for different interface elements, since the same adaptation manifests itself very differently for different elements. For instance, the Emphasis pattern has a different functional description for a graphical button than it has for a graphical list.

```

Adaptation executor "ButtonEmphasisExecutor"
Adaptation:      ButtonEmphasis
Interface element: Graphical button
Property changes:
x = x - 5
y = y - 5
width = width + 10
height = height + 10
textColor = color.yellow

```

**Fig. 2.** Example of an adaptation executor, which perform the Emphasis adaptation on a button.

### 3.3 Application of the Adaptation Patterns in an Interactive System

The previous section discusses how adaptations are defined, but not how the adaptive system decides whether to execute a suggested adaptation. This section introduces two approaches for defining which adaptations to execute: selection by the system designer and an automatic procedure in which the adaptation component decides automatically which adaptations to execute.

**Specification by the System Designer** An integration of formalized adaptation patterns into the model-based development process thus facilitates the development of adaptive interactive systems. At design time, the formalized adaptation patterns provide tool support to the designer of the adaptive system. For this purpose, the system designer can decide when the individual adaptations should be executed by enabling or disabling adaptations for certain parts of the system.

Adaptations can be enabled on three levels: globally, for an interface context, or for an individual element. First, if an adaptation is enabled globally, it is executed whenever it is recommended by an adaptation selector. Second, adaptations can be enabled for interface contexts. An interface context is for example a graphical screen or a speech component, which is a set of speech output prompts and speech input grammars that are enabled together. When an adaptation is enabled for an interface context, it is executed in the respective context, but not in others where the adaptation was not enabled. Third, adaptations can be enabled on a per-element basis.

In addition, conditions can be added these definitions. For instance, an adaptation should only be executed when the user is a beginner, based on information stored in the user model or the semantic layer. Therefore, the system designer can decide in a very flexible way which adaptations should be executed, without the need of defining the adaptations manually.

**Automatic Execution by the Adaptation Component** In addition to the developer deciding about which adaptation to apply, an adaptation component

can decide automatically at runtime about the execution of adaptations. This decision is based on the semantic representation of both the interactive system and the adaptations. The overall procedure is similar to the specification of adaptations, but decisions taken at design time are instead performed by the adaptation component at runtime.

The adaptation component can automatically decide about the level of adaptivity, based on the proficiency of the user in general or with respect to individual parts of the system. For example, adaptive help can be used for parts of the system that the user has not used extensively, while no adaptations are used in parts that the user knows well.

## 4 Adaptation Patterns

This section presents a set of adaptation patterns for interactive systems. These adaptation patterns change the user interface of the system, but do not directly depend on the application logic. Certain adaptations are outside of the scope of this paper, because they do not address general problems, but specific algorithms, such as for instance an adaptation of the route generation algorithm of a navigation device by the Adaptive Route Adviser [15]. Instead, this paper deals with general adaptations for user interfaces of interactive systems. Since multi-modal adaptive systems are the subject of the adaptations, their applicability to speech-based interfaces is also considered.

The patterns presented in this section are more general in their nature than patterns in other pattern collections, which make them applicable to a broad range of interface components. Unobtrusiveness is a main principle of these adaptations to comply with usability principles such as consistency and learnability. Adaptations that reconstruct the whole interface thus interfere with such usability principles.

Adaptations are executed based on an observation of the user-system interaction. This observation of user behavior, called user modeling, creates a representation of the user that serves as a basis for decisions about adaptations. For this purpose, a user modeling component observes the user (e.g. from log data) and constructs a model using different algorithms, such as the ones presented by Zukerman and Albrecht [21], thus providing information about user actions and preferences. The user modeling phase, which is therefore a crucial part of adaptive interfaces, is not explicitly part of these pattern descriptions. However, the “Adaptation Trigger” section of the patterns describes which observations of the user modeling component trigger the respective adaptation.

Adaptation selectors and a set of generic adaptation executors, as introduced in Section 3, were defined for the patterns presented in this section. In order to make an adaptation fit into a specific system seamlessly, custom adaptation executors can be defined in addition to the existing ones.



## 4.1 Component Emphasis

**Intent** Guide the user by emphasizing certain elements of the interface. Limit the changes to the part of the interface that requires emphasis. In doing so, enable users to reuse their acquired knowledge of the interactive system and avoid distracting the user through fundamental changes of the interface.

**Motivation** During the interaction with an interactive system, a user has a goal and is looking for interface elements that can help in fulfilling it. For instance, the user might look for a graphical button triggering an action. The system provides support by guiding the user to the respective interface elements.

### Forces

- The user follows a certain goal when using the system and might spend considerable time looking for interface elements that facilitate reaching this goal.
- Performing major changes to the system can confuse the user and distract from the current task. Subtle guidance instead supports the user.
- Emphasizing wrong elements can impede the user, therefore a sufficiently good user modeling prediction is crucial for this adaptation.
- The adaptive emphasis should be conceived in a way that the user does not confuse it with a regular selection in the user interface.

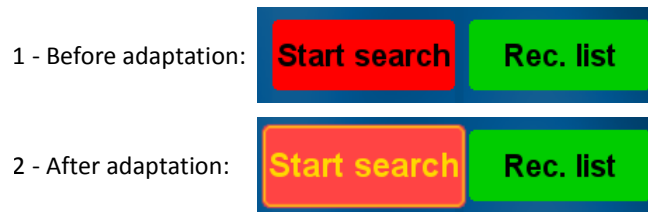
**Solution** Make the adaptive system change properties of interface elements in a way that they draw the user's attention. Use assumptions of a user modeling component, such as a prediction of the most likely next action or an action the user has not used yet. Help the user reach the current goal by emphasizing interface elements that are related to the respective assumption of the user modeling component.

**Adaptation Trigger** The following observations of the user modeling component trigger the Component Emphasis adaptation:

- Prediction of the next user action.
- Actions that the user has not used yet, but which others have used.

**Related Patterns** The “List Item Emphasis” pattern emphasizes elements in a list and is therefore related to this pattern, which emphasizes arbitrary elements related to triggering actions.

The “Prominent ‘done’ button” pattern [18] statically emphasizes a button that finishes a task associated with a graphical view, but the emphasis is not performed based on the current user's behavior.



**Fig. 3.** Emphasis of a button in an interactive TV system. The “Start search” button is emphasized compared to the non-emphasized “Rec. list” button. The reasoning is to provide non-intrusive and subtle hints, in this case by increasing the size of the button and changing the text color.

**Example** Consider an electronic program guide, where the user specifies filter criteria, such as channel or time, to filter the list of TV shows. After a number of criteria was selected, the user has to press a “Show results” button to see all shows that match the selected criteria. Increasing the size of the button and changing colors (see Fig. 3) emphasizes the button, thus supporting the user in finishing the current task.

The Emphasis pattern is also applicable to voice interfaces. If a user enters a state where the system reads the possible utterances, saying a phrase as the first or the last one draws a user’s attention to this phrase.

#### 4.2 List Element Selection

**Intent** Support the user in selecting similar-looking elements from a list, for instance by highlighting frequently used entries from the list.

**Motivation** When selecting elements from a list, users often select some elements frequently and others not at all. The selection process can be improved by emphasizing frequently selected elements from the list.

#### Forces

- Selecting frequently used items in a list should take less time for the user than selecting others.
- If a list is longer than one screen, highlight the interesting items also in the scrollbar to enable the user to quickly scroll to the interesting elements.
- Emphasized list elements should be highlighted in a way that the user does not confuse them with elements the cursor is placed on.
- Emphasizing wrong elements can impede the user, therefore a sufficiently good user modeling prediction is crucial for this adaptation.



**Fig. 4.** Three elements are emphasized in a selection list by the List Item Emphasis pattern. The emphasized elements are supposed to be selected more frequently by the user than others.

**Solution** Emphasize these elements in the list that have been selected more often before than others. In doing so, let the user more quickly see these elements which are of increased interest. For instance, change the text or background color of these elements or add markers to differentiate interesting elements from others.

**Adaptation Trigger** The following observations of the user modeling component trigger the List Item Emphasis adaptation:

- List entries that have been selected more often than others either by the current user or by other users.
- Elements in a list that the user has not yet selected, but which should be interesting based on the user’s previous behavior.

**Related Patterns** The “Element Emphasis” adaptation pattern also emphasizes interface elements, but these elements are not necessarily similar, as are list elements, and are mostly used for navigating within the system.

The “Annotated scrollbar” pattern [18] recommends adding information to the scrollbar, which is also proposed by this pattern to mark the position of recommended elements in the list. Moreover, the “Adaptive Anchor Annotation” [14] pattern describes how to annotate links in a hypertext system, and this pattern can be considered as a sub-set of the anchor annotation pattern by annotating emphasis.

**Example** Selecting elements from a list is a very common action when interacting with interactive systems. For instance, selecting a name from the address book is one of the fundamental functions of interactive systems that support phone calls, such as mobile phones or automotive dashboard systems. Since users call a small number of people from their phone book frequently, the selection of these names from the address book can be quickened by highlighting these

names. An example of such a system is given in Fig. 4, which shows an address book that emphasizes the three most frequently selected elements.

Different visualizations of the “List Item Emphasis” pattern are possible and have been examined by research projects. One example are fisheye menus [1] that assign a different font size to different elements; this kind of visualization can be employed for adaptations as well.

### 4.3 Alternative Elements

**Intent** Provide a set of configurations for different interface components or screens and select the appropriate configuration to better support the requirements of an individual user.

**Motivation** Since the demands as well as the skills of users of interactive systems vary, different system configurations can better reflect the needs of an individual user. Instead of providing one configuration that tries to consider all possible users, the adaptation selects the version which is best suited for the needs of the current user. A user modeling component provides information about the proficiency of the user, which it derives for instance from the interaction speed and the number of user errors.

#### Forces

- Different configurations of interface components or graphical screens better reflect the needs of individual users.
- Automatically generated alternatives can break with existing usability principles.
- Additional time has to be spent developing the different alternatives, but the user can benefit from an improved user-system interaction.

**Solution** Provide different versions of a certain part or component of the interactive system to the adaptation component, for instance of a graphical screen, a speech output prompt, or a property (e.g. font size). Support users by selecting the appropriate alternative for the respective entity. Use information from the user modeling component at runtime to derive the most suitable configuration for the current user. By providing a set of alternatives to the adaptation component, which were created by the system designer, it is ensured that the interactive system adheres to design principles.

**Adaptation Trigger** The following observations of the user modeling component trigger the Alternative Elements adaptation:

- Preferences or properties of the user, such as the knowledge level or experience of the user.

**Related Patterns** The “Alternative views” pattern [18] lets the user decide among alternative views, for instance of a web page. However, the most appropriate view is not selected automatically.

**Example** The Alternative Elements pattern can be employed at different levels. For instance, when a user has to enter different values in an input screen, such as selecting the destination in a navigation device or selecting criteria in an interactive TV program guide, a simple version of the screen is provided to novice users and a more powerful version to advanced users. On a lower level, a larger font size improves the readability for visually impaired users.

On the other hand, a speech interface can provide different levels of speech output prompts. Novice users receive extended prompts when they enter a new part of the system. These prompts explain the most important functions to them. Intermediate users only require shorter prompts, which list the commands, but do not necessarily explain them. Finally, expert users, who could be annoyed by long and repetitive speech output, only hear a short prompt explaining the current state of the system and receive more explanation on request. A system that employ this kind of adaptation is for example presented by Hassel and Hagen [10].

#### 4.4 Adaptive Help Presentation

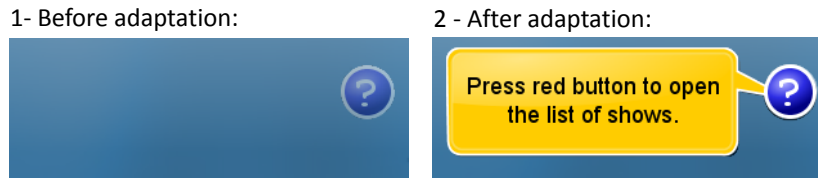
**Intent** Present adaptive help for the current situation of the user.

**Motivation** Help in interactive system is often static or only considers the currently active screen, but different people are likely to have different problems in different contexts. Providing help to the user is more valuable if it covers the current task of the user. By not only taking into account the current context, i.e., the graphical screen or speech state, but also the user’s interaction history, help is more specific and can thus support a user more precisely in the current task.

##### **Forces**

- Help tailored to the current task of the user is more valuable than static help.
- Static help can be too advanced for beginners and at the same time too superficial for expert users.
- Providing help can be assistive for beginners, but annoying for expert users.

**Solution** Provide specific help for the current situation of the user. Observe the user-system interaction to determine the situation and the context of the user. Present the help either on a separate area of the screen, or use an icon (or an acoustical “earcon”) to indicate the availability of help. Give the user an option



**Fig. 5.** Adaptive help supports the user by showing a text message that fits the current situation of the user. If a user is sufficiently proficient in working with the system, help messages are no longer shown.

to open this help once it is available. But ensure at the same time that the user is not distracted by the provided help. Therefore, avoid messages that fully engage the user’s attention, as for instance modal help messages. Provide an acoustic signal instead of a graphical hint for speech interfaces or visually impaired users.

**Adaptation Trigger** The following observations of the user modeling component trigger the Adaptive Help Presentation adaptation:

- Prediction of the next user action.
- Detection of user problems.
- Preferences or properties of the user, such as the knowledge level or experience of the user.

**Related Patterns** The “Multi-level help” pattern [18] suggests using different help techniques. Adaptive help is one kind of help that provides information adjusted to the current situation of the user.

**Example** In an interactive TV system, the user can browse the TV program in an electronic program guide and for this purpose specify different filter criteria, such as channel or time. Help is presented to the user by fading in a yellow message box on the top of the screen. When the user enters the selection screen for the first time, the help explains how to select filter criteria. After some criteria were selected, the help text on the screen tells the user to open the result screen next. Fig. 5 gives an example of the adaptive help feature in a digital TV system.

#### 4.5 Shortcut Area

**Intent** Present shortcuts for executing actions or selecting values to the user on a separate part of the interface. In doing so, accelerate the execution of frequent actions or sequences of actions and selection of the user’s favorite values.

**Motivation** Users often select the same values repeatedly, for instance when selecting elements from a list, such as a list of fonts, or by executing the same actions over and over again. The interaction is simplified by presenting these items to the user as shortcuts. By employing a special area for the shortcuts, the decision whether to use shortcuts is left to the user.

### **Forces**

- Finding frequently used elements and executing actions repeatedly can be very time-consuming for the user. Shortcuts can therefore simplify the user-system interaction.
- Shortcuts that automatically pop up on top of the interface interfere with the user interface and distract the user. A separate area that is always visible instead allows the user to decide whether or not to use shortcuts and limits the distraction of the user.

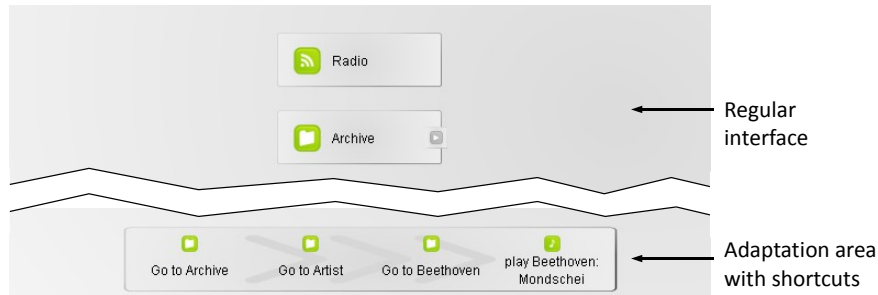
**Solution** Employ a separate area of the screen – called shortcut area – to present shortcuts to the user, thus avoiding a distraction of the user. Make this list either part of one interface element (e.g. of a list) or make it a separate part of the whole screen for presenting global shortcuts. In doing so, enable the user to find frequently used elements more quickly by selecting them from a distinct area of the screen. Use the output of a user modeling component to create the list of shortcuts.

**Adaptation Trigger** The following observations of the user modeling component trigger the Shortcut Area adaptation:

- Prediction of the next user action or a sequence of user actions.
- Prediction of a user preference, such as a TV channel.

**Related Patterns** The “Streamlined repetition” pattern [18] suggests considering repeated operations when creating an interface. The Shortcut Area provides a solution for this recommendation. The “Action panel” pattern [18] presents a list of available actions to the user, which is similar to an Shortcut Area that contains user actions. If the Shortcut Area presents a sequence instead of single items, the adaptation is similar to the “Autocomplete” pattern [18], since the adaptation anticipates user behavior.

**Example** In a selection list, a separate area on the top of the list presents the most frequently selected entries of the list to the user. By selecting them, the user does not have to scroll through the whole list. One example of such a list is the font selection list in Microsoft Word (2000 and later), which shows the most recently used fonts in a separate area on the top of the list.



**Fig. 6.** A separate adaptation area presents a list of buttons based on a prediction of the user’s next actions. These actions are executed by pressing the respective buttons.

A different application of the Shortcut Area pattern is to provide navigation shortcuts. A user modeling component recognizes user actions and predicts a sequence of possible next actions, with each action being represented by a button in an adaptation area. If the user presses one of these buttons, the action associated with the button and all actions before the pressed button are executed, thus reducing the number of required interactions. An example of an interface that provides navigation shortcuts to the user is shown in Fig. 6: In addition to the regular interface (shown on top), the interactive system presents a list of likely next actions to the user on the bottom of the screen. If the user presses one of these buttons, the interactive system automatically executes the respective actions.

## 5 Implementation and Sample Use Case

This section gives an overview of an implementation of the adaptation approach presented in Section 3, which includes a definition of the adaptation patterns introduced in Section 4. In addition, a sample use case further illustrates the use of this approach.

### 5.1 Implementation

In order to investigate the approach presented in this paper, an adaptation framework was developed within a model-based development environment called EB GUIDE Studio [8]. The tool comprises a simulation component that executes a specified application. A semantic layer was added that uses an OWL-based ontology to describe the system, but also the user-system interaction. The semantic layer is implemented using the Jena framework<sup>4</sup> and is available both at design time and at runtime.

The framework comprises an adaptation component, which works as discussed in Section 3. Moreover, it includes a set of adaptation selectors for the

<sup>4</sup> Jena Semantic Web Framework: <http://jena.sourceforge.net/>



adaptations presented in Sect. 4. Since these selectors are defined on the abstract level of the semantic layer, they can be reused between different interactive systems. In addition, the framework includes a set of default adaptation executors that only change properties common to all graphical elements. In doing so, these can also be used for different systems, but a better integration into the specific style of a certain system can be achieved by defining custom executors for a specific system.

The prototype implementation includes a user modeling component that models user behavior from low-level events, such as key presses by the user and system reactions to these inputs. Based on these low-level events, information such as the most likely next interaction step or a user preference, such as a favorite TV channel or font, are computed and forwarded to the adaptation component and makes these user modeling events available as triggers.

To show the feasibility of this approach, adaptations were implemented in two different interactive systems. First, an adaptive version of a interactive TV prototype application was created. Among the implemented adaptations are Adaptive Help Presentation, which guides novice user through the system, and Component Emphasis, which emphasizes buttons the user is most likely to use next. The latter adaptation is discussed in detail as a use case in Section 5.2. Second, an adaptive version of a system resembling an automotive dashboard system was created. It includes the Alternative Elements adaptation, which in this system selects between a beginner and an expert version of a route guidance entry screen, and the List Item Emphasis adaptation, which highlights frequently selected names in an address book.

## 5.2 Sample Use Case

This section presents a use case of applying an adaptation pattern to an interactive system. In addition to the application of the adaptation pattern, the preparation of the user modeling component is also discussed. As an example, the Component Emphasis adaptation is applied to a TV system, which includes an electronic program guide (EPG). After selecting a set of filter values, such as channel, time, or genre, the user can open the results screen by pressing a button labeled “Show Results”. Since novice users could not be aware that the “Show Results” button has to be pressed, emphasizing it can help the user in browsing the EPG.

The user modeling component has to be set up in a way that it observes the user’s behavior and predicts user actions. In this framework, an algorithm based on Markov chains is used for action prediction and an adaptation trigger is emitted when such a prediction was made.

Most of the semantic layer is generated automatically from the model-based description in the development tool. However, some manual annotation was still required. In this case, the information that the “Show Results” button triggers the “OpenResultList” action is annotated to the button and loaded into the semantic layer.

Finally, a custom adaptation executor could be defined for this interactive system. The behavior of the default generic adaptation executor, which simply increases the size of the interface element, might not fit in well with the design of the system, and therefore, a custom adaptation executor, which increases the size through an animation and make the background color slightly lighter, was defined.

The execution of the adaptation then works as follows. First, the user modeling component emits an event that a user action was predicted and thus triggers “ButtonEmphasisSelector” selector, which is part of the framework and selects a button that triggers the predicted user action. This selector then recommends the abstract “Component Emphasis” adaptation for execution and the adaptation component tries to find an appropriate executor. Two executors are available: the default executor, which is part of the framework, and the custom executor defined for this system. In this case, the adaptation component gives priority to the custom executor and executes the custom adaptation executor accordingly.

## 6 Conclusion

In this paper, we presented a definition of adaptation patterns that can be integrated into the model-based development process. A set of general adaptation patterns for adaptive interactive system was defined. A sample implementation of the presented framework was presented to show the feasibility of this approach. The adaptation patterns defined in this paper were integrated into the framework and the adaptations were applied to two sample systems. A detailed use case further illustrated the approach.

## Acknowledgements

The author is indebted to Paul Adamczyk, the shepherd of this paper for EuroPLoP 2009, for his time, comments, and observations, which allowed the paper to evolve during the shepherding process. Many thanks also for the insightful and encouraging comments from the discussion group at EuroPLoP 2009.

Copyright retained by author. Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.

## References

1. B. Bederson. Fisheye Menus. In *ACM Conference on User Interface Software and Technology (UIST)*, pages 217–226. ACM Press, Suracuse, NJ, USA.
2. M. Bezold. User Modeling from Basic Events in Interactive Systems for Intelligent Environments. In *International Conference on Intelligent Environments (IE)*, pages 319–326, 2009.
3. J. Borchers. *A Pattern Approach to Interaction Design*. Wiley, Chichester, UK, 2001.

4. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented Software Architecture*. Wiley, Chichester, UK, 1996.
5. J. Danculovic, G. Rossi, D. Schwabe, and L. Miaton. Patterns for Personalized Web Applications. In *European Conference on Pattern Languages of Programs (EuroPLoP) 2001*, pages 423–436, 2001.
6. S. Fincher, J. Finlay, S. Greene, L. Jones, P. Matchen, J. Thomas, and P. J. Molina. Perspectives on HCI patterns: Concepts and Tools. In *Extended Abstracts on Human Factors in Computing Systems (CHI)*, pages 1044–1045. ACM, 2003.
7. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Upper Saddle River, NJ, USA, 1995.
8. S. Goronzy, R. Mochales, and N. Beringer. Developing Speech Dialogs for Multimodal HMIs Using Finite State Machines. In *9th International Conference on Spoken Language Processing (Interspeech), CD-ROM*, 2006.
9. J. O. Hallstrom and N. Soundarajan. Formalizing Design Patterns: A Comprehensive Contract for Composite. In *Proceedings of the 7th FSE Workshop on the Specification and Verification of Component-Based Systems*, pages 77–82, 2008.
10. L. Hassel and E. Hagen. Adaptation of an automotive dialogue system to users expertise and evaluation of the system. *Computers and the Humanities*, 40(1):67–85, 2006.
11. S. Henninger and P. Ashokkumar. An Ontology-Based Metamodel for Software Patterns. In K. Zhang, G. Spanoudakis, and G. Visaggio, editors, *Conference on Software Engineering & Knowledge Engineering (SEKE) 2006*, pages 327–330, 2006.
12. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical report, World Wide Web Consortium, May 2004.
13. A. Jameson. *Human-computer Interaction Handbook*, chapter Adaptive Interfaces and Agents, pages 305–330. Erlbaum, Mahwah, NJ, USA, first edition, 2003.
14. N. Koch and G. Rossi. Patterns for Adaptive Web Applications. In *European Conference on Pattern Languages of Programs (EuroPLoP) 2002*, pages 179–194. Universitätsverlag Konstanz, 2002.
15. P. Langley. User Modeling in Adaptive Interfaces. In *Conference on User Modeling (UM) 1999*, pages 357–370. Springer-Verlag, New York, NY, USA, 1999.
16. T. Mikkonen. Formalizing Design Patterns. In *Proceedings of the 20th international conference on Software engineering (ICSE)*, pages 115–124, Washington, DC, USA, 1998. IEEE Computer Society.
17. M. K. Smith, C. Welty, and D. L. McGuinness. OWL Web Ontology Language Guide. Technical report, W3C, 2004.
18. J. Tidwell. *Designing Interfaces*. O’Reilly Media, Sebastopol, CA, USA, first edition, 2005.
19. M. van Welie and G. C. van der Veer. Pattern Languages in Interaction Design: Structure and Organization. In *Interact 2003*. IOS Press, Amsterdam, The Netherlands, 2003.
20. C. Zannier and F. Maurer. Tool Support for Complex Refactoring to Design Patterns. In M. Marchesi and G. Succi, editors, *Extreme Programming and Agile Processes in Software Engineering (XP)*, volume 2675 of *Lecture Notes in Computer Science*, pages 123–130. Springer, Heidelberg, Germany, 2003.
21. I. Zukerman and D. W. Albrecht. Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2):5–18, 2001.

# A Pattern-based Approach Against Architectural Knowledge Vaporization

Uwe van Heesch  
University of Groningen,  
Fontys University of Applied Sciences  
Venlo, The Netherlands  
u.vanheesch@fontys.nl

Paris Avgeriou  
University of Groningen  
Groningen, The Netherlands  
paris@cs.rug.nl

## Abstract

*Architectural documentation is often considered as a tedious and resource intensive task, that is usually skipped or performed inadequately. As a result the rationale of the architect's decisions gets lost. This problem is known as architectural knowledge vaporization. We propose a documentation approach for architectural decisions concerning the application of software patterns. Based on the assumption that patterns and pattern languages incorporate generic architectural knowledge, we recommend reusing this documented knowledge in application specific architecture documentation to preserve an important part of the rationale, that went into the architect's decisions, while saving time compared to other documentation approaches.*

## 1. Introduction

The documentation of architectural knowledge (AK) in software development often consists of design models and other artifacts recording the outcome of a design process, if at all, but misses the design decisions, the context and the rationale of a specific solution [11]. This is partially because comprehensive documentation is considered a resource-intensive task that interrupts the natural design flow and does not have an immediate benefit for the running project. By skipping it, some of the important details that a decision is based on, such as the decision context, assumptions, decision drivers, consequences and considered alternatives, get lost. This phenomenon is known as architectural knowledge vaporization [2, 15, 9]. It becomes particularly problematic when software systems have to be maintained and adjusted to changing requirements or execution contexts, because we often fail to understand the architect's original motivation for a specific design construct. Consequently design rules and constraints may be violated during system evolution [9].

AK is multi-faceted. [12] makes the distinction between application-generic and application-specific knowledge. The former consists of expertise that can be generally used in many different software projects. Architecture patterns and styles are some examples. The latter concerns knowledge of a specific application, gained during its development or evolution. Application-specific knowledge includes all decisions made during the architecting process as well as information about the problem and solution space of a concrete project. In this paper we will primarily address the vaporization of application-specific AK.

Additionally application-specific knowledge is further subdivided into context knowledge, reasoning knowledge and design knowledge according to [14]. The first involves information about the problem space, for example architecturally relevant requirements, forces and the problem context. Reasoning knowledge incorporates the rationale behind decisions including considered alternatives and trade-offs. Finally design knowledge is a collection of system designs like component and connector models or other architectural models. In [14] one more category of AK is identified, namely general knowledge which is the same as the aforementioned application-generic knowledge.

Architectural knowledge vaporization occurs with all three kinds of application-specific AK. For instance, although context knowledge is represented in requirement documents, specific requirements are seldom explicitly related to architectural decisions. It remains unclear which part of the problem is solved by an architectural decision. Reasoning knowledge is usually neither represented in requirements, nor in design documents. The rationale behind design decisions, including the

design alternatives, trade-offs, assumptions and consequences gets lost if it is not explicitly documented. Design knowledge is normally represented in design documents, but the latter are often incomplete, out of date or incorrect.

In this paper, we propose a lightweight approach to support software architects in systematically documenting application-specific architectural knowledge by reusing existing, codified application-generic AK encapsulated in software architecture patterns<sup>1</sup>. We focus on recording decisions made when applying patterns and make the concrete drivers that motivated the decisions explicit, by reusing the different types of AK that patterns incorporate. This allows to uncover alternatives, trade-offs and consequences.

The remainder of this paper is organized as follows: In section 2 patterns and pattern languages are discussed with respect to their architectural knowledge aspects. Section 3 presents a conceptual model of application-generic and application-specific architectural knowledge that supports our documentation approach. In section 4 we explain, how our approach can be used to document and analyze application-specific AK. Section 5 presents an example based on a pattern story that describes a real world architecting project. Section 6 derives some basic use cases for tool support and finally, in section 7 we conclude and give an outlook to future work.

## 2. Patterns and Pattern Languages

Patterns describe the solution to a problem in a certain context [1, 6]. Therefore they capture reusable knowledge providing a design to a specific, recurring design problem arising in a particular context and domain [13]. Architectural patterns capture knowledge that is not specific to a certain project, but application-generic. They contain information about the context and the problem space in terms of forces, as well as concrete implementation advice for the solutions they propose. Finally pattern descriptions reason about design rationale, alternatives, consequences and trade-offs that are performed when applying them. This knowledge is applicable in infinite cases. When the pattern is applied in a project and all its details are finalized, the pattern's knowledge becomes application-specific for that specific project.

A pattern language incorporates patterns from a particular domain or area of concerns and combines them to form a web of related patterns. It describes a whole system of patterns, including their interrelationships and dependencies. Patterns from a pattern language can be applied one after another in an incremental process of refinement to form a whole. Therefore each pattern defines its place in possible pattern sequences [16]. By concentrating more on the various relationships of patterns and the synergetic effects of their combination, pattern languages complement the AK captured in single patterns with additional knowledge of their relationships.

## 3. A Conceptual Model of Architectural Knowledge related to Architectural Patterns

To support the effective documentation and analysis of AK, we have developed a conceptual model that identifies the types of application-generic AK provided in the documentation of patterns and pattern languages and links this knowledge to application-specific AK. The model shown in figure 1 is divided in two parts: application-specific AK elements in the upper section, and generic AK elements in the lower section.

### 3.1 Application-generic Architectural Knowledge

The central concept in the application-generic section is architectural pattern. Patterns have a name and are applicable in a context. In addition, we provide a short description, the documentation date and the source of the pattern. One of the interesting relationships between patterns is the variant. Variants of patterns describe solutions to very similar, but different problems, that can vary in some of the forces [4]. The application of a variant potentially has different consequences. Variants are considered as independent patterns, with their own problem and solution descriptions.

The relationship type alternative means that patterns can be alternatively applied in a specific context. Pattern languages sometimes explicitly mention alternatives. The Layers pattern for instance can be seen as an alternative for the Microkernel pattern (both [4]) concerning the structural decomposition of a system.

Another type used to capture the relationship between patterns is combination. This type encapsulates all other interdependencies between patterns. Pattern combinations can be very rich and complex and many subtypes exist, that are not made explicit in our model. A simple example for the combination relationship type is the Model-View-Controller pattern [4] that

---

<sup>1</sup>In the remainder of this paper we will also refer to software architecture patterns as patterns.

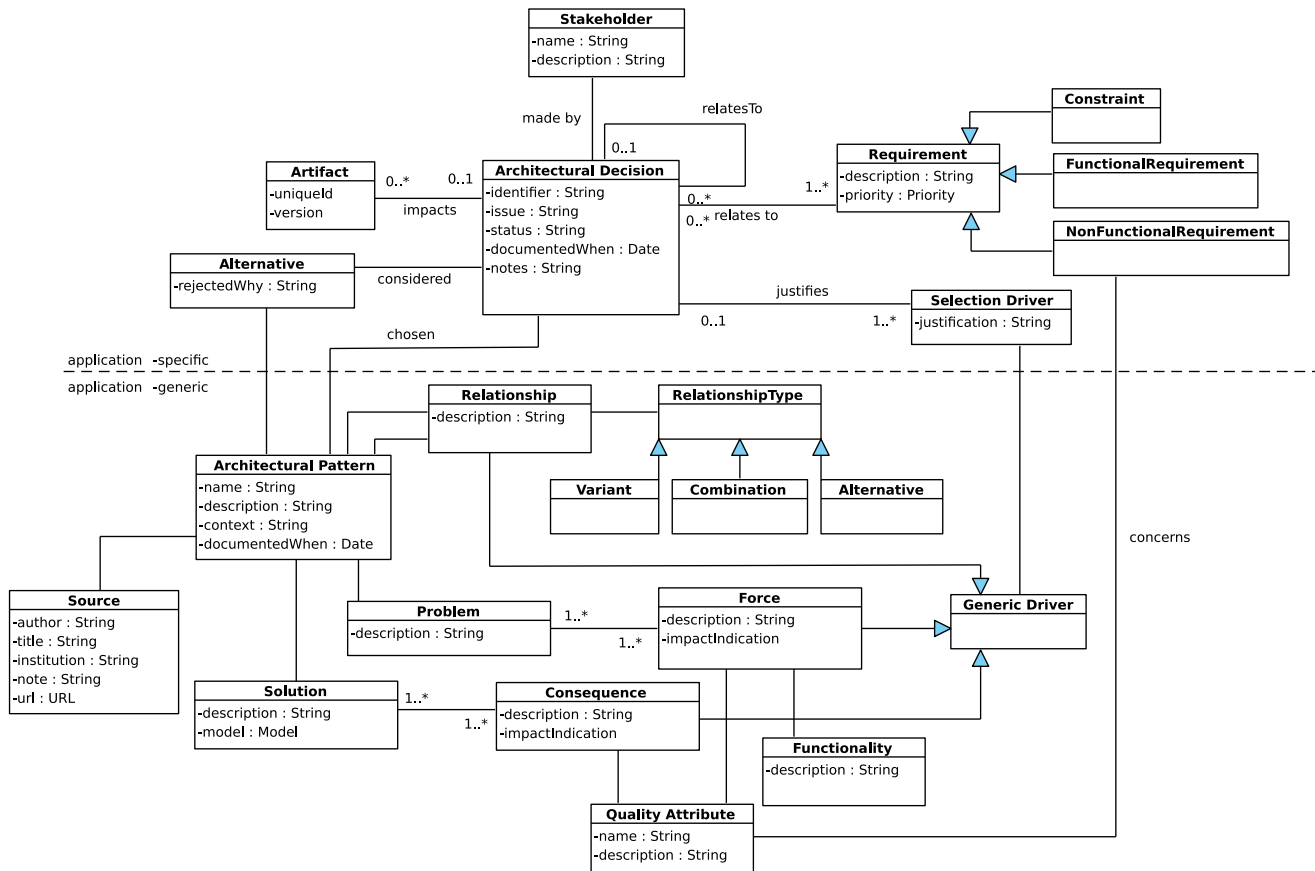


Figure 1. A conceptual model of AK related to Architectural Patterns

uses the Publisher-Subscriber pattern [4] for state change propagation. The Broker pattern [4] is another example. It can be combined with many other patterns to realize distributed systems.

The relationship between two patterns is a potential reason for applying a pattern. Consider a situation where multiple patterns have already been chosen and applied. When searching for additional patterns that address open requirements, it makes sense to choose a pattern that is known to work well with the previously selected ones, if multiple alternatives exist. By modelling the different types of relationships between patterns, we conserve reasoning knowledge that architects and stakeholders can revert to when analyzing architectural decisions later.

The context knowledge of patterns is described in terms of a problem consisting of multiple forces, while their design knowledge is incorporated in the representation of the solution. The application of architectural patterns has an impact on the quality attributes (QAs) of the target architecture by resolving related forces [10, 7, 16]. Therefore quality attributes are linked both to the forces and the consequences. The *impactIndication* field in the consequence and force classes can be used to express, whether a quality attribute is impacted positively or negatively. The terms *force* and *consequence* refer to the format for pattern descriptions used in [4]. Some pattern description formats explicitly describe forces and consequences; others like [5] do not mention them explicitly, but nevertheless contain the same information in other parts of the description. Forces and consequences are potential drivers for the selection of an architectural pattern, as they denote requirements and desired properties of the target architecture.

### 3.2 Application-specific Architectural Knowledge

The upper section of our model is concerned with the documentation of application-specific AK, and specifically architectural decisions. We are not aiming at capturing all decisions that need to be made in a software project, but those concerning the application of architectural patterns.

Patterns are abstract and generic (often called “half-baked”), and have to be finished off when they are applied in a project. We do not consider that the architecture of a system is composed only of patterns; however a fundamental part of the architecture can be constructed using architectural patterns. If so, then the AK encapsulated in the patterns is transformed into application-specific. For documenting the design decisions related to applying the patterns, the relevant parts of the pattern documentation can be reused.

In the AK model presented in figure 1, a decision addresses an issue. This can be the specification of a concrete problem to be solved, or something that needs to be decided without directly solving a problem, for instance the choice of a programming language. A decision can be related to zero or more requirements. Besides functional and non-functional requirements, a constraint is also seen as a requirement that reduces the number of possible outcomes of a decision. A non-functional requirement concerns a specific quality attribute.

An architectural decision can also be related to other architectural decisions. As an example, if the shared repository pattern was chosen to share data between two subsystems, then a related decision could concern the choice of a database access strategy like the one provided in the Table Data Gateway pattern [5]. The concrete types of relationships between architectural decisions are subject to further research.

Decisions are personalized by also documenting the decision-maker and the date when the decision was made.

The notes in architectural decision should give hints on where the chosen pattern was applied in the architecture. The same pattern can be applied in different places for different reasons.

Often, architectural decisions concern the choice of a solution among alternatives. Therefore the model explicitly considers alternative patterns along with the reasons for their rejection.

The AK behind the decision is captured by referencing an architectural pattern along with the relevant drivers for the selection of the pattern. As mentioned earlier, forces, consequences and pattern relationships may be referenced as relevant drivers. By doing so, the rationale behind the decision that the architect made is recorded. The resulting documentation of the architecture not only explains which patterns were applied, but also why they were applied, by explaining which of all the potential drivers were decisive for the architect. This does not replace application-specific design and requirements documents. It is in the application-generic nature of patterns that they are abstract and have to be adjusted and modified to fit in a concrete design situation. Referencing their pure form in an application-specific context however has the benefit that the concept of the pattern is much clearer and easier to understand in this representation than the concrete design of the architecture after the pattern was applied. Nevertheless we also tried to find an effective way of documenting the real design outcome of a decision. We suppose that ADs affect many different documentation artifacts, for example UML component or class diagrams or textual documentation. Our assumption is also that the most software projects make use of versioning systems like Mercurial<sup>2</sup> or Subversion<sup>3</sup>. By documenting the version numbers of all relevant versioning repositories before and after an AD was enforced, we conserve the architectural delta. This is the part of the architecture that the architect actually changed to apply the chosen pattern, be it textual documentation, or UML diagrams or other artifacts.

---

<sup>2</sup>see <http://mercurial.selenic.com>

<sup>3</sup>see [subversion.tigris.org](http://subversion.tigris.org)

## 4. Documentation and Analysis of Architectural Knowledge

Our approach is complementary to existing architecting processes [8]. It can be used at the time when architects apply architectural patterns to satisfy architectural requirements. The patterns described in the generic part of the model should be managed in a repository that can be used from software development projects. We will describe some basic use cases for tool support in section 6.

### 4.1 Documentation of AK

Figure 2 shows the process of documenting architectural decisions concerning architectural patterns. The process starts when the architect decides to apply a pattern in the architecture. Subsequently the core of the decision (i.e. a unique identifier, the issue, the status, the documentation date) is documented, along with the stakeholder who made the decision. If applicable, related decisions and related requirements are linked to the current decision. Then a snapshot of all relevant project artifacts is taken. This can be done capturing version numbers of artifacts managed in versioning systems like Subversion or Mercurial for instance.

Every pattern that has been applied or considered in the software project should be documented in a pattern repository. This repository can be reused in many different software projects. Hence the patterns have to be added to the repository once, if they do not exist yet.

Once the pattern is chosen and eventually documented, the architect links the chosen pattern and the considered alternatives to the current decision. Finally the concrete drivers that lead him in choosing the pattern are also linked to the decision. The last aspect is very important. A pattern potentially has many benefits and liabilities expressed in terms of consequences but also in terms of forces. But perhaps not all of them were relevant when the architect decided to choose the pattern. This information is valuable when the decision is analyzed later, e.g. during the maintenance process. It clarifies *why* the architect chose the pattern.

### 4.2 Analysis of the documented AK

In this section we will explain how the different types of architectural knowledge map to the elements in the presented conceptual model, and consequently to the AK documented using our approach.

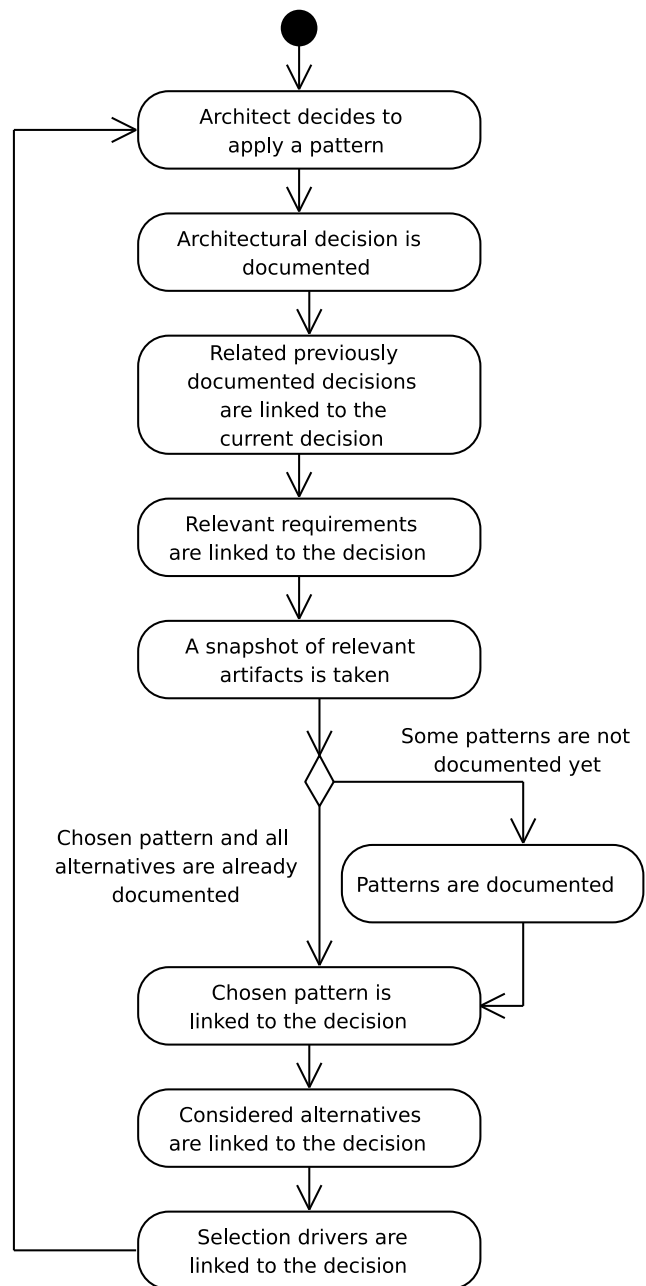


Figure 2. Documenting Architectural Decisions



### 4.2.1 Context Knowledge

Context knowledge is spread across different parts of the conceptual model. The issue field, which is part of the AD, gives a first impression of the decision context. The relation to specific requirements explains which constraints, functional and non-functional requirements are addressed by the decision. Finally the forces and problem description of the related pattern can be browsed to understand the decision context.

### 4.2.2 Design Knowledge

We suppose that large parts of the design knowledge are documented in design artifacts like box-and-line or UML diagrams. The documented patterns, however, also contribute to understanding the design.

On the one hand, the design presented in the solution part of a pattern is generic and incomplete; it has to be adapted to fit in concrete design situations. On the other hand, the design documentation of a project is more concrete and complete than the design snippets presented in pattern descriptions; but it is also more complicated and harder to understand. The pattern solution presents a very clear view that is reduced to the essentials of the respective pattern and thus easier to understand. This clear view is complemented by the architectural delta, which can be gained by comparing the artifact snapshot taken before an architectural decision was enforced to a snapshot of the same artifact after it was enforced. This allows to analyze the influence of a decision on the system design and documentation. Stakeholders can use both, the clear pattern design and the actual realization in the architecture to understand the design behind an architectural decision. In section 6 we give two examples.

### 4.2.3 Reasoning Knowledge

Making reasoning knowledge explicit, particularly the rationale behind the decisions is the most challenging part. We make use of the concrete drivers that were documented along with the chosen patterns. The drivers explain why the architect chose a specific pattern. The considered alternatives, which were also documented, correspond to the ‘paths not taken’ during the decisions making process. Additionally, the pattern descriptions of the chosen and considered patterns contain valuable information about consequences and trade-offs that were made when applying the pattern.

## 5. An example

To exemplify our approach we will use parts of a pattern story describing the design of a real-world warehouse management system presented in [3]. We focus on the part of the story that addresses the architecture of the warehouse system. The story is suitable for explaining our approach, because it describes architectural decisions concerning the application of architectural patterns taken in a real software project. The complete presented set of decisions and patterns is taken from [3].

### 5.1 A Pattern story

The pattern story concerns a control system for warehouse management. The functional requirements cover stock management, order management, shipping, receiving, material flow control and the warehouse topology management.

The following non-functional requirements were identified by the authors:

- Distribution: The functionality of the system must be accessible by clients remotely.
- Performance: The system must ensure that all orders are performed efficiently without visible interruption.
- Scalability: The system must be scalable in the number of warehouse “bins” and computational devices connected to the warehouse.
- Availability: The system demands a minimum availability of 99.999%.
- Persistence: Most state information maintained by the warehouse system must be persistent.
- Portability: The warehouse must run on different hardware platforms and operating systems.
- Dynamic configuration: The system must be runtime configurable.

- Human-computer interaction: A wide variety of user interfaces must be supported.
- Component integration: The system must integrate useful third-party products.
- Generality: The system must provide a general solution that is usable in a variety of cases in the warehousing domain.

The listed system requirements are reduced to information needed to show the applicability of our approach. Please refer to [3] for detailed descriptions.

## 5.2 Architectural Decisions made in the story

The following architectural decisions made in the story concern architectural patterns: *Layers* (AD1), *Domain Object* (AD2), *Explicit Interface* and *Encapsulated Implementation* (AD3), *Broker* (AD4), *Model View Controller* (AD5), *Half-Object Plus Protocol* (AD6), *Active Object* (AD7), logging Domain Object using *Leader/Followers* (AD8), *Database Access Layer* (AD9), *Component Configurator* (AD10).

To exemplify our approach we show its use in documenting AD1 and AD5. Then we will explain how to analyze the application-specific architectural knowledge behind the decisions. Remember that one of the goals of the presented approach is to keep the documentation effort for the architect low. The form of documentation is derived from the model presented in figure 1. Additional to the information in the tables presented in the following subsection the decision maker, status and time would have been documented in a real project, but this information was not available in the pattern story. Besides, according to the model, a reference to artifacts would exist that shows the resulting design in UML diagrams for instance. The design that resulted from the decisions here is taken from the pattern story as well.

We documented all patterns that are mentioned in the example according to our conceptual model as well, based on the descriptions in [4]. In this example we will only refer to excerpts of the documented patterns. We focus on the documentation of the actual architectural decisions.

### 5.2.1 AD1 - Layers

Architectural decision 1 is documented in Table 1. It concerns the usage of the Layers architectural pattern. The following parts of the application-specific architectural knowledge behind this decision can be made explicit using the pattern description.

- **Context knowledge:** The documented context refers to the partitioning of the warehouse system into coherent parts. The general context of the Layers pattern refers to the decomposition of a large system. The problem description of the Layers pattern in the generic part of the conceptual model leads to further insights. The Layers pattern is applicable if the system consists of a mix of high-level and low-level functionality, where the high-level operations rely on low level operations. This is the case here, high-level functionality such as order management and shipping rely on low-level functionality such as material flow control and warehouse topology management. This application-specific knowledge can be derived from the general description of the Layers pattern without explicitly being documented during the architecting process. It puts the decision in the right context. The architect was searching for a way to decompose the whole system into coherent parts while taking the specific characteristics of the warehouse system into account.
- **Design knowledge:** Although the information in table 1 does not contain any design information, a decent part of the design knowledge behind the decision can be inferred from the pattern description. First the general description of the Layers pattern includes a design template depicting the structure of the Layers pattern. It gives a clear view of the involved components and their connectors. The whole system is structured into multiple layers. Starting at the bottom, each layer contains components at the same level of abstraction. Services provided by one layer are used by components in the above layer. Note that this short description is very general in nature. It is derived from the solution documented along with the Layers pattern in our conceptual model and has no reference to the concrete case of the warehouse management system. It gives a first impression of the design that resulted from the architectural decision. More concrete information can be gathered from the reference to impacted artifacts that is part of our conceptual model. It is a pointer from the decision to the resulting design. Here, the system was actually partitioned into five layers. A presentation layer, a business process layer, a business object layer, an infrastructure layer and an access layer. In this example the information as well as any other information about the warehouse system is taken from the pattern story in [3]. There we can also find the components that reside in the respective layers.

Seq. No	1
Issue	The warehouse system needs to be partitioned into coherent parts.
Specific Requirements	<ul style="list-style-type: none"> <li>• Portability</li> <li>• Generality</li> </ul>
Arch. Pattern	Layers
Sel. Drivers	<ul style="list-style-type: none"> <li>• D1 (Force): Complex components need further decomposition.</li> <li>• D2 (Force): Support changeability.</li> <li>• D3 (Force): Support grouping of components along responsibilities.</li> <li>• D4 (Force): Support task division between programmers.</li> <li>• D5 (Consequence): Support for standardization</li> <li>• D6 (Consequence): Dependencies between components are kept local</li> <li>• D7 (Consequence): Separation of concerns</li> </ul>

**Table 1. Architectural Decision 1**

- **Reasoning knowledge:** Table 1 mentions a couple of forces and consequences that were decisive for choosing the Layers pattern. They indicate why the pattern was chosen. Note that there are other, unmentioned forces and consequences concerning the Layers pattern. One of the key advantages of the Layers pattern is support for reuse of system layers. This factor implicitly plays a role when applying the Layers pattern, but in this case it was not relevant for the architect. This information can help a lot in understanding the decision and its consequences. The application of the Layers pattern has some potential liabilities. It might lead to communication overhead when upper layers have to pass multiple intermediate layers to use functionality of low layers. This might lead to lower efficiency compared to a non-layered system where components may access each other freely. This is a trade-off the architect made when choosing the Layers pattern.

### 5.2.2 AD5 - Model-View-Controller

The documentation of architectural Decision 5 is shown in Table 2. It concerns the usage of the Model-View-Controller (MVC) Pattern.

- **Context knowledge:** The documented context refers to the separation of the warehouse's user interfaces from the warehouse functionality. The general context of the Model-View-Controller pattern taken from [4] is providing flexible user interfaces for interactive applications. The problem section of the MVC pattern includes more information.

Seq. No	5
Issue	The warehouse's User Interface needs to be separated
Specific Requirements	<ul style="list-style-type: none"> <li>• Human-computer interaction</li> <li>• Generality</li> </ul>
Arch. Pattern	Model-View-Controller
Sel. Drivers	<ul style="list-style-type: none"> <li>• D1 (Force): The user interface must reflect data changes immediately.</li> <li>• D2 (Force): Support for changeability.</li> <li>• D3 (Force): The functional core of components needs to be separated from the user interface.</li> <li>• D4 (Consequence): Views are synchronized.</li> <li>• D5 (Consequence): System parts are exchangeable.</li> </ul>

**Table 2. Architectural Decision 5**

User interfaces are likely to change more often than system functionality. Different users have different requirements regarding the user interface and often several different user interfaces must be incorporated. This also describes the specific problem the architect wanted to solve for the warehouse system by applying the MVC pattern.

- **Design knowledge:** The description of the MVC pattern explains its general design. The user interface is divided in a model containing the data, Views that display the information to the user and controllers that manage user input. Views and controllers act as observers of the model and are informed automatically if data in the model is updated. This is the application-generic solution that the MVC pattern proposes. Again, the artifact reference adds application-specific design knowledge. Here we could see, that the views and controllers reside in the presentation layer, while the model is represented in the business process layer. Change propagation components were introduced for providing messaging functionality that is used to inform controllers and views if an update occurs. Please refer to [3] for more details. This example shows, that the straightforward design that is proposed by the MVC pattern description helps to understand the application-specific design solution. In the real design the involved components have more than one functionality and names that refer to the respective application domain. This makes it harder to find out, that the MVC pattern has actually been applied and even harder to understand how the solution works.
- **Reasoning knowledge:** Table 2 shows the selection drivers that let the architect choose the Model-View-Controller (MVC) pattern [4]. They indicate why the pattern was chosen. There are other potential drivers for choosing MVC. For example the possibility for presenting the same information differently in multiple views. Although this possibility is automatically given when applying MVC it was not decisive for the architect. Some negative consequences come along with the MVC pattern. MVC introduces a very close coupling between view and controller. When porting the user interface it is very likely that both have to be changed. MVC also leads to more complexity and the potential for excessive updates of multiple views resulting in a large communication overhead. These are liabilities that the architect accepted for getting the advantages of the MVC pattern.

## 6. Tool support

To tap the full potential of the presented approach, comprehensive tool support is indispensable. We have started to elicit some basic high-level use cases for a tool supporting our approach.

The following two use cases support the management of the application-generic AK from the conceptual model in figure 1. Essentially, there has to be create, retrieve, update and delete (CRUD) functionality to manage all entities in this part of the model:

- **UC1: Manage architectural patterns:** This use case includes adding, updating and deleting architectural patterns. According to our conceptual model, every pattern must be described in terms of a context, a category, a problem statement, forces, a representation of the solution and consequences. Additionally technologies supporting the patterns can be managed.
- **UC2: Manage pattern relationships:** Maintain the various relationships between the patterns in the repository. Basic types of relationships are variant, combination and alternative. Each of them should be refineable.

To support the architectural documentation process presented in section 4, the following use cases are applicable:

- **UC3: Document architecture relevant requirements:** Functionality is needed to select and document architecture relevant requirements from existing project documentation.
- **UC4: Add architectural decision:** When the architect decides to apply a pattern in the architecture, it needs to be recorded as an architectural decision. Every documented decision references the relevant drivers for choosing the concerned pattern, as well as a set of requirements that the decision satisfies. Additionally it must be possible to link a decision to a repository version in versioning systems like CVS or Subversion. By this it is possible to document how existing design documents or generally all versioned project artifacts have been affected by the decision.
- **UC5: Explore decision rationale:** Additionally to the chosen pattern, the satisfied requirements and the relevant drivers, it must be possible to explore the consequences of the chosen pattern, alternative patterns, variants and other patterns, that are related to the chosen pattern. This information can be taken from the data captured in the application-generic part of the conceptual model. It should also be possible to visualize the change in design documents and other artifacts that the decision caused.

Some of the presented use cases reference existing project artifacts like requirement documents and versioning systems. Therefore it would make sense to develop the documentation tool as an extension to existing Computer Aided Software Engineering (CASE) tools that are used to create analysis and design documents. We are going to implement these use cases in prototypes to validate our concepts. One of our goals is to provide lightweight tooling, that supports our approach and easily integrates with existing CASE tool chains to keep the barriers for its usage low.

## 7. Conclusion and Outlook

In this paper we presented an approach to address architectural knowledge vaporization by documenting decisions concerning the application of architectural patterns. A part of application-specific architectural knowledge comes from instantiating application-generic architectural knowledge in the form of architectural patterns. We make use of this by referencing chosen patterns, considered alternatives and specific decision drivers when documenting architectural decisions. This keeps the documentation effort that an architect has to spend during the architecting phase low, while preserving great parts of the rationale that went into the decisions. The patterns that are referenced do not have to be documented during the architecting phase or by the architect himself. As they are general in nature, they can be easily documented in advance and then be reused in many different software projects. We proposed a conceptual model of AK that supports our approach.

The process of documenting ADs as well as the analysis of the specific knowledge can easily be supported by tools. We described some basic use cases for that.

Our approach needs to be extended. Patterns do not cover the whole problem space of applications. Not every architectural problem can be solved by applying a pattern. If no suitable pattern exists, then the current approach cannot be used. Additionally, not all architectural decisions concern the usage of architectural patterns. Some architectural decisions concern the selection of technologies instead of patterns. Existing software systems, frameworks, middlewares and application platforms

are some examples. These decisions cannot be documented using our model yet. However, we assume that technologies can be described similarly to architectural patterns. We are currently looking into documenting architectural decisions concerning the use of technologies in the same way as decisions concerning patterns.

One might argue that documenting patterns in a repository is actually a high effort that interrupts the architect's design flow, but this is not necessarily the case. First, the patterns in the repository do not have to be documented by the architect himself. There's not much expertise needed to add a pattern to the repository based on a pattern description in a book or an article. Second the patterns do not necessarily have to be documented during the architecting phase. It would be even better to have a repository of potentially applicable patterns before the architecting phase starts. Ideally such a repository would even be publicly available for use and contribution.

We proposed a documentation approach that allows architects to record architectural decisions related to patterns. By referencing chosen patterns, considered alternatives and the concrete drivers that induced the architect to choose the pattern, we implicitly preserve the rationale behind the decision including variants, consequences and related patterns. Effort has to be spent once to document the patterns. They can then be reused by referencing them in different software projects. The documentation of the decisions does not require extra effort.

## 8. Acknowledgement

Thanks to Neil Harrison for giving good advice and providing useful feedback during the shepherding of this paper for EuroPLoP 2009.

## 9. Copyright

Copyright retains by authors. Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for the Hillside Europe website.

## References

- [1] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press, August 1977.
- [2] J. Bosch. Software architecture: The next step. *Software Architecture*, pages 194–199, 2004.
- [3] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*. Wiley, May 2007.
- [4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley, August 1996.
- [5] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, November 2002.
- [6] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, November 1994.
- [7] N. Harrison and P. Avgeriou. Leveraging architecture patterns to satisfy quality attributes. In *Proceedings. First European Conference on Software Architecture*. Springer LNCS, 2007.
- [8] C. Hofmeister, P. Kruchten, R. Nord, H. Obbink, A. Ran, and P. America. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126, January 2007.
- [9] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Proceedings. WICSA 2005. 5th Working IEEE/IFIP Conference on Software Architecture, 2005*, pages 109–120, Washington, DC, USA, 2005. IEEE Computer Society.
- [10] A. Jansen, J. Bosch, and P. Avgeriou. Documenting after the fact: Recovering architectural design decisions. *Journal of Systems and Software*, 81(4):536–557, April 2008.
- [11] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer. Tool support for architectural decisions. In *Proceedings. Working IEEE/IFIP Conference on Software Architecture, 2005*, volume 0, pages 4+, Los Alamitos, CA, USA, 2007. IEEE Computer Society.
- [12] P. Lago and P. Avgeriou. First workshop on sharing and reusing architectural knowledge. *SIGSOFT Softw. Eng. Notes*, 31(5):32–36, 2006.
- [13] D. C. Schmidt and F. Buschmann. Patterns, frameworks, and middleware: their synergistic relationships. In *Proceedings. 25th International Conference on Software Engineering, 2003.*, pages 694–704, May 2003.
- [14] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar. A comparative study of architecture knowledge management tools. *Journal of Systems and Software*, September 2009.
- [15] J. Ven, A. Jansen, J. Nijhuis, and J. Bosch. Design decisions: The bridge between rationale and architecture. In *Rationale Management in Software Engineering*, chapter 16, pages 329–348. 2006.

- [16] U. Zdun. Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience*, 37, 2006.

# Business Patterns for Knowledge audit implementation within SMEs

---

*Albena Antonova<sup>1</sup>, Elissaveta Gourova<sup>2</sup>*

<sup>1</sup>*Sofia University, Faculty of Mathematics and Informatics, Sofia, Bulgaria, [a\\_antonova@fmi.uni-sofia.bg](mailto:a_antonova@fmi.uni-sofia.bg)*

<sup>2</sup>*Sofia University, Faculty of Mathematics and Informatics, Sofia, Bulgaria, [elis@fmi.uni-sofia.bg](mailto:elis@fmi.uni-sofia.bg)*

## Abstract

In the last decades, Knowledge Management has gained momentum as an important tool for competitiveness of organizations. A number of Knowledge Management approaches are described in the literature. Some authors focus in particular on the importance of Knowledge Audit, first, as a start point for any Knowledge Management initiative, and second, as a regular base for the measurement of Knowledge Management progress and effectiveness. Knowledge Audit aims to investigate the company status at a given moment regarding the knowledge availability and needs, its flow and usage in processes, by employees, etc.

The present paper aims to introduce business patterns for the implementation of Knowledge Audit. These patterns describe the process of taking decisions for Knowledge Management implementation and the first step toward it – auditing knowledge. After an overview of the concept, objectives and results of Knowledge Audit, some practical recommendations for successful Knowledge Audit practice are suggested and a systematized approach for the Knowledge Audit process is presented.

**Keywords:** *Knowledge Audit, Knowledge Management, Patterns*

## 1 Introduction

Today, the information overload raises new challenges to individuals and organizations. Global networks provide access to an enormous quantity of information and knowledge coming from a great variety of sources. At the same time, the mobility of knowledge workers, and the increased value of information and knowledge have significantly raised the importance of knowledge assets and their proper usage for higher competitiveness and growth. Subsequently, Knowledge Management has gained momentum as an important tool for competitiveness of organizations. Its successful implementation, however, depends on a number of interrelated factors, including technology, human beings, organizational culture and leadership, etc. The Knowledge Audit if properly carried out contributes to building a Knowledge Management strategy based on extended knowledge of the company status, its internal and external environment, and thus, enables the organization to take appropriate decisions to overcome existing gaps and possible drawbacks.

Linking organizational strategy with the Knowledge Management strategy is the first step towards Knowledge Management in organizations [6]. Here, a clear understanding is necessary of the existing knowledge gaps coming out of the recognized strategic gaps. Therefore, a need emerges to make an analysis of the available knowledge assets, their usage, the knowledge processes and flows in the organization, etc. The Knowledge Audit is



the appropriate tool for answering all these issues, and at the same time, it is an important tool for monitoring of Knowledge Management effectiveness [1, 2]. Knowledge Audit, like other audit processes and methodologies, aims to investigate the company status at a given moment regarding the knowledge availability and needs, its flow and usage in processes, by employees, etc. In fact, Knowledge Audit is a repetitive process aiming to clarify whether knowledge resources are properly managed and what Knowledge Management strategy, tools and solutions could contribute to gaining maximum benefits [1].

The concept of patterns is widely described in literature [12, 13, 14, 15]. The authors were involved in a project for developing business cases and studied the eXperience methodology for case studies development [16]. Application of case studies was considered very suitable for the teaching process for Knowledge Management at University of Sofia. At the same time, after studying several Knowledge Management cases, as well as getting to know the patterns approach, the authors came to the idea to capture and apply patterns for Knowledge Management purposes. Subsequently, the aim of this paper is to introduce business patterns for Knowledge Audit implementation. These patterns describe the process of taking decisions for Knowledge Management implementation and the first step toward it – auditing knowledge. The paper captures the following patterns:

1. Knowledge Audit Plan
2. Knowledge Audit Team
3. Knowledge Audit Methodology
4. Knowledge Audit Questionnaire
5. Knowledge Audit Report

### **Audience**

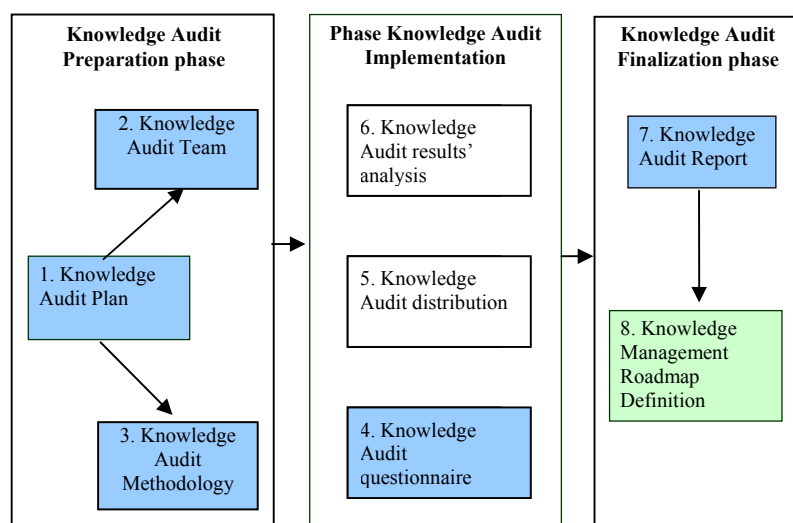
These patterns are intended to codify business practices in the area of Knowledge Audit implementation in a pattern language so that they may be better understood, communicated, applied and studied.

The patterns are intended for Knowledge Management practitioners and Knowledge Management enthusiasts, for Small and Medium Enterprises (SMEs) managers and entrepreneurs, for students, experts and consultants. The patterns may be applied in the context of SMEs or knowledge-intensive public or private organizations.

## 2 The Patterns

This paper considers the main phases and processes of Knowledge Audit implementation (Fig. 1) and each process links to a pattern to be followed by practitioners. From the Knowledge Audit road-map will be presented the patterns 1, 2, 3, 4 and 7:

<b>1. Knowledge Audit Plan:</b>	Planning of Knowledge Audit scope, activities and time schedule
<b>2. Knowledge Audit Team:</b>	Selecting the right Knowledge Audit Team plays an important role for the Knowledge Audit outcomes
<b>3. Knowledge Audit Methodology:</b>	Methodology how to perform and implement successfully specific Knowledge Audit tasks and activities.
<b>4. Knowledge Audit Questionnaire:</b>	How to select, compose or adapt Knowledge Audit Questionnaire according to specific company needs
<b>5. Knowledge Audit Questionnaire Distribution:</b>	Methodology for Knowledge Audit distribution (via e-mail, paper-based questionnaires, conducting interviews, on-line questionnaire, mixed approach), and notification of the target audience.
<b>6. Knowledge Audit Analyses of Results and Feedback:</b>	Analyses of the Knowledge Audit results, testing and verifying hypothesis based on the collected quantitative and qualitative data. First feedback of the results.
<b>7. Knowledge Audit Reporting:</b>	Preparation and presentation of meaningful Knowledge Audit Report as major outcome of the Knowledge Audit process
<b>8. Knowledge Management Roadmap Definition:</b>	Knowledge Management Roadmap consideration



**Figure 1** Knowledge Audit phases and processes

## 2.1 SUCCESSFUL KNOWLEDGE AUDIT PLAN

---

### 2.1.1 Context

An organization wants to implement Knowledge Management or needs to monitor Knowledge Management effectiveness and improve it by taking new knowledge-related initiatives. It needs to prepare a detailed Knowledge Audit plan, and report it to the company management as a preliminary step for approval of the Knowledge Audit implementation. The organization delegates the Knowledge Audit Plan preparation to external consultants or to internal staff such as the Chief Knowledge Officer, the Knowledge Management team, or experts from the Human Resources and/or the Information Technology (IT) department.

### 2.1.2 Problem

**How do you prepare a successful Knowledge Audit Plan for an organization?**

### 2.1.3 Forces

The Knowledge Audit plan needs to identify and clarify the principle hypothesis expected to be justified in the audit process. The forces influencing the Knowledge Audit Plan can be grouped around 4 main areas:

*Force 1:* Companies are aware of the importance of measuring innovation and knowledge creation results, but often they do not measure the right things, do not measure enough, and, in some cases, do not measure at all. The measurement of intellectual capital, and in particular knowledge assets, creates large difficulties and there is a lack of a generally accepted methodology for valuing intangible assets. However, the increasingly complex environment, the fast changing technologies and increasing customers' expectations, as well as the evolving new complex relationships with employees (part-time, free-lancers), contractors, external experts etc., force companies to rethink the overall Knowledge Management strategy (what in fact they know and how they can use it), and thus, regularly undertaking Knowledge Audit.

*Force 2:* Knowledge exists in explicit and tacit form in organizations, but it is hard to identify and measure it. Knowledge can be embedded in various forms: documents, procedures, methodologies, routines, organizational culture, group practices, IT systems, databases, people heads, lessons learned, best practices, social networks and social interactions and many others.

*Force 3:* Knowledge Audit is a time- and resources-consuming process, but companies are not eager to invest much in such initiatives. It is essential to properly clarify the scope of Knowledge Audit, its main objectives and expected deliverables, to prepare a time schedule and allocate resources for its implementation.

*Force 4:* Knowledge Audit implementation needs a team of experts with in-dept knowledge of its processes, but also aware of the business processes, strategic goals and assets of the organization. The knowledge gaps can be discovered only after sophisticated analyzes of the existing knowledge resources and assets, so experts involved need to have access to them and to understand the strategic goals of the company, as well discovering the driving forces in the environment.

#### 2.1.4 Solution

##### **Be clear for the purpose of Knowledge Audit and balance organizations' needs with available resources**

The Knowledge Audit Planning should be performed like a company project aimed at identifying, analyzing and tracing knowledge assets in the company. It is essential to identify first the basic requirements for the Knowledge Audit, and the purpose of doing it – if it is initial auditing process or will be a repetitive process in a Knowledge Management initiative, or exceptional audit aimed at supporting decision making in the organization.

Second, determine what will be measured, e.g. internal or external knowledge resources, knowledge gaps in a specific area, tacit and/or explicit knowledge, IT Systems and applications used for knowledge management, social network analyses and knowledge sharing, or organizational culture, etc.

Third, ensure management support by setting clear, measurable and achievable objectives for Knowledge Audit, and showing the expected impact on company management and the benefits of investing time and efforts in this process. The planning document should state which results are expected, which will be the project size – a pilot audit or involving the whole staff and departments, or if it will focus on permanent staff and/or part-time staff, etc. There is a need to balance the competing demands for quality, scope, time and cost of the Knowledge Audit. Thus, set scope, expected time framework and cost for the Knowledge Audit in efforts (working hours) and show how will be guaranteed the quality of the results by management supervision and active participators feedback.

Finally, clarify who will be in charge of the project, choosing between available internal resources or external consultants or a mix of both. For the Knowledge Audit team should be determined the main expertise needed, the responsibilities and tasks to be accomplished. Leadership is essential, and senior staff should be involved in the overall Knowledge Audit process. This will ensure the Knowledge Audit visibility, and that no important knowledge assets, processes and skills forming company competitive advantage are missed.

The organization can choose between two main approaches for Knowledge Audit implementation – outsourcing the Knowledge Audit activity to a consulting company or designating a team within the company to carry out the Knowledge Audit. It is possible to combine both approaches, involving external experts in the Knowledge Audit team or to elaborate the own Knowledge Audit methodology collaborating with external consultants. Some stages of the Knowledge Audit can be implemented internally and others – externally. At the time of planning should be considered the advantages and disadvantages for both methods:

- **Choosing a Consulting company:** Consultants can provide unbiased assessment of organizational knowledge assets, not taking into account personal experience, prejudices gossip, etc. However, consultants can miss some important sources of knowledge, not getting deeply into details of operations or ignoring corporate culture, competitive advantage etc. Besides, they usually follow their own methodology for Knowledge Audit that is more general and not company-specific.
- **Internal team:** Company team knows very well the operational activity of the organization; knows employees and principle knowledge flows. Often the Knowledge Audit team members become the most serious Knowledge Management champions. However, employees often have a narrow view of the company activity, so they can overestimate or underestimate important knowledge opportunities and strengths.

Assessment can be biased by personal attitude and emotions of the team members. They normally have less expertise in the methodology of Knowledge Audit.

### **2.1.5 Consequences**

You have laid the foundation for a clear and meaningful Knowledge Audit that can be easily performed and reproduced. You are now ready to perform the Knowledge Audit. The Knowledge Audit planning provided you the Knowledge Audit requirements and context, the knowledge assets to be measured, the goals and tasks to be performed, the organization of the Knowledge Audit process and its main risks, the resources needed and the project time framework, and last, but not least determined how the Knowledge Audit team will be composed in order to overcome the limitations.

### **2.1.6 Rationale**

Knowledge Audit Planning is crucial step towards better Knowledge Management implementation. Better understanding of organizational knowledge processes and assets and the knowledge life-cycle is substantial for its better exploitation. The Knowledge Audit Plan ensures a well performed Knowledge Audit process and guarantees clear standards and procedures.

### **2.1.7 Example**

The planning of Knowledge Audit processes is discussed in a number of case studies and research projects. Detailed planning was performed during the project TRAINMORE-KNOWMORE [11], as the overall requirements and objectives of the Knowledge Audit process were set up. As a project outcome, a Knowledge Audit self-evaluation tool with detailed instructions for further use was designed. Overall 14 pilot Knowledge Audits were performed following the proposed methodology and comparable results are reported. The Knowledge Audit Planning guarantees the Knowledge Audit process and results and improves the Knowledge Audit implementation in practice.

## 2.2 KNOWLEDGE AUDIT TEAM

---

### 2.2.1 Context

The company decided to implement Knowledge Audit and approved a plan for implementing it using internal resources and expertise. As the Knowledge Audit is a complex instrument to identify the knowledge assets' state-of-the-art and future trends, it has to be performed carefully by knowledgeable and experienced staff. The company should select a well-balanced team of experts to carry out all Knowledge Audit processes and analyses. It is important to identify the necessary expertise and choose among company experts.

### 2.2.2 Problem

#### **How to form a good Knowledge Audit Team?**

### 2.2.3 Forces

The proper composition of the Knowledge Audit Team determines how successful will be the Knowledge Audit and the Knowledge Management processes.

*Force 1:* The team has an important role to identify and analyze the knowledge within the company and to perform the Knowledge Audit process. Having motivated, open-minded and educated team members will guarantee well performed Knowledge Audit processes and outcomes. The selected team should identify knowledge assets and perceive the important knowledge and communication flows, taking into account that a big part of the knowledge is informal, tacit, personal and fuzzy. However, it is not easy to find a proper mix of skills, both personal and expert in only one company unit.

*Force 2:* Team members need to understand the strategic vision and the global business processes of the organization and its environment, but they should also know in details the business processes and core knowledge assets of the company, as well as how technology is used to support organizational performance.

*Force 3:* Team members should understand the Knowledge Management and Knowledge Audit principles, Knowledge Audit goals and processes, specific Knowledge Management tools and techniques, but also have leadership skills and be able to motivate people and involve them in the Knowledge Audit processes.

### 2.2.4 Solution

#### **Make a mixed team of experts from different functional areas of the organization**

The Knowledge Audit Team should be composed of experts coming from several departments within the company in order to ensure the necessary mix of expertise and skills. Ideally, the team should include people from different levels of the organization in order to ensure the knowledge of strategy and company mission, as well as the awareness of company customers and suppliers and operational daily tasks (knowing well the product, technology, service). In order to equip the team with knowledge of technology, human resources, research methodology or accounting, the team should include also representatives of different functional areas within the company. It will be wise to invite persons with substantial vision about the tacit knowledge within company and people with good social (informal) networks.

Ideally, the Knowledge Audit team is composed of:

- **Corporate strategist:** Sets goals, determines optimal performance levels, brings the big picture perspective into the analysis.
- **Senior management, company visionary, or long-term planner:** Brings long-term KM vision, aligned with the business strategy of the corporate strategists.
- **Financier:** Brings the ability to value and attach a fair-dollar figure to knowledge assets.
- **Human resource manager:** Brings good understanding of employee skills and skills distribution within the organization.
- **Marketing specialist:** Provides a fair picture of actual market performance of the firm and the possible implications of its knowledge assets on the marketability of the company products and services at new price-service function points.
- **IT expert:** Brings in knowledge, skills, and expertise for mobilizing the technology implementation aspects of your knowledge management strategy. Also has intimate knowledge of existing infrastructure.
- **Knowledge manager, CKO, or knowledge analyst:** The middle role that integrates inputs from all other participants on the knowledge audit team in a consensual, and fair manner. The analyst contributes a reasonably accurate market valuation of proprietary technology and processes based on perspectives elicited from other team members.

### 2.2.5 Consequences

The successful Knowledge Audit Team identifies the main Knowledge Audit forces and takes principle considerations about knowledge assets in organization. The Knowledge Audit Team has to overcome the basic limitations of the Knowledge Audit approach, preventing it from focusing only on people (tacit knowledge) or only on documented and codified knowledge and IT. The Knowledge Audit Team determines the main hypothesis of the knowledge within company and it organizes and implements the Knowledge Audit processes. The heterogeneous team will overcome successfully the personal (biased) look and understanding for organizational knowledge, and will build complex and dynamic model of organizational Knowledge Management.

After appointing the Knowledge Audit Team, you should train it and motivate it how important is Knowledge Audit and Knowledge Management for the company. In principle you are now ready to perform the Knowledge Audit processes. The Team has the necessary set of expert knowledge to accomplish successfully the Knowledge Audit.

## 2.3 KNOWLEDGE AUDIT METHODOLOGY

---

### 2.3.1 Context

An organization decided to conduct Knowledge Audit, elaborated a Knowledge Audit Plan and appointed the Knowledge Audit Team. It scoped the Knowledge Audit project and took general decisions about the Knowledge Audit implementation resources and schedule. It needs to decide how to carry out its tasks in order to achieve the results considered in the Knowledge Audit Plan.

### 2.3.2 Problem

**How to choose the right sequence of Knowledge Audit actions in order to accomplish successful Knowledge Audit within the specific organization?**

### 2.3.3 Forces

The Methodology for implementing Knowledge Audit should be adapted to the specific situation in the organization. It should reflect not only the company status and profile, but also some constraints like cost, time, and staff. At the same time, it should produce and guarantee the desired Knowledge Audit outcomes. The Knowledge Audit team has to discover the most convenient among the existing Knowledge Audit methodologies, depending on the desired outputs and management practice.

*Force 1:* The Knowledge Audit Team needs a proper methodology and sequence of tasks and activities in order to perform successfully the Knowledge Audit, but there is a big choice of Knowledge Audit approaches in research and practice [5, 6, 7, 8]. It is important to make a good choice out of available methodologies or develop its own approach. Normally, consultant companies and Knowledge Audit experts come with their own methodology for Knowledge Audit. Thus, this could be an optional step if the Knowledge Audit project is outsourced.

*Force 2:* The Knowledge Audit Plan has provided the scope and objectives of the project, but there is a need to elaborate more working details and choose the proper tools for measurement of knowledge assets and flows. In order to plan and allocate properly the necessary efforts and time, some further details should be taken into consideration:

- The company staff status profile /number, education, age, experience, expertise, turnover rate/
- The level of knowledge codification, IT infrastructure and knowledge available in electronic form (in data warehouses)
- The way of the processing knowledge coming from clients/suppliers, third parties
- The value of tacit knowledge (and know-how), value of social networks, informal/formal knowledge sharing in the company value-creation process
- How knowledge-intensive is the industry/sector and what are the general trends among main competitors
- How will look the expected Knowledge Audit outcomes.

*Force 3:* A large variety of knowledge audit tools are proposed in research and practice, but it is important to choose those of them which are easy to implement and will help to gain the needed results and meet the objectives. The Knowledge Audit team should choose if it needs to make a full Knowledge Assets map and Intellectual Capital Inventory, Knowledge Flowchart and Analysis, carry out a Competitive Knowledge Analysis, Critical Knowledge Function Analysis or Knowledge Management Benefit Assessment, etc., and if it should focus the measurement on qualitative or just quantitative approaches.



### 2.3.4 Solution

**Find the right balance between activities to be performed and tools to be used, and the necessary resources for achieving the best results**

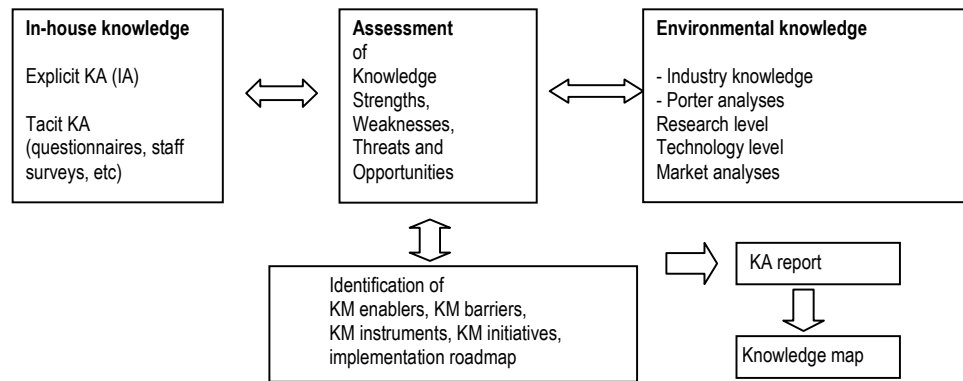


Figure 2: Knowledge audit processes details

In practice, companies adapt the methodology for Knowledge Audit to their specific needs after studying the well-known approaches and tools. In order to better guide the Knowledge Audit processes (Fig. 2), assess and evaluate the company knowledge status, and produce the expected Knowledge Audit outcomes, the following approach could be followed:

1. **Identifying state-of-the-art** – making an overview of documents (explicit knowledge) about organizational knowledge assets, processes, workflow procedures and internal structures. This is an important step for Knowledge Audit Team to acquire an overview of operations and workflow, and to get a strategic vision for Knowledge Management roadmap development. The Knowledge Audit Team has to estimate the value of tacit knowledge, tacit knowledge sharing practices, organizational culture and motivation.
2. **Focus setting** – choosing the target group for Knowledge Audit – the overall company, specific department/s or teams and work groups
3. **Adjustment of inventory** – customizing the audit tools (questionnaires, interview scope) to the company/audit objectives and requirements
4. **Conducting survey** – gathering data (sending questionnaires to the selected target group and/or conducting face-to-face interviews with the process owners). Sometimes it can be useful to organize a workshop to explain the Knowledge Audit goals and objectives encouraging employees to participate, and thus, increasing the feedback rate.
5. **Analyzing the results** – evaluate data, verify Knowledge Audit consistency (response rate, level of participation) and draw general Knowledge Audit conclusions;
6. **Preparing a Knowledge Audit report**, presenting the Knowledge Audit conclusions and suggesting list of suitable actions facilitating Knowledge Management implementation – as modeling of business processes, improvement of existing Knowledge Management policies and procedures (for example review of HRM policy, adapting remuneration policy, improve motivation culture for knowledge sharing), creating a roadmap with recommendation for further actions;
7. **Organizing feedback workshop** – by means of workshop, the results are reported back to the management and public and the suggested measures are prioritized – establishing a detailed Knowledge Management roadmap and Knowledge Management action plan;

8. **Preparing a Knowledge Management implementation project**, based on the approved Knowledge Management roadmap and action plan

The Knowledge Audit process needs strong support from organizational leadership and commitment and engagement of all employees. Therefore, it is recommended that before initiating any Knowledge Audit activities to inform all employees about their objectives using suitable form and tools (workshop, presentation, public discussions and debates, company newsletter, or organizing information kiosk). In order to achieve successful Knowledge Audit results, all employees should understand and support it (and thus minimizing internal opposition and misunderstanding).

### 2.3.5 Consequences

The Knowledge Audit Team is now equipped with a methodology and has better understanding of all tasks to be performed and tools to be applied, as well as how to implement in practice all Knowledge Audit processes. The company is well prepared for the next steps, and will achieve better Knowledge Audit performance.

The specification of Knowledge Audit details will ensure clear implementation procedure and guarantee better performance. Thus, the company will be able [5]:

- to uncover strengths and weaknesses within the actual corporate management of knowledge assets and business processes;
- to analyze circumstances, barriers and enablers of the Knowledge Management as corporate culture, leadership, human resources management (HRM), information technology (IT), process organization and control;
- to increase awareness of Knowledge Management within the company;
- to design a roadmap for Knowledge Management implementation and measure;
- to collect measurable data for controlling purposes.

### 2.3.6 Rationale

All Knowledge Audit approaches have a common feature – their focus on the current status of the company knowledge – locating it throughout the organization, and examining knowledge flows and processes. The real Knowledge Audit should, however, go beyond the company internal status and deliver a broader picture of the global processes and stakeholders, and the knowledge position of the company against its competitors. It should include, in addition to the competition or industry branch analysis, an analysis of the level of technology development, current research state, available resources and macroeconomic perspectives, customer demands and requirements, industry growth trends, leading industry experts and human factors. This analysis will guarantee more successful level of Knowledge Management implementation and better action plans, while designing Knowledge Management tools, IT systems or HRM techniques.

## 2.4 KNOWLEDGE AUDIT QUESTIONNAIRE

---

### 2.4.1 Context

The Knowledge Audit Plan, Team and Methodology are in place. The company decided to use a questionnaire as a reliable tool to carry out the analysis and investigate the state-of-the-art of knowledge assets, knowledge exploitation and knowledge flows.

### 2.4.2 Problem

**How to prepare the questionnaire in order to ensure detailed inputs on organizational knowledge assets? How to compose, customize, adapt, or select it?**

### 2.4.3 Forces

When composing and using questionnaires in Knowledge Audit process it should be taken into account form, content and methodology of the Questionnaire process:

- **Obtained Knowledge Audit Results:** The main problem of composing and using questionnaires is that they reflect not the real facts, but personal opinion about these facts. If formulated not properly, questions could mislead the respondent and provide biased results. On the other hand, tacit knowledge could not be easily recognized, and thus, cannot be easily reported by employees. Questionnaire has to be adapted to the Knowledge Audit purposes, focusing on specific objectives and goals stipulated in the Knowledge Audit Plan.
- **Questionnaire Form:** The Knowledge Audit aims at gathering more information and knowledge from the employees. However, the sequence of the questions and the length of the questionnaire can influence the responses and the return rate.
- **Limitations and Constraints:** It is important to obtain personal data for gaining maximum information and tracing it to the source for further clarifications, if needed, but the legal limitations and privacy should be respected. At the same time, often people will avoid responding frankly if anonymity is not guaranteed.
- **Questionnaire Content:** It is important to consider what type of questions to include in the Knowledge Audit Questionnaire and how to balance the content using both open-ended and closed questions. Open-ended questions could provide more information and insights from employees, but their processing is more difficult and time-consuming.

### 2.4.4 Solution

**Ensure balance between the objectives of the questionnaire and its length and content while respecting legal and personal constraints**

The Knowledge Audit Questionnaire should respond to the purpose and objectives of Knowledge Audit. It should be composed carefully, taking into account the best practices, available in literature [11]. The derived solution should be considered from some general points of view:

- **Knowledge Audit Results:** The Knowledge Audit Questions should mainly focus on facts, while some sections can ask for personal opinion ("what do you think will/should..."). It is important to determine in advance the quantitative and qualitative output data needed for further analyzes, and to optimize the number of questions. It is advisable to have a short overview of the questions and make revision of style, terminology and language of the Questionnaire in order to be clear and unbiased.
- **Questionnaire Form:** The Knowledge Audit Questionnaire should differ depending from the media and delivery method used – face-to-face interviews, online questionnaires or paper-based form. The questionnaire length should reflect the way of distribution. It is advisable to make a concise questionnaire that could be filled within 5 - 10 minutes while it is sent by e-mail, post or electronic form. More detailed

questionnaires could be used in face-to-face interviews and group discussions. Inform people in advance about the time needed to fill in the questionnaire or to conduct the interview.

- **Questionnaire Limitations:** Avoid questions that could be treated as unethical or are asking for sensitive personal information. Select appropriate scale for response (yes/not, scale of 3/5 positions, open response). Always give an option for answering "I don't know" and "other". Clearly identify that responses will be treated anonymously, but personal information is needed for further clarifications and details.
- **Questionnaire Content:** Begin by determining the basic sections of the questionnaire. The questions within the sections can be formulated later depending on how deep and a detailed analysis is needed.

#### 2.4.5 Consequences

The Knowledge Audit Questionnaire has identified and provided first-hand raw data about:

- Core knowledge assets and knowledge flows – who create knowledge and who use it
- Gaps in information and knowledge needed to manage the business effectively
- Areas of information policy and ownership that need improvement
- Opportunities to reduce information-handling costs and to improve coordination and access to commonly needed information
- A clear understanding of the contribution of knowledge to business results
- IT use and application for Knowledge Management in business
- Measurable outcomes for the company culture
- Understanding of social relationships and network analyzes
- Motivation techniques that could best fit to the organization

#### 2.4.6 Rationale

The Knowledge Audit Questionnaire is a critical instrument for collecting first-hand raw data, adapted to the focus of the Knowledge Audit. It should complete the other sources of data, available in the company, as company records, data bases, documents, workflow analyses, etc. The best picture of the overall knowledge combines both – the explicit knowledge overview with documents and procedures and the Knowledge Audit results covering tacit knowledge and company culture.

#### 2.4.7 Example

A detailed questionnaire was developed during the Leonardo da Vinci project TRAINMOR-KNOWMORE [11]. It is adapted especially for SMEs and public organizations. The questionnaire was tested in organizations in partners' countries. It included several sections which could be adapted to the organizations' specific needs, and the questions could be deepened according to the goals of the analyses:

- Demographic analyses
- Knowledge Profile Analysis
- Work Nature Analysis
- Strategy and management style
- Knowledge and Information Sources
- Information Technologies use
- Social Network Analyses
- Corporate Culture and Staff fit
- Motives and salaries

## 2.5 KNOWLEDGE AUDIT REPORT

---

### 2.5.1 Context

The organization implemented successfully the Knowledge Audit methodology using the most appropriate tools for gaining maximum inputs. The process should be finalized with a document describing the results and providing inputs to a further decision-making process linked to Knowledge Management strategy, systems, tools and instruments, improvements, etc. The research and analysis outcomes are needed for Knowledge Management evaluation and progress measurement, as well as for determining a Knowledge Management roadmap and further steps to take use of Knowledge Management enablers and overcome potential barriers [1].

### 2.4.2 Problem

**How do you compose a good Knowledge Audit Report?**

### 2.4.3 Forces

The Knowledge Audit Report role is to present the final outputs of the Knowledge Audit process and to address the next steps for Knowledge Management implementation. The main challenge of preparing this report is that it is a complex document, proposing a roadmap and Knowledge Management action plan.

**The Knowledge Audit Report preparation** – The Knowledge Audit Report should overview the main outcomes of the Knowledge Audit process. A complete, useful and focused on the company needs Knowledge Audit Report should include multiple sources of information about the organization and its knowledge assets, analyzed in a proper and detailed manner. It must examine, analyze, assess, verify, validate, review and report the findings about the current state, but also provide recommendations for future steps for developing new knowledge assets in the organization [9].

**The Knowledge Audit Report presentation** – The Knowledge Audit Report should be properly presented, discussed and accepted in the organization. Its real value is not the written document, but the process of creating it, discussing it and gaining deeper understanding about the existing knowledge in the company and necessary for its survival.

### 2.4.4 Solution

**The good Knowledge Audit Report should put emphasis on explicit and tacit dimensions of knowledge, including internal and external factors for knowledge development.**

The Knowledge Audit Report starts with in-house knowledge overview and general information audit, including knowledge resources, people, key organizational knowledge assets – patents, trademarks, experts; then business processes (innovations, learning, sharing) and knowledge flows, IT systems, social aspects and culture. The second part comprises tacit dimensions of company knowledge or assessment of individual and group knowledge. Finally, analyses of the company environment provides a short description of the industry knowledge (global aspects, demand and supply curves, fluctuations, main players), Porter analyses (for knowledge possessed and acquired from customers, partners, suppliers, competitors and substitutes), research achievements (university and research centers, key achievements, key researchers working in the area, recent inventions and publications, conferences), technology level (technologies in the sector, trade fairs and events, publications, PR).

The Knowledge Audit Report finally identifies the organization's readiness to adopt a Knowledge Management initiative – pointing out the Knowledge Management enabling

factors and persons, what are potential barriers, suitable Knowledge Management instruments and initiatives to start with, and finally – implementation roadmap.

The Knowledge Audit is presented usually as:

- **Printed hand-out Report.** It can be used as a reference document and for internal communication. It is advisable the Knowledge Audit Report to be concise, well balancing the content, including multiple charts, figures and images, while applying user-oriented terminology and design.
- **Electronic version of the Knowledge Audit Report.** It could be published on the company website, where a public discussion could be organized, reflecting the major issues and outcomes of the Knowledge Audit process.
- **Knowledge Audit Workshop** – Usually, the Knowledge Audit team prepares a short workshop, where it reports the Knowledge Audit outcomes. The resulting discussions and feedback could be taken into consideration when Knowledge Management roadmap and Knowledge Management Plan are assessed.

### 2.3.5 Consequences

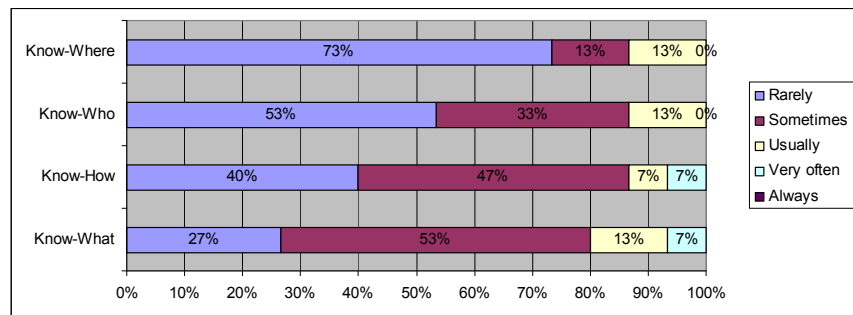
The Knowledge Audit Report outlines the state-of-the-art and the Knowledge Management needs and gaps and on basis of the information collected, identifies and assesses the knowledge strengths and weaknesses and knowledge opportunities and threats. The Knowledge Audit Report provides sound recommendations for further Knowledge Management initiatives assessing the current state-of-the-art and scenarios for future development. It ensures better understanding of Knowledge Management strategy and investments needs. The Knowledge Audit Report plays a role for achieving a better success rate of any Knowledge Management program, saving unnecessary efforts, resources and time and customizing the Knowledge Management approach to the concrete needs of the organization.

### 3. Examples

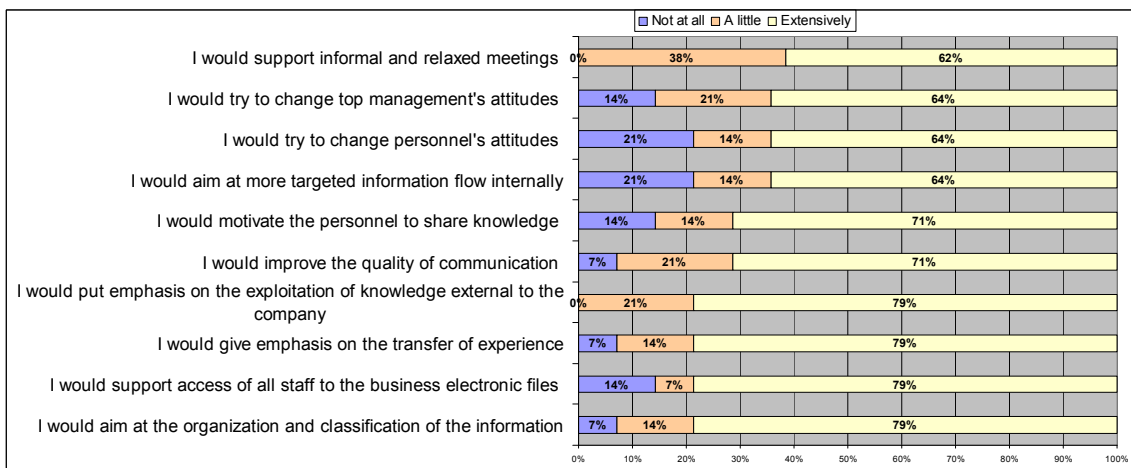
A knowledge Audit was carried out in 14 organisations in Europe within the Leonardo da Vinci project TRAINMOR KNOWMORE. One of the pilot studies was a knowledge audit carried out in the Centre of Information Society Technology (CIST) – a dynamic research unit of Sofia University [2].

In the Knowledge Audit survey of TRAINMOR-KNOWMORE took part 15 persons from CIST permanent staff (almost 90% of the permanent employees). Multiple relationships and attitudes of the employees, relevant to Knowledge Management were identified.

Some of the main findings are related to time spending of the employees for knowledge gathering, information flows in CIST, organizational climate, knowledge-related problems (Fig. 3) and Knowledge Management activities needs (Fig. 4).



**Figure 3:** Frequency of knowledge-related problems

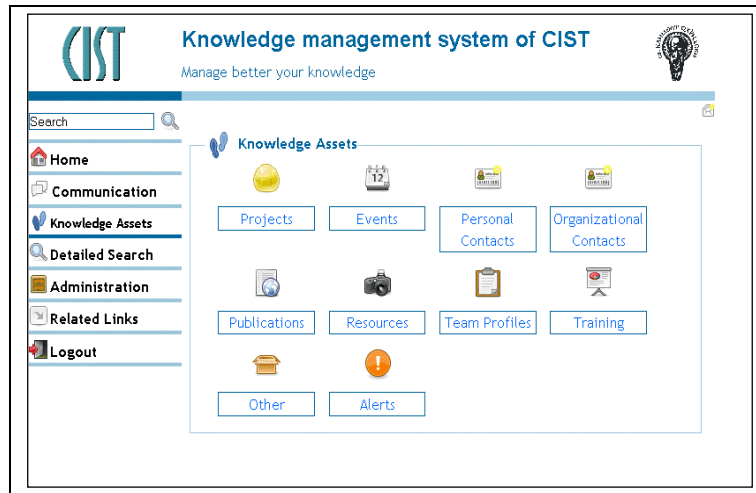


**Figure 4:** Knowledge Management activities that have to be performed in CIST, according to staff

Following the Knowledge Audit of CIST was proposed an action plan, and on this base the main characteristics of a Knowledge Management System that will enhance Knowledge Management processes in CIST were identified. Subsequently, a prototype of the Knowledge Management system is designed and developed, responding to the identified problem areas (Fig. 5). The Knowledge Management system is designed as a single entry-point to the knowledge assets and resources of CIST, accessible anywhere and anytime via an Internet – based Knowledge Management portal. The main functionalities of the



knowledge portal include the following principles – all information is uploaded in standardized templates, which include information about the context, purpose and metadata for any document are created. All information uploaded on the portal can include personal comments (tacit knowledge); can be linked to other files or templates, can be described by keywords, etc.



**Figure 5:** Knowledge Management system of CIST – Section knowledge assets

#### Related work and sources

TRAINMOR KNOWMORE Consortium, <http://www.trainmor-knowmore.eu/>

#### 4. Conclusion

The Knowledge Audit is a critical step for Knowledge Management in organizations, as it supports the initial Knowledge Management implementation, the evaluation of its results, and thus the decision making process in the organizations for making greater use of knowledge strengths and the competitive advantages of the organization [1]. In fact, a wider understanding of company interests, global situation and processes could facilitate all employees to contribute better to the competitive position of the company and the management of its knowledge. This means not only better acquiring (learning) and generating (innovating and experimenting) knowledge, sharing it (communicating) and storing it (codifying) but also better anticipating the future, and finally, better preparing for it [1].

As the Knowledge Audit represents a process which could be applied in all knowledge-based organizations, it is important to create guidelines for its proper implementation based on study of real cases. In order not to reinvent the wheel, business patterns for Knowledge Audit could be applied in organizations. The paper presents five different patterns corresponding to the main steps needed in the Knowledge Audit-implementation chain - Knowledge Audit Plan, Knowledge Audit Team, Knowledge Audit Process, Knowledge Audit Questionnaire and Knowledge Audit Report.



## History

<i>First draft</i>	<i>14 February 2009</i>
<i>Second draft</i>	<i>20 April 2009</i>
<i>Third draft</i>	<i>24 May 2009</i>
<i>Fourth draft</i>	<i>29 June 2009</i>

## Acknowledgements

The authors express their deep appreciation to Lise Hvatum for shepherd this paper to EuroPLoP 2009, and to Allan Kelly for all suggestions during the writers workshop.

The authors gratefully acknowledge the results obtained during the pilot project under the EU programme Leonardo da Vinci “TRAINMOR KNOWMORE” (2005-2008), and the support provided under the FP7 project SISTER.

## References

1. Antonova, A., E. Gourova, An extended Knowledge Audit Approach, Proc. of International Scientific Conference ‘Business Informatics’, 11-12 October 2007, pp. 151-159
2. Antonova, A., E Gourova, Knowledge Management in Universities – the CIST case, 9th European Conference on Knowledge Management Southampton Solent University, Southampton, UK, 4-5 September 2008, pp. 27-34
3. Dalkir, K., Knowledge management in theory and practice, Elsevier, Butterworth Heinemann, US, 2005
4. Hylton, A., A Knowledge Management Initiative is Unlikely to Succeed Without a Knowledge Audit, 2002, available: <http://www.annhylton.com>
5. Mertins, K., P.Heisig, J.Vorbeck, Knowledge Management – Concepts and Best Practices, Springer Verlag, Berlin-Heidelberg, 2003
6. Liebowitz, J., Knowledge management Handbook, CRC Press LLC, 1999
7. Bukowitz, W., R.Williams, The knowledge management fieldbook, Prentice Hall, 1999
8. Pfeifer, J., Sutton, R., The Knowing – Doing Gap, Harvard Business School Press, 1999
9. Hylton, A., The knowledge audit is first and foremost an audit, 2004, available: <http://www.annhylton.com>
10. A Guide to the Project Management Body of Knowledge, Third Edition (PMBOK Guides) by Project Management Institute
11. Organizational Knowledge Management Handbook, TRAINMOR KNOWMORE, March 2008, available: <http://www.trainmor-knowmore.eu>
12. Galic M. et. all, Academic Edition: Applying Patterns Approaches Patterns for e-business Series, IBM Redbooks publication, 2007, SG24-7466-00.
13. Kelly A., Business Strategy Design Patterns, The Porter Patterns, 2005, available: <http://www.allankelly.net>
14. Kelly A., Business Strategy Patterns for the Innovative Company, The Porter Patterns, 2005, available: <http://www.allankelly.net>
15. Schubert P., R.Wolfe, eXperience-Methodik zur Dokumentation von Fallstudien, in: Wolfe, R., P.Schubert, Wettbewerbsvorteile in der Kundenbeziehungen durch Business Software, Munchen: Carl Hanser Verlag, 2008, pp. 17-24

# Applying Distributed Development Patterns

## Stories and Pattern Sequences

Lise B. Hvatum

The experience reflected in these patterns comes from years of working for an international technology company with product development centers in Europe, North America and Asia, and through interaction with people from other companies practicing distributed development. The patterns were initially built on observing internally established practices, and later influenced by learning through reading, conference participation, and discussion.

Despite the effort of capturing over thirty patterns to create a solid foundation of knowledge, it is not enough to make the material really useful to an organization. The manager or team member of a distributed team faced with all these knowledge pieces must feel like a kid opening a large box of Lego just to find that the instruction manual is missing. There are lots of parts, but no beginning or end. It may look overwhelming, and if he or she needs all the parts depends on the complexity of the system they are building.

This paper follows the stories of three projects as a way to illustrate the use of the patterns in the context of a small, medium and large project. Alternatives and possibilities are explored, while making it clear what needs to be prioritized, or even practiced to the level of “must-do”.

### Background

A distributed team is not a new invention. People have collaborated on research and product development projects internationally for decades if not more. Several years ago I worked as the manager for projects split between Norway and England, without considering that to be a problem. Traveling was easy and inexpensive. The project members would visit on a regular basis, and communicate frequently by phone and e-mail.

Then a few years ago I got involved in projects running between the US and Asia. What had been small, manageable issues now escalated to become major obstacles. The Asian location hired several young developers with little knowledge of the business domain. In-depth business and technology expertise was mainly with people in the US locations. Cultural differences as well as language problems were hurting communication. The time zones allowed no overlapping work time, which meant full day delays in exchanging e-mails, and caused very early/very late presence for video conferences and phone conversations. Travel was time consuming and expensive.

Luckily the people involved were enthusiastic and wanted to make it work. Although stumbling from one problem to the next, the teams eventually found ways to collaborate and be productive. As managers and key engineers moved on to other activities, their experience was captured to benefit future teams.

## Introducing the Patterns

The patterns themselves are not included in this paper as it would make it far too extensive. Most of the patterns are presented in papers at PLoP conferences [EuroPloP 2004, PLoP 2004, EuroPloP 2005], and they are all represented by thumbnails in the appendix. Here is an overview to aid the readability of the paper. After reading the following two pages, the reader should have enough knowledge about the patterns to be able to appreciate the stories and pattern sequences following this introduction.

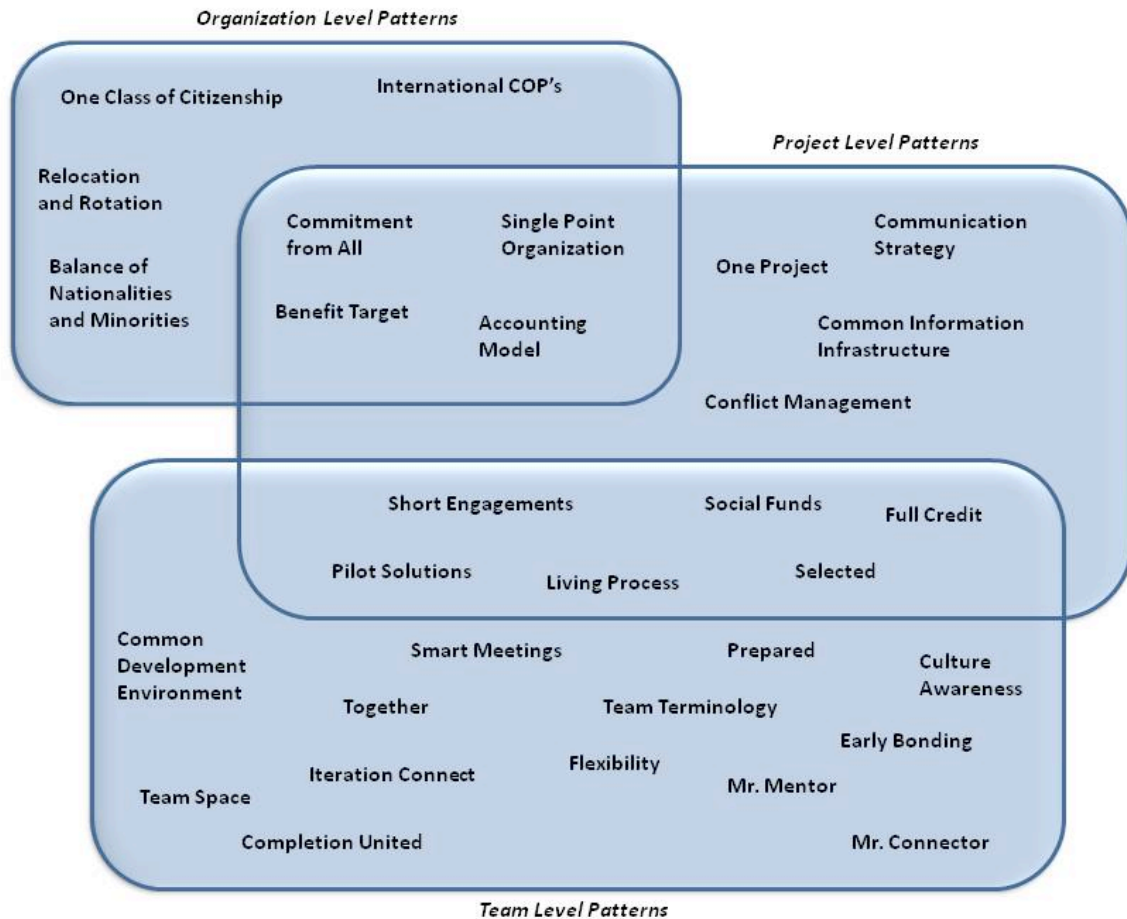


Figure 1: Distributed Teams Patterns Overview

The patterns are sorted into three overlapping levels. The first set of patterns applies for the whole organization. Some are independent of the project: **One Class of Citizenship**, **Relocation and Rotation**, **Balance of Nationalities and Minorities**, and **International COP's**. These patterns focus on creating an organizational culture that will enable the organization to succeed with distributed teams.

Other top-level patterns are needed to create an organizational structure that will support a distributed project: **Commitment from All**, **Benefit Target**, **Single Point Organization**, and **Accounting Model**. The purpose of each of these patterns is given in figure 2.

*Organization Level Patterns*



Figure 2 Patterns applied for the overall organization

Additional patterns are applied on the particular distributed project: **One Project, Communication Strategy, Common Information Infrastructure, and Conflict Management** to set up the project properly for a distributed setting. These patterns are typically applied by the Project Manager.

*Project Level Patterns*

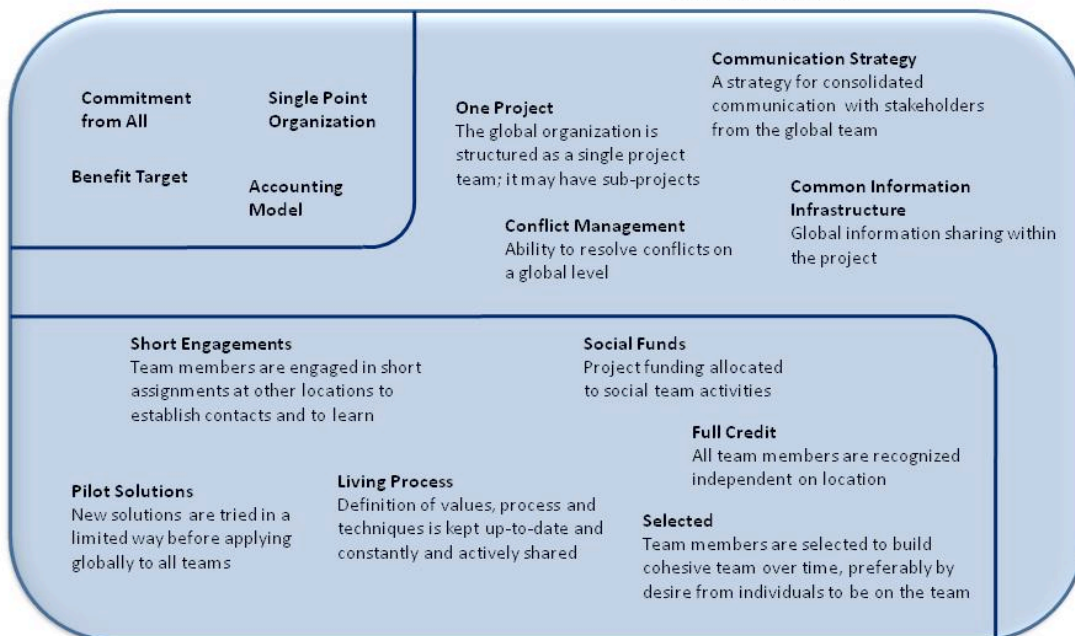


Figure 3 Patterns applied for the project

Finally there are a large number of patterns that are applied by the team to enable the team for distributed development. The more control the team is taking in adapting and

implementing these practices the better is the chance of successful implementation. The patterns on this level are practices that apply on a team level only, although some of them require financial and organizational acceptance from the organization.

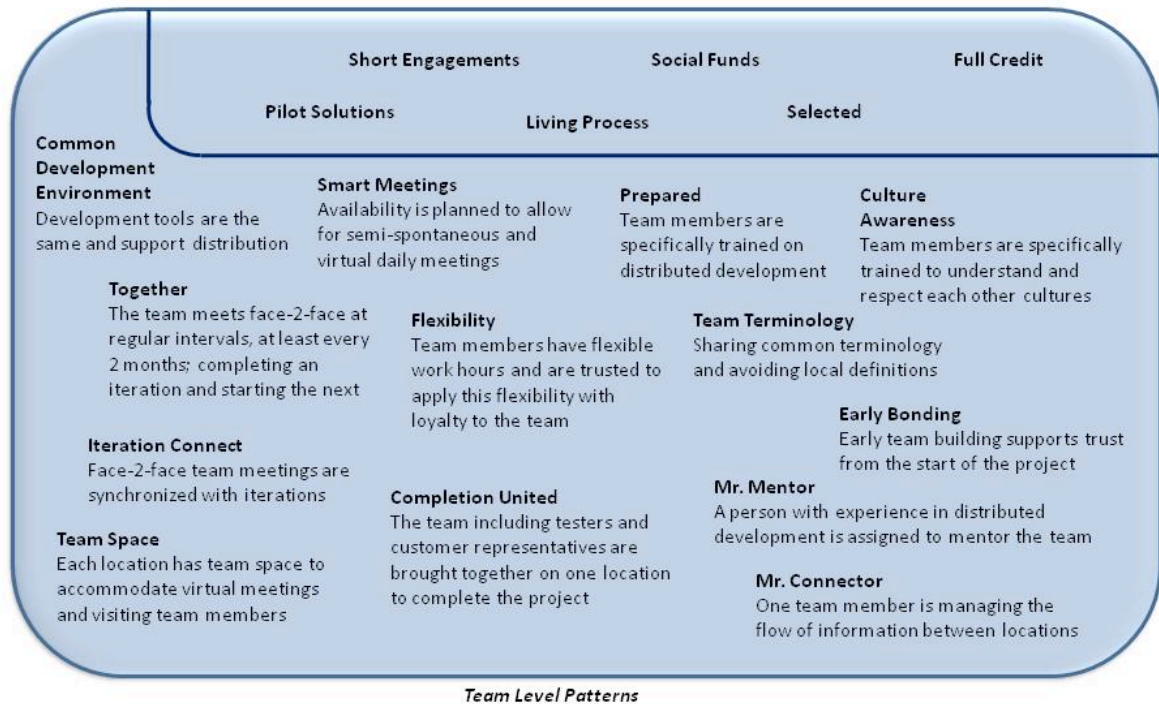


Figure 3 Patterns applied internally by the team

## Introducing the Projects

In the following sections, we will look at projects requiring an increasing level of formality and organizational structure. The project stories build on real experience and reflect real projects. For confidentiality reasons actual names and project details have been somewhat modified, although in a way that should not affect the understanding of why and how a pattern is applied.

The first project is a small activity running for a few months, and the focus of the story is team communication and collaboration. The second team is larger and requires some support from the organization. The third project is large and involves multiple locations resulting in the need to establish practices and workflows far outside the project itself. The essential characteristics for each project in relation to distributed teams are summarized in the table below:

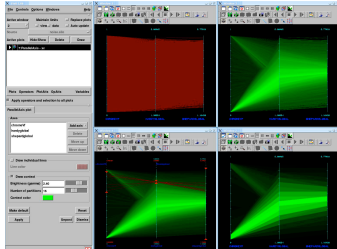
Project	Visualization Tool	Asset Management System	Control System
<b>Size</b>	Small	Medium	Large
<b>Budget (USD)</b>	0.3M	5.9M	32M
<b>Effort</b>	1.3 my/7 people	27 my/27 people	120 my/55 people
<b># Locations</b>	3 2 in China, 1 in the US	3 US, France, India	6 2 in US, UK, France, Italy, India
<b>Time Difference</b>	China: Same time zone China – US: 13 hours	US – France: 7 hours US – India: 11 hours	US – UK: 6 hours US – France/Italy: 7 hours US – India: 11 hours
<b>Complexity</b>	Low	Medium	High
<b>Innovation</b>	No	No	Yes (new)

Throughout the project stories the following formatting and structure is used:

- 1) Pattern names are written in **bold**.
- 2) Throughout the stories text in *italics* headed *Exploration* explain and suggest alternative implementations to better understand the use of the patterns.
- 3) Side effects and negative results of applying the patterns are included in the exploratory text under *Issues*.
- 4) A few external patterns are mentioned in the exploratory text. These are represented with underscored font. They are listed with references on page 23.

At the end of the paper there is a short discussion of other patterns and pattern languages that complement the patterns in this paper. The reader looking for more detail will benefit from reading the actual patterns (see references).





Picture Source: Google Images  
(sample picture for illustration purposes)

## Project #1 – The Data Visualization Tool

Company A sells measurement data to customers. The data by itself is meaningless unless you know how to represent it graphically in a way that makes sense to analysts in the client company. So it is customary to provide a viewer tool with the data. The behavior of the viewer tool depends on the physics of the measurement which is already well known. It is developed on a software framework which is mature and stable. The developers have made several similar viewers for other measurements before. This is a low-risk project estimated to take about 3 months (15 man-months of effort) including testing, with a USD 250k budget.

The resources assigned to the project are a manager (Mark) and three developers in China (Yuan, Chen, and Feng), and a domain expert (Alfredo) and a physicist (Linda) in the US. There is also a qualification engineer in China (Li), but in a different location. They are all experienced and have worked together before.

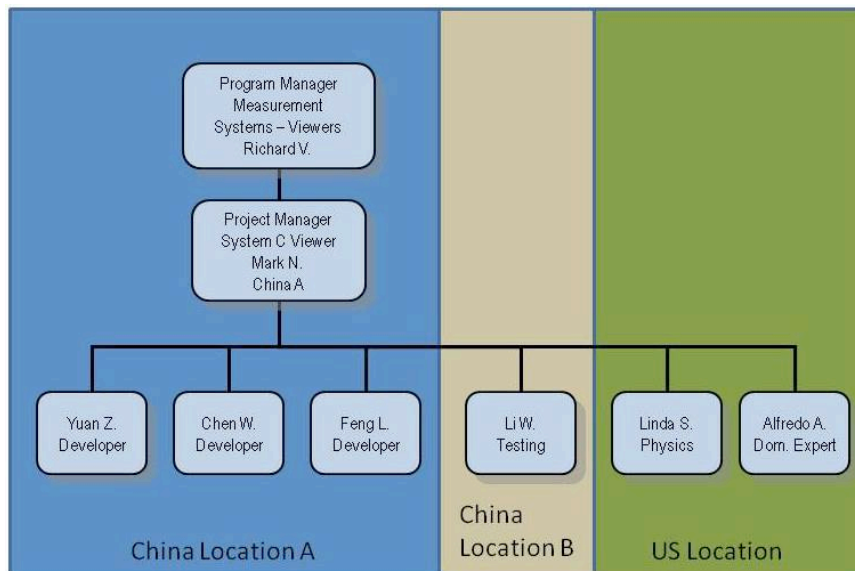


Figure 4: Team organization for the Data Visualization Tool team

The company has learned to always establish a **Single Point Organization** which establishes well-defined roles and reporting on each level of the organization to ensure clear decision making and responsibilities. It creates the new project as a part of the Chinese project portfolio under the program manager (Richard) who is in charge of a family of viewers for similar products. This is natural as this is the location of the project manager. Mark makes sure to run the development as **One Project** with shared team

objectives, and to get **Commitment from All** especially for the external resources. Alfredo is assigned full time to the project for its duration, but Linda has another project to deal with as well.

*Exploration: It is important to ensure that there is a clear, communicated and simple reporting structure for the project. In such a **Single Point Organization**, decision at each level and in each entity in the organization is sitting with a single role to remove confusion. Roles and their assigned authority are clearly defined.*

*Team focus is also made clear through applying the **One Project**, having only one manager and team world-wide with common objectives and goals. The project manager is the only interface to the customers and upper management in the company when it comes to defining the scope of the product, change management throughout development, and all other major decisions. The stakeholders can directly collaborate with team members in discussing the product (requirements discussions, demos, acceptance testing), but no decisions are made without the project manager in charge.*

*The up-front **Commitment from All** involved locations supports the project with the resources needed throughout the development. The resources are definitely people with particular skill sets and knowledge, but may include other support like facilities, access to test systems etc.*

*Issues: There is an assumption in the above exploration that the organization is structured in projects. That may not be the case for your company, and so the implementation of the above will need to take another shape. What is important to achieve is the clear structure and responsibilities for decision making in respect to the developed project, but also for the utilization of resources. In particular it is necessary that an individual engineer assigned has enough time available and that priorities (and possible priority conflicts) are taken care of by management and not left to the individual to deal with.*

Even though his team is experienced, Mark decides to fly over the two people from the US for one week so they can start the project **Together**. The team starts planning the product and creates the initial project plan. The qualification engineer Li is also coming to the development location in China for the same week so that the whole team is there. Mark enforces the **Early Bonding** by using **Social Funds** for some teambuilding. The team members already have a good level of **Culture Awareness** and are well **Prepared** through earlier experience with distributed teams, but their week together enforces this.

*Exploration: The most important time for a team to physically be **Together** is at the start of a project. Even for larger systems, a good foundation both for the system architecture and the team collaboration (trust) can be done in a two-week boot camp. The goal is to establish a common understanding of the product, and of the main principles of the system architecture. Several possibilities should be discussed, and an initial framework agreed on. This is a first achievement for the team, and by tightly sharing the evolution of this framework each individual will feel the ownership and enforce the **Early Bonding** that builds trust in the team that they will benefit from throughout the development. By including all resources in the event, they are on the same level, share the terminology and the understanding of the product, and can plan for their own tasks in the project early. Having a customer representative present as in this case is very beneficial. Many mistakes and misinterpretations of requirements can be avoided by the customer being heavily involved at this stage.*



One good effect of early collaboration is the **Early Bonding** between team members from the beginning. By observing each other during team interactions, all get a better understanding of the individuals, their communication style and their knowledge. **Social Funds** should be used to invest in good people relations through social activities. This can include team dinners, an outing to visit a local attraction, an evening at a local go-cart track, etc. Strong team interrelations and trust is invaluable on a project. Although hard to quantify directly, most problems on software projects can be traced back to communication problems within the team or with the stakeholder.

When talking with developers who have experienced multi-cultural teams, they often point out the importance of understanding the body language and communication style of other nationalities. Unless all team members are familiar with the main cultures represented it may be a good investment to send the team members to training to build up their **Culture Awareness**. The same goes for training on distributed development to ensure the team members are well **Prepared**. An investment in training the team members on issues and solutions for multi-site software development will surely pay off, although it may be hard to find this kind of training offered yet.

*Issues:* All of the above patterns require that the company is willing to do the financial investment in travelling and training. If this is not possible in your company you need to emulate the effect of together-time. You still need the effort on architecture and product vision, and a virtual workshop may be a way forward. This means that team members at each location run local workshops, with limited sessions run with the complete team using collaboration tools (Live Meeting, phone, video conferencing etc.). For team members to better know each other you could create a gallery where team members can post pictures, stories and personal statements, or even link up with tools like MySpace. The manager must make sure that people are comfortable with using these methods, and that what is posted is not offensive to any of the team members.

The team agrees on short iterations of one week. They have a **Common Development Environment** that makes their collaboration easy. A full integration test on completed functionality (Use Cases) is done by Li on every integration baseline. The team also has a collaboration space (their **Common Information Infrastructure**) where they post all their documents, but that also includes chat options and live meeting capabilities.

*Exploration:* The importance of a common development process and a common development environment is often not understood by a company. Most fundamental is the shared code base, especially if the development is truly one product and the developers need to share/modify some of the same files. Iterative development with frequent integration into a common baseline, and automated testing based on full unit test coverage is giving the team the benefits of good control of progress, and early detection of problems. This way of working really means that the team must be using the same development tools.

Common space like a wiki, SharePoint or using a collaborative development tool like VersionOne ensures that each team member always has access to the latest version of important project documents. You may also decide to share the history of your chat sessions since a discussion thread may have information valid for other team members.

*Issues:* For historic reasons, locations may be using and have experience with different development tools. The organization must be willing to invest in common tools for development and team collaboration and train people on the use of these.

During the development the team has frequent **Smart Meetings**, meetings that emulate a Daily Scrum in a distributed setting, to keep each other up to date on the progress. Attending these meetings is mandatory for all team members, but except for that time there is a lot of **Flexibility** for the individuals in work hours. Mark pays special attention to Linda to make sure her dual assignment is properly managed.

*Exploration: For a short development it may not be necessary to have more session where the whole team is on one location, but the best is if the teams can meet face-to-face about every 6 weeks. If traveling budgets are too restrictive, or the team members know each other well from earlier development activities, face-to-face meetings may be less essential. But the **Smart Meetings** emulating the daily Scrum are totally necessary, and all team members participates each time unless there is a very strong reason they cannot make it. In these meetings the team members keep each other up to date with individual status, any changes, and any dependencies they have to each other. Because meeting hours may be at odd hours for some team members (in the Data Visualization Tool project the majority of the team sits in China, and the meetings take place in the morning as they get to the office, meaning the team members in the US need to attend late in the evening local time), work days are kept flexible for the individual as long as they work enough hours and participate in any common events.*

*Issues: This loyalty must work both ways between the company and the employee. The focus must be on progress rather than work hours, and respect the **Flexibility** for each person in when they work.*

At the completion of the project Mark decides in agreement with the team to do a partial **Completion United** by bringing in the stakeholders and the qualification engineer Li only. He and Li travel to the US, where they have a weeklong workshop with stakeholders. With Linda and Alfredo as support, Li is teaching the users to use the new viewer, and she is tracking any defects and improvement requests during the day. One developer in China is on call during the US work hours to fix any major defects should they find any, but the other two are working normal hours fixing problems during regular Chinese work hours. They make sure a new baseline is built and working before the start of a new work day in the US. In this way they sort out any problems in the system within a few days. The test users from the key clients are happy with the product, and it is ready for shipment.

The team is confident that the project has fulfilled the requirements and that the new Visualization Tool is completed. The stakeholder has a couple of small requests that are sorted out in two days, and the project is at its end. Mark makes an open report back to his management pointing out the effort of the team members and crediting them with their achievement making sure that all team members get **Full Credit** for their involvement. He and Li celebrate with the US team members before leaving, and again with the China team members when back in China. He buys all a small gift to show his appreciation of their contributions.

*Exploration: The completion phase brings the extended team together for a very efficient closing of the project. This activity can be done as a combination of testing and training of the beta user community. Being **Together** for the final phase is also an opportunity to get retrospect and get closure for the team. It is important to make sure that all members of the extended team are recognized for their contributions.*

*Issues: It may be hard to get the financial support in your company to bring the team together at the end of the development. Combining it with training users may make it*

*easier, since it will be cheaper for a small development team to travel to many users than the other way around. Maybe it could even be possible to get the clients to pay for this training, but do manage the customer expectations of quality by making it clear that they are getting trained on a Beta version and they will likely find issues with this system.*

The sketch below shows the patterns sequence from the above story. If patterns are deemed necessary they are marked in **bold** with a thicker line around the shape. Note that each of these patterns can be applied independently, although they mostly benefit by being used with other patterns that they relate to in the sequence diagram (like Early Bonding and Social Funds).

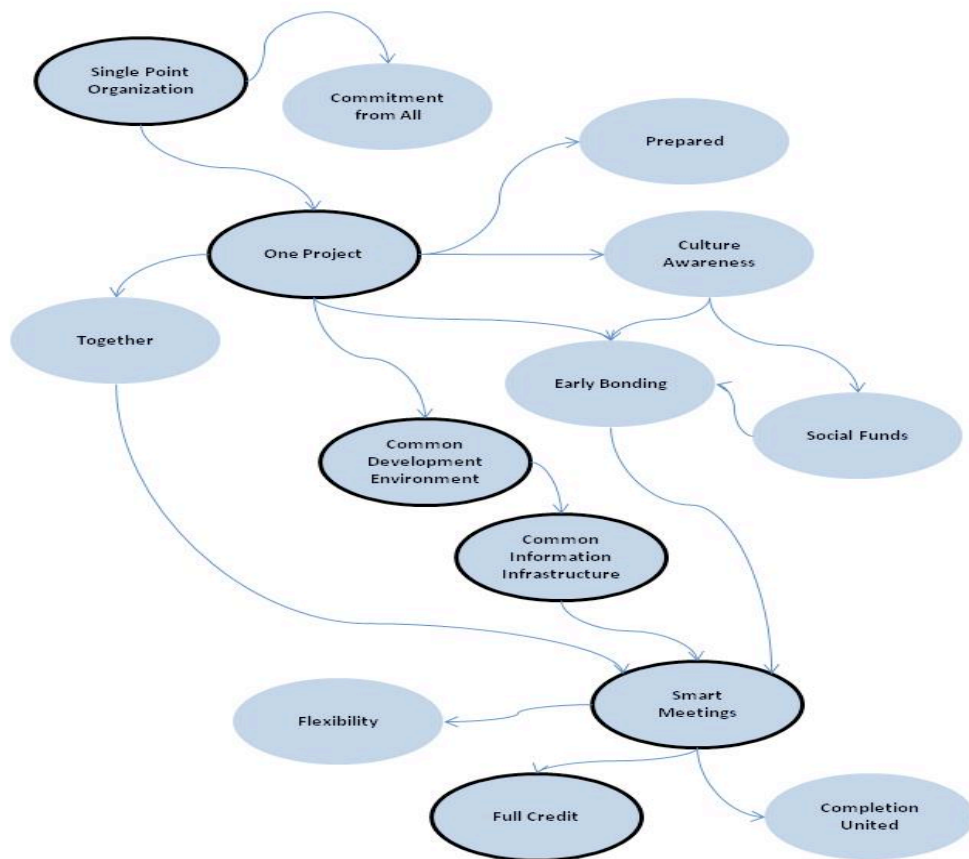
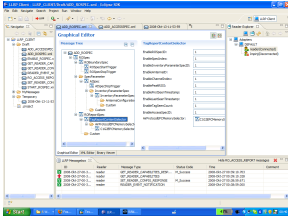


Figure 5: Patterns sequence for the Data Visualization Tool team



Picture Source: Google Images (sample picture for illustration purposes)

## Project #2 – Asset Management System Update

Company B is developing an asset management system for important client. The current system is having some severe problems caused by use of outdated 3<sup>rd</sup> party technology, as because some of the implemented business logic is unstable. new project is started to re-factor parts of the system a replace an Access database with Oracle. The project development time for this project is one year (27 man-year with a budget of USD 5.9M.

The existing team is expanded with an Oracle DBA (data base administrator) and several developers. Team members are located in the US, in France and in India. The developers in India have less experience with the technology, and are new to the company. The most experienced people (the architect and the system developers) and the project manager are located in France, while the DBA team sits in the US. The stakeholder representative is also located in the US. A testing team of three people are co-located with the application development team in India. Note that only the team with the project manager is part of the budgeted effort of 27 man-years. The other roles are part of the organization but manage/support several projects and are budgeted otherwise. The team structure looks like this:

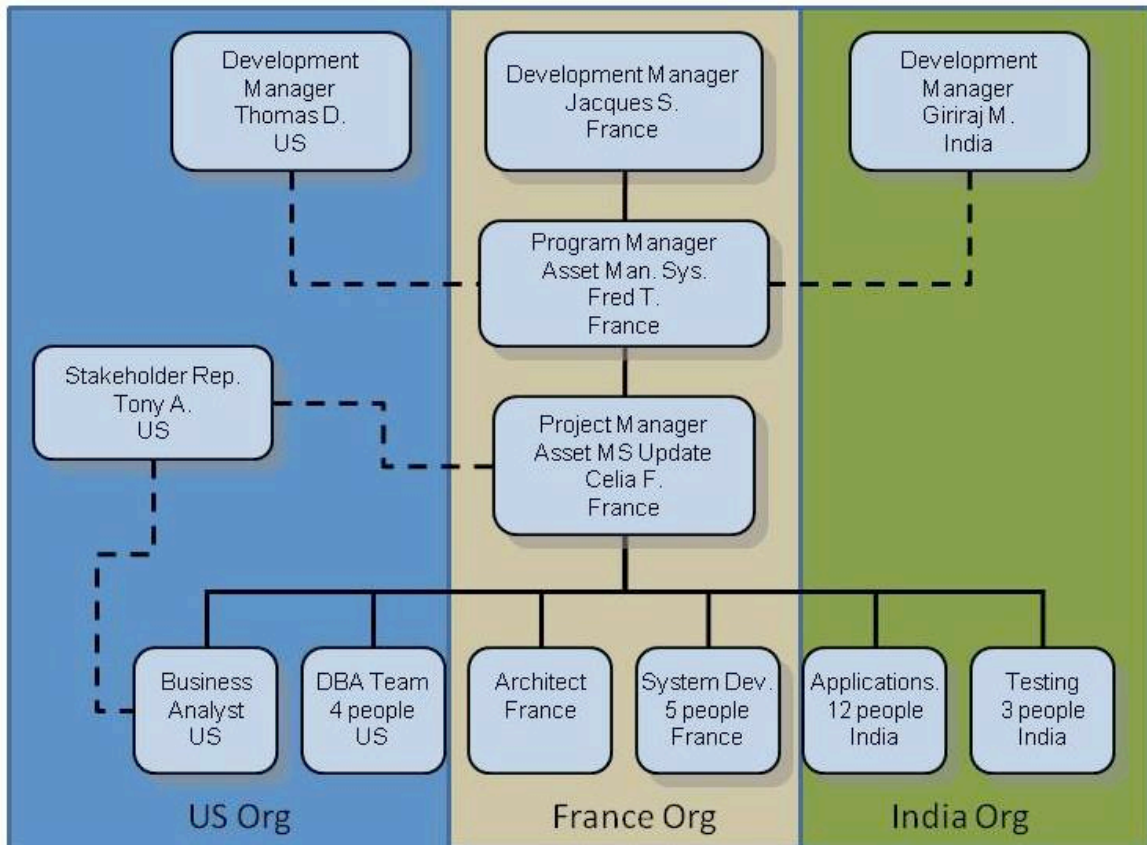


Figure 6: Team organization for the Asset Management System team

This is clearly an organization that is more challenging than Project #1, with dependencies and need for good collaboration between multiple sites both within the team and within the management levels.

Celia, the project manager, works on two levels when starting up the project. She collaborates with the involved development centers to make sure there is **One Project** organization where all team members share the team objectives with the same priority. With her own management located in France it is natural that this project is added to the French project portfolio. She also makes sure there is a **Single Point Organization** for all decision making needed in the organization to support the project, and she ensures **Commitment from All** for team allocation and local resources at each location. This includes **Team Space** to accommodate visits, and an **Accounting Model** to make sure the project can deal with budgets and financial reporting in a uniform and consolidated fashion. Finally she agrees a **Communication Strategy** with the stakeholders and the center management at the different locations to keep all informed the way they desire.

*Note! The exploration text will not repeat what has already been explored in previous project examples, but rather focus on what is introduced in each project.*

*Exploration: Creating the **Single Point Organization** is highly important to enable the operation and performance of the team. This will provide a clear direction and faster decision making as all team members are clear on who needs to be involved and how decisions are made. With multiple projects the organization needs to balance the project ownership between entities to even this out in the bigger picture, and ensure credit to all involved at project completions.*

*Accounting needs to include both what part of the budget is allocated to each location, and how local spending gets rolled up into a global project account. How to handle deviations in planned spending needs to be included, usually overspending is taken at the location that has the project in its portfolio (for the Asset Management System this would be the location in France where the Project Manager is located). The project status communicated according to the project's **Communication Strategy** will include spending and resource issues at the various locations, as well as progress on system functionality.*

*A part of the commitment from each location must be how to accommodate visiting team members, and any needs for system testing and acceptance. Local **Team Space** should be designed to accommodate visiting team members and also allow for virtual meetings (no need to find free meeting rooms elsewhere in case of urgent meetings that are not pre-planned).*

***Commitment from All** also comes in the form of supporting the project objectives, and allowing for the project team objectives to take priority over local objectives for the location. For the global team to share objectives is totally necessary for the joint focus on the **One Project**.*

*Issues: The first time a large organization is faced with the challenges of a distributed team it will most likely be a difficult and time consuming process to apply the patterns as described in the above paragraphs. The demands on the organization will challenge established and hard to change internal workflows and authority distributions. With multiple locations and organizational entities involved this can be a political challenge as*

*the supporting centers may feel that they do the work but lose the influence and the credit at delivery.*

Celia focuses on her team from the start. She interviews her assigned team members and decides to substitute two people: one because he has problems with travelling and flexible working hours, the other because she seems to be overly negative about working across locations. This way Celia starts with a team where the members are **Selected** to increase the probability of a well-functioning team. Since distributed development is new to almost all her team members, Celia decides to ensure that her team is well **Prepared** by sending them to an intensive training program on distributed development, as well as an internally developed session on **Culture Awareness** to improve respect and team communication.

*Exploration: With the extra challenges a distributed team has, it is important that all the team members are positive to this way of working, and that they are given all the support possible to succeed. A well-known pattern in agile development is Self-selecting Teams where the team shares common interests and addition of new team members is a team and not a management decision. This pattern applies well for distributed development with the high level of commitment and team loyalty needed from every team member.*

*Issues: Even with all these preparations it is not a given that all team members will function well together, and Celia needs to keep an eye on team chemistry as well.*

In the initial phase, the team decides to use SharePoint for their collaboration and to store their documents, serving as their **Common Information Infrastructure**. They also select tools to be part of their **Common Development Environment**: source control system, requirements management tools, IDE, issue tracking system etc. There are some discussions and disagreement about some tools because of previous experience, but Celia makes sure that the team reaches an agreement in a friendly way.

*Exploration: This is another area where the **Commitment From All** must apply. License costs will likely be taken by the project itself, but local resources are needed to ensure that the development tools are installed and operating locally.*

*Issues: In addition to the tools, the project team needs to have clear rules for how to manage and use the tools. This will include how requirements are managed (who can add new requirements, how are requirements traced in deliverables, who is signing off that a requirement is fully implemented etc.), how documents are managed (always check out from the repository before modifications, who can modify what), and the process around accepting new code into the main developer baseline. As an alternative to SharePoint the project documents can be stored in the source control system using the same check-out/check-in mechanism, and with for instance a wiki page as portal into the documents for easy access/overview.*

Just like Mark, Celia kicks off the project with their first **Together** event, and combines it with team activities for **Early Bonding** spending some of the teams **Social Funds**. Since they spend 10 days together near Paris it is not hard to find social activities, including a trip to Château de Versailles in the weekend. Celia is creating a very special memory for the team by taking them to a beautiful historic location, and of course including a good dinner with lots of conversation. It will be hard not to create friendships during this week for the team.

*Exploration: Some of the time the team is **Together** should be spent on pure social activity as this gives team members time to get to know each other personally, creating bonds*

*across locations that will strengthen collaboration for later. With multiple locations and team members visiting locations in some kind of rotational order, there is opportunity for each hosting part of the team to expose visiting team members to some of their local culture.*

*Issues: Social activities must be planned with respect to other team members. Bull fighting would be a valid example of an activity that people may find it difficult to enjoy.*

The team decides to have 3 week iteration cycles, and get **Together** every 3<sup>rd</sup> cycle (every 9 weeks) and so to do a 5 day **Iteration Connect** every 3<sup>rd</sup> iteration. Again the team has **Smart Meetings** during the 8 weeks that they work at their respective locations, and remember to apply **Flexibility** in the work style.

*Exploration: Doing the end of an iteration and the planning of the next when the team meets face-to-face has turned out to be a practice that teams find to be very powerful. This is the time of the project where the communication is most intense. The team will run the system together and align their understanding of the desired operation, as well as of how to proceed and what changes may be needed for the underlying architecture. It can be a good idea to walk through all Use Cases (or other requirement representation) for the next iteration together as a team to make sure there are no unclear points and that all team members understand the requirement details the same way.*

*Issues: Meeting physically with the whole team is usually not possible for each iteration (unless iterations are made long which is generally a bad idea).*

A few weeks into the project some tension is starting to build in the team. The architect is upset because he feels that some of the team members in India are ignoring his directions on the architecture. The DBA team in the US seems to be out of sync with the development team in France, and these teams are accusing each other for delays in the last iteration. It becomes clear to Celia that what is going on needs more specific attention than what can be done in the **Smart Meetings**, and it is still more than 5 weeks until the team is due for another **Together** session. As she is frustrated as well, she decides to bring in a facilitator and another senior project manager to discuss with team members and analyze the problems.

After a couple of days interviewing various team members the facilitator and project manager have a pretty good understanding of the problems, which all have their root cause in communications problems. Since it is not possible for this large team to meet more often, they decide for other means to improve communication within the team. Celia manages to transfer an experienced colleague (Tom) to India for the duration of the project who takes on the role of **Mr. Mentor** for the Indian team members. Since Tom also has experience with the technology used for the development he becomes a core asset for the team. Celia also assigns somebody at each location in the role of **Mr. Connector** to reduce the risk of miscommunication between the locations. Finally the team decides to establish the set-up with the external facilitator and second project manager as a permanent **Conflict Management** solution; each time team members are frustrated they know they can contact this “crisis team” for help. The next time the team is together Celia makes sure to spend some of the **Social Funds** for the team to have fun together and get over any personal conflicts.

*Exploration: It is unrealistic to think that a team of this size even without being distributed will function without any conflict for several months, especially as the pressure to deliver*

increases. Using outside resources to help resolve differences ensures an impartial approach, and having an established procedure will likely mean that conflicts are managed earlier and before they get time to deepen. To have a **Mr. Mentor** involved who has experience with distributed development and can anticipate problems and who understands the team members frustrations is invaluable, and this role will pay off for the team from the beginning.

Very often conflicts are caused by misunderstandings and different perceptions of the system to be built. By identifying individuals that serve as the knowledge hubs locally any team member knows where to go with questions. **Mr. Connector** does not necessarily know the answer to every question, but he or she must know how to get to the answer within reasonable time. These roles at the various locations must keep in close contact with each other, and update each other whenever a local question is resolved. It would be beneficial to use a tool to capture these issues, as many questions may be valid for other developers as well (FAQ database, accessible and searchable by all team members).

*Issues:* For small teams it may be hard to find the right people for the **Mr. Mentor** or **Mr. Connector** assignments. It may be possible to get help from other people in the locations, for instance having multiple teams sharing the resource for these roles.

Finally after 13 months, the full team and the stakeholder representatives get together for a final **Completion United** event scheduled to last 4 weeks. Extensive acceptance testing is done, and although the event has to be extended for additional 2 days, the project is successfully completed and delivered. The final evening, the full team goes out to celebrate; spending what is left of their **Social Funds** and some more. Celia makes sure her team members get **Full Credit** for their work on the project. She extends her thanks to all locations involved in the project to enable local management to take credit for the contribution of the location.

*Exploration:* Bringing the team and the stakeholders **Together** for the final tuning is usually very efficient for a project. Preferably the testing activity should be done in a team room with continuous communication between testers, developers and stakeholders. The team can use Information Radiators<sup>[18]</sup> on the walls of the team room to show the list of bugs and needed modifications, plus the status on correcting these. For distributed teams, it is important to avoid the delays that would normally arise if the testing and customer input is at one location and corrections have to be done by developers at other locations. Valuable time will be lost in misinterpreting customer feedback when time zone differences mean that the attempted correction is only checked by the customer the next day.

As for any other project, to celebrate success (even if the project was slightly delayed) and to give credit to everybody involved is important. For a distributed team of this size there have been many people involved in addition to the core team, and a good project manager will make sure the local management of these people are informed of their contribution (not least to ensure support on future projects). One should remember the support of functions like accounting, personnel, local administrators etc.

*Issues:* If the team cannot be physically at one location for the final work, they should try to emulate the **Completion United** as much as possible by using collaboration tools and flexible work hours (so they are actually working simultaneously for this period even if it means working nights at some locations).



Below is a sketch of the patterns sequence from this story. If patterns are deemed necessary they are marked in bold with a thicker line around the shape. Note that each of these patterns can be applied independently, although they mostly benefit by being used with other patterns that they relate to in the sequence diagram (like **Early Bonding** and **Social Funds**).

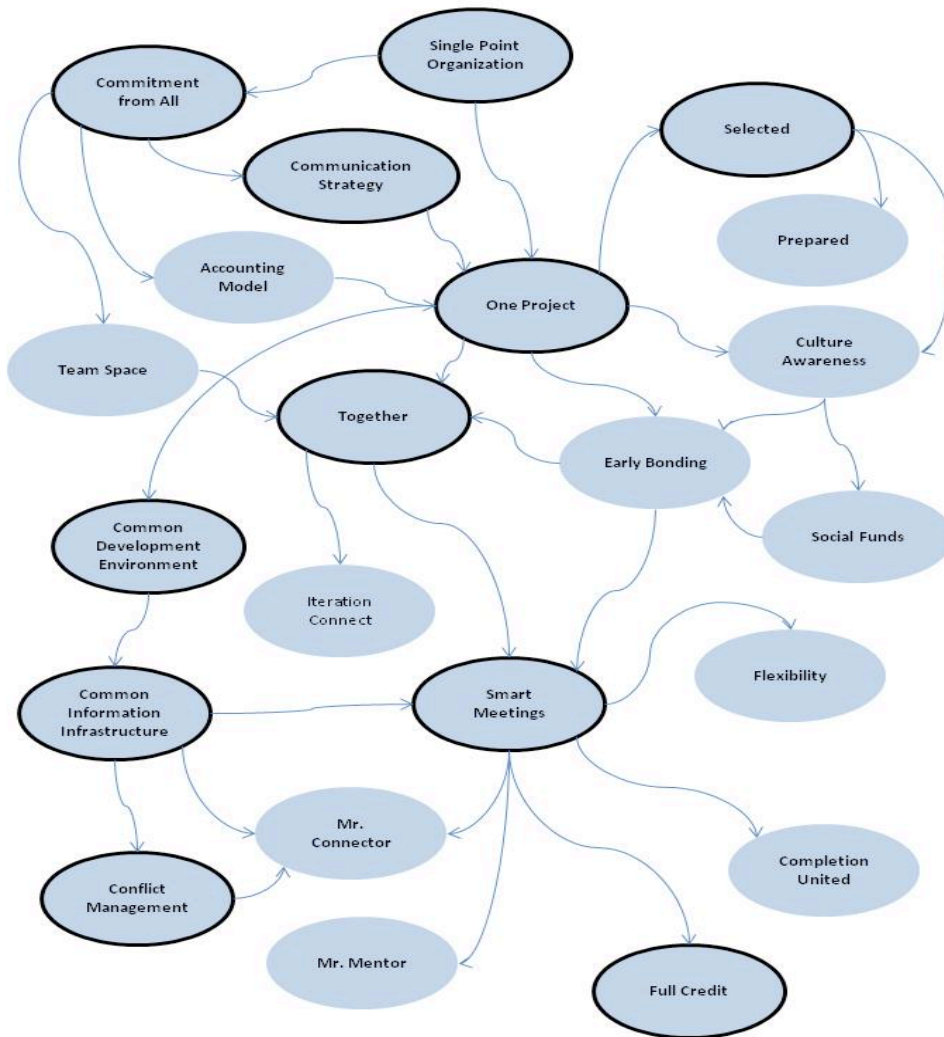


Figure 7: Patterns sequence for the Asset Management System team



Picture Source: Google Images  
(sample picture for illustration purposes)

### Project #3 – New Control & Monitoring System

Company C is developing a major new control monitoring system for internal use. The project has about 120 staff years and a budget of USD 32M. It is distributed on 6 locations from the US to India, and is expected to take 3 – 4 years to complete. It is completely new development and although there are senior team members and managers with experience from earlier generation systems (i.e. they understand the problem domain well), the technology chosen is new to most of the team members. This is clearly a major undertaking for the company, and the success of the development is core to the survival of the company.

An undertaking of this size is really more of a program than a project, and in this case it is split in a number of sub-projects each with a project manager and an architect. A core team is developing the system framework, another team the graphics, a third team works on the data management part, and several teams are working on control and monitoring applications. The overall organization is shown below:

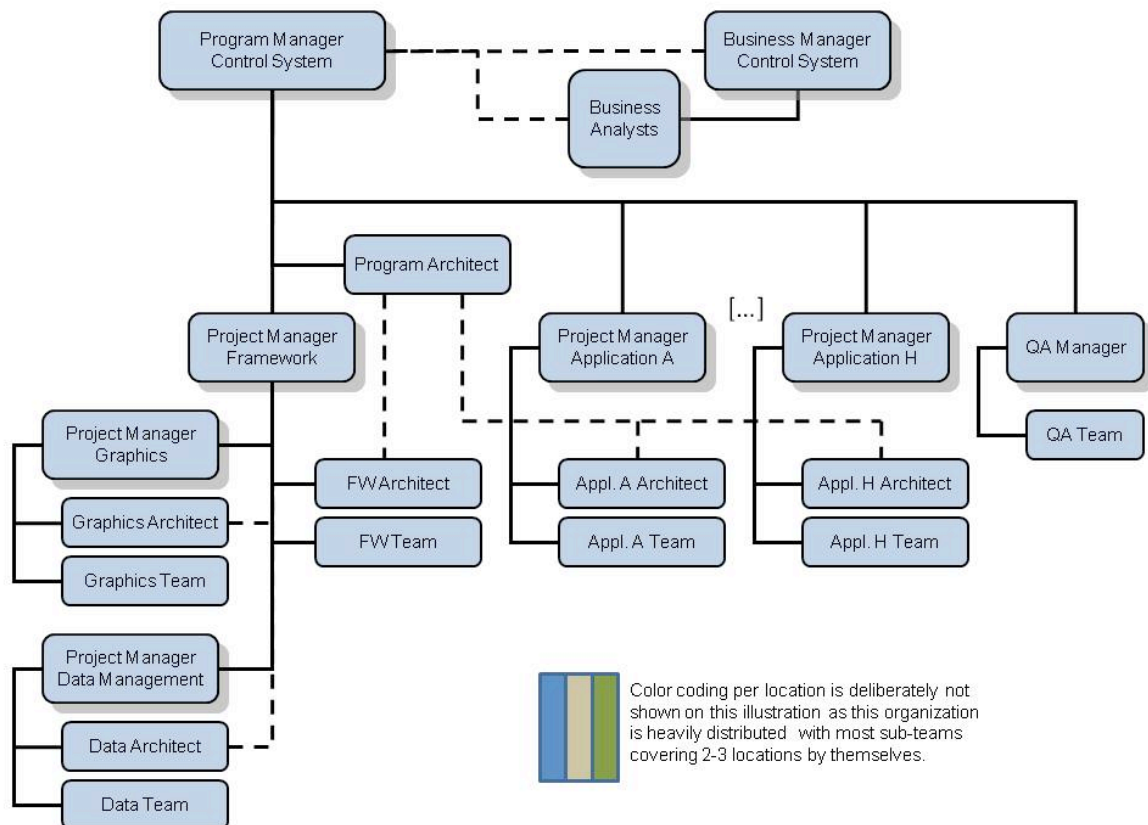


Figure 8: Team organization for the Control & Monitoring System team

Each of the sub-projects may again be a distributed team similar to the stories in project#1 or #2 (depending on its size) and the patterns sequences from these stories would apply. In

this case the focus is on managing the large team, and so the individual team practices are not repeated here.

The development of this large system (about 3M lines of code) is still organized as **One Project** (or program) with high-level objectives shared by all. On the sub-project level this is broken down into more detailed objectives for the individual project manager or developer. In this case the **Commitment From All** is coming down from the CEO level as the future of the company depends on this project succeeding. A multi-level **Single Point Organization** is defined as is seen in the organization chart above. Although there are functional reporting lines as well, there is never more than one direct reporting line for any individual, and the authority is clearly defined along this.

*Exploration: For a complex organization like this it is even more important to pay close attention to a clear definition of roles and responsibilities, communication of shared goals, and distribution of authority for decision-making. Some of the patterns mentioned in the comments section at the end of this paper aim at organizing the work with the geographical organization in mind. Conway's Law, Divide and Conquer, Loose Interfaces and Organization Follows Location are all patterns that will further improve the work situation for a distributed team and (most likely) lead to better performance and a higher quality architecture.*

This organization decided to go beyond having a defined **Communication Strategy** by introducing dedicated resources assigned to facilitate frequent communication between the different entities (information champions). These resources support the program manager and the various project managers in their effort to keep the development synchronized. As part of this communication there is a published **Team Terminology** to ensure that the problem domain terms are well understood.

*Exploration: Recognizing the importance of knowledge and communication within the team from the start, the Program Manager actually decided to invest in having a few individuals with team communication as their primary focus. Information was managed on wiki's, through small newsletters, and by presentations. Unfortunately, this team missed an opportunity in not providing a mechanism for bottom-up information from individuals. They focused too much on the leaders need to inform the troops, and lost some of the knowledge from individuals to managers that could make their decisions even better. These information champions had some collaboration with the **Mr. Connector** roles, but their coverage tends to be more on project status, requirements/functionality status and feedback from testing and stakeholders, while **Mr. Connector** focus is more on the technical aspects of the system.*

*The emphasis on frequent communication through use of information champions and the **Mr. Connector** people was not only needed to keep the development team focused, but it was an important means to manage stakeholder expectations. Major delays and difficulties were communicated on a regular basis to the stakeholders, and this early notification meant that there were few bad surprises at the later part of the project. Delays were agreed when they actually happened, rather than optimistically believing that the development could catch up later. This is not to be interpreted as the stakeholders being happy about delays, as they were not... but at least they got to know early and could plan accordingly.*

A **Living Process** is evolving and discussed frequently, and kept documented in the **Common Information Infrastructure**. By re-visiting/updating the core principles of the

development process every iteration, all team members keep their understanding of the collaboration up to date as well. The fundament of the development is agile on the level of the individual teams, while for the program level it is rather the Unified Process<sup>[19]</sup> flavored with a strong focus on iterations and frequent integration into a common and verified baseline.

As can be seen from the patterns diagram below, most of the team patterns related to development process are still used on the program level, although on a sub-system level. It is not possible to bring the whole team **Together**, but representatives from each sub-system will meet at major integration points and run a meta-level iteration transition, including demonstrations of completed functionality to the stakeholders.

*Exploration: Running an iterative process with major iterations every month on the system level, and additional smaller iterations within the teams turned out to work really well. A major challenge with a large and distributed team is that sub-teams deviate over time from each other, so forcing the system together on a monthly basis avoided major rework and delays.*

Another part of team communication is through the levels of assigned architects. They are supported in their effort by the **Common Information Infrastructure** allowing all team members to keep up to date across locations as the architecture is evolving; and not the least through the **Common Development Environment** where frequent code integration is supported. The system architecture is a constant focus. The initial architecture was used to structure the development assigned to each sub-project team (by applying Conway's Law; each part of the system is assigned to a project team). Interfaces between the system components are well-defined and any changes must be approved by the team of architects.

*Exploration: The role of the architects and the system architecture cannot be underestimated in distributed development. An architecture with clear interfaces and few interdependencies lends itself more easily to distributed teams, and the development team assignment should map to clearly bounded system modules, which again reduces the communication need between sub-teams.*

Going into details on what happened on this large project would make this paper too long. The core categories of challenges experienced throughout the development were:

- Architecture and Design consistency – keeping the architecture and design quality throughout the long development time, and communicating enough details to all developers for them to understand the framework they were developing within, and finally making sure code was not breaking architecture and design principles. This was an ongoing battle for the architects, but their presence combined with frequent integration and good communication (**Together** and **Smart Meetings**) through the **Common Information Infrastructure** and the **Common Development Environment** contributed to keeping these problems under control.
- Requirements to framework – as application work increased it became harder to prioritize and complete functionality in the framework needed to support the application. Framework changes led to needing to change applications that were believed to be completed. Eventually a formal process was put in place as part of the **Living Process** to manage interdependencies in the development which helped to decrease the frustration and reduce delays caused by rework or waiting for framework functionality.

- Stakeholder Management – not least because of the importance of this project for the company, stakeholder interest and expectations were very high, and the team had to incorporate frequent visits from upper management, project audits, presentations etc. The **Communication Strategy** and the information champions were of great help, but it still meant that the Program Manager and his staff spent a lot of their time managing out, and gave them less time to focus on the project.
- Knowledge Transfer – people coming on during the project who needed training in how to develop on this framework, tasks being redistributed, people leaving for multiple reasons, all meant that there was an ongoing need for internal training and supervision. The **Mr. Connector** and **Mr. Mentor** roles were very valuable in this context, as was the **Common Information Infrastructure** where new team members could find historic/background information as well as easy access to current status and plans and architectural modifications to understand their task in relation to the overall system.

To some surprise very few people issues surfaced on this project. It may be that the diversity and heterogeneity of the team made each team member more open to other cultures and ways of working. It seemed like the bigger challenges of delivering a system of this size contributed to bond the team members very strongly against the “common enemy” of delivering on time.

When this project finally completed after more than 5 years and a budget overrun of 20% it was still seen as a huge success by Company C. Certainly some people had left the company during this time, and a few not voluntarily. There had been difficult periods where everybody had severe doubts if this project would succeed, and the initial product scope had been somewhat reduced. But the clients were positive after the first teething problems, and the company was still solid. In the retrospective the practices around team communication and trust-building stood out as practices never to forget.

A possible patterns sequence for this story is:

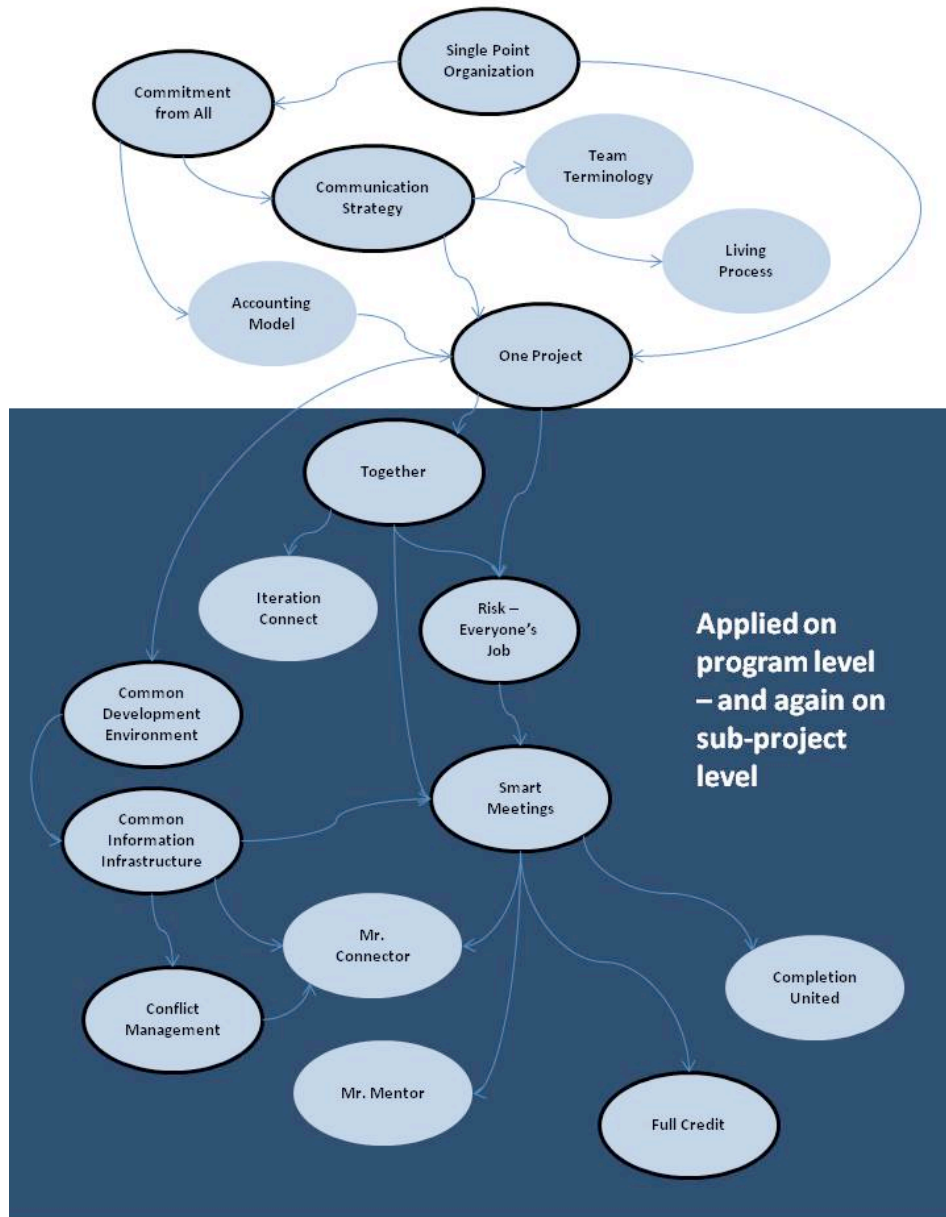


Figure 9: Patterns sequence for the Monitoring & Control System team

### Other Patterns in this Collection

There are seven patterns from the thumbnails that are not used in the three stories in this paper. Most of these patterns are patterns that can be applied long-term in an international organization to create a culture that more easily would support distributed development: an international workforce with one class of citizenship independent of nationality, and global communities of practice for sharing of knowledge.

## Relationship to Other Patterns

Note that the focus in this paper is on distributed development, and the stories and discussions are not extensively covering other aspects of project management or development.

As mentioned in the introduction, there are other patterns outside this collection that are of great value to distributed teams, too many to explicitly mention here. I have selected a few collections to elaborate on here, and can just suggest to interested reader to explore more pattern collections for useful practices.

The book by James C. Coplien and Neil Harrison “Organizational Patterns for Agile Software Development” has one hundred patterns that will help the performance of a software project team. Below is a short list of patterns that are especially important for distributed teams sorted into organizational focus (how to structure the team and the work) and people focus (communication and mentoring):

- Conway’s Law that promotes aligning the organizational structure with the structure of the business domains reflecting the structure of the architecture
- Divide and Conquer to partition a larger organization into parts with mutual interest/focus
- Loose Interfaces to limit the number of explicit, static interfaces to allow for more rapid development
- Organization Follows Location to organize work so that people who are co-located work on the same subjects
- Subsystem by Skill recommends organizing sub-teams based on their skills
- Form Follows Function to create domains of expertise
- Size the Organization to balance good communication and the ability to mentor new people joining (start with about 10 people on the team)
- Producers in the Middle to make sure the producer roles are at the center of communication as they need to have necessary information to ensure the right product is being developed
- Named Stable Bases promotes having stable code bases for people to work against to give more stability for their development; needs to be carefully applied in combination with Conway’s Law and Organization Follows Location, as well as the attitude to continuous integration
- Community of Trust that aims to build a foundation of trust and respect to benefit effective communication
- Face to Face Before Working Remotely as emphasized in this paper as well,
- Unity of Purpose to make sure all agree on the purpose of the team from the start
- Day Care where one person is assigned the role of mentoring the novices because this is more effective than distributing the task among all the experienced staff
- Compensate Success is the basis for the Full Credit pattern in this paper, emphasizing on the importance of rewarding any success for the team
- Team Pride which is important for any team but even more when the team is particularly challenged which is usually the case for distribution

- Self-Selecting Teams where the team members share common interests also outside work, and where the selection process is a team effort and not a manager decision

### **Acknowledgements**

A big thank you to Till Schümmer for his very thorough work as a shepherd for EuroPLoP 2009! I also want to thank Amir Raveh for his initial shepherding of this paper for EuroPLoP 2008, for giving me good feedback and surviving my slow progress. I am grateful to shepherds of earlier work that has led up to the patterns in the thumbnails (Jens Coldewey, Mary Lynn Manns, Neil Harrison, Didi Schütz and Joe Bergin), and of course all the workshop participants giving valuable feedback to the patterns.



## Related Reading

1. “Organizational Patterns of Agile Software Development” by James O. Coplien and Neil B. Harrison, ISBN 0-13-146740-9, Pearson Prentice Hall 2005
2. “Using an Agile Software Process with Offshore Development” by Martin Fowler, <http://www.martinfowler.com/articles/agileOffshore.html>
3. “Global Software Development Handbook” by Raghvinder Sangwan, Matthew Bass, Neel Mullick, Daniel J. Paulish, and Juergen Kazmeier, ISBN 0-8493-9384-1, Auerbach Publications, 2007
4. “Can absence make a team grow stronger?” by A. Majchrzak, A Malhotra, J. Stamps and J. Lipnack, Harvard Business Review, May 2004
5. “Interaction Patterns of Agile Development” by Jens Coldewey, 2004
6. “Capable, Productive and Satisfied” by Paul Taylor, PLoP 1998.
7. “201 Principles of Software Development” by A.M. Davis, McGraw-Hill, 1995
8. “Culture Clash – Managing the Global High-Performance Team” by Thomas D. Zweifel, ISBN 1-59079-051-0, Swiss Consulting Group 2003
9. “Global Teams – How Top Multinationals Span Boundaries and Cultures with High-Speed Teamwork” by Michael J. Marquardt and Lisa Horvath, ISBN 0-89106-157-6, Davies-Black Publishing 2001
10. “Mastering Virtual Teams – Strategies, Tools, and Techniques that Succeed” by Deborah L. Duarte and Nancy Tennant Snyder, ISBN 0-7879-5589-2, Jossey-Bass 2001
11. “Trust within Global Virtual Teams” by Olivier Chavaren, ISBN 0-595-27577-X, iUniverse, Inc 2003
12. “Virtual Teams – Reaching across Space, Time, and Organizations with Technology” by Jessica Lipnak and Jeffrey Stamps, ISBN 0-471-16533-0, John Wiley & Sons 1997
13. “The Manager’s Pocket Guide to Virtual Teams” by Richard Bellingham, ISBN 0-87425-615-1, HRD Press, Inc 2001
14. “Global Software Development – Managing Virtual teams and Environments” by Dale Walter Karolak, ISBN 0-8186-8701-0, IEEE Computer Society 1998
15. “Managing Virtual Teams – Practical Techniques for High-Technology Project Managers” by Martha Haywood, ISBN 0-89006-913-1, Artech House 1998
16. “Working Virtually – Managing People for Successful Virtual Teams and Organizations” by Trina Hoefling, ISBN 1-57922-032-0, Stylus Publishing 2001
17. “The Distance Manager – A Hands-on Guide to Managing Off-Site Employees and Virtual Teams” by Kimball Fisher and Mareen Duncan Fisher, ISBN 0-07-13065-4, McGraw-Hill 2001
18. <http://alistair.cockburn.us/Information+radiator>
19. [http://en.wikipedia.org/wiki/Unified\\_Process](http://en.wikipedia.org/wiki/Unified_Process)

## **Appendix: Pattern Thumbnails**

The short descriptions below are based on the author's earlier papers at EuroPloP and PLoP conferences, but include some that are not yet published or submitted (ongoing work). A major revision is planned for some of the earlier work.

### **1. Relocation and Rotation**

You need an international workforce with a strong common identity that collaborate well and that is not having a "them" and "us" mentality.

Relocate a part of your workforce on a rotational basis to other engineering centers.

This will build relationships and trust across locations, and enforce common work methods.

### **2. Balance of Nationalities and Minorities**

You want a workforce that is representative of your business involvement worldwide, and where individuals are equally respected independently of their background.

Hire with a clear goal that the different nationalities represent the amount of involvement in a country or region. The distribution of nationalities must be reflected in the management organization all the way up to the CEO.

### **3. One Class of Citizenship**

You want a workforce with a strong common identity, where individuals are equally respected and feel equally valuable to the organization.

Make each location comply with the overall company policies and give all employees the same opportunities and the same benefits within what is possible given local laws and regulations.

### **4. International Communities of Practice**

You need to fully benefit from the knowledge and creativity of your whole workforce to stay competitive. You want the participation of every individual, and that they actively participate and collaborate with other experts within their application domain.

Grow and support technical communities of practice on the global level. Provide means of networking through blogs and wikis, and enable workshops and conferences where members can physically meet.

### **5. Benefit Target**

The success of distributed teams is heavily discussed. Results are often questionable, and the drawbacks often by far outweigh the achieved benefits.

To be able to keep the focus on the expected benefits and to determine achieved improvements, define 3 major targets for the expected benefits that are not in conflict with each other. The benefits must be specific to the organization and allow the organization to determine afterwards if each targeted benefit was reached.

**6. Commitment from All**

Even if the project is distributed over several locations, the management interest and attention may be stronger in some locations. This may lead to lower priority and lack of resources in the other locations.

Any engineering center involved must have a clear commitment to the project. In the initial phase, a clearly defined involvement must be agreed between the involved centers.

**7. Single Point Organization**

Running projects in a distributed way often results in confusion and frustration for the involved parties. Conflicting objectives and unclear authority and responsibility distribution are just examples.

The complexity can be reduced and controlled by a clear and communicated organization. This is more than defining the roles and responsibilities: it is key to ensure that the decision-making authority lies with only one manager at each level in the organization.

**8. One Project**

The daily life of team members will be influenced by the location. Developers may find conflicts between local objectives and what is expected as product deliveries. Create a clear project identity across the centers. All team members share the same team objectives.

**9. Communication Strategy**

Distributed teams may have a more complex set of management and stakeholders, and how/when/whom to keep informed may be a challenge.

The team needs to work out a clear communication strategy that lists who to inform, what each are expecting, and the communication means.

**10. Accounting Model**

A project team depends on efficient accounting “services” in the organization to keep track of spending versus funding. These services are rarely set up to support a team on multiple locations.

Set up a meta-level model of the accounting that defines how to manage the project financial status, and assign a clear owner within the accounting organization.

**11. Common Information Infrastructure**

A project team must share information across the complete team, not based on physical location.

Set up an information structure for the global team that enable consolidated reporting of status and easy sharing of information.

**12. Common Development Environment**

Distributed teams have the same needs for baseline management as co-located teams. Keeping the code-base in sync requires immediate updates at all locations.

Carefully identify, select and implement one single common development environment for the project team to use regardless of their location. All components of this common development environment must support multiple locations with different time zones, and come with worldwide 24/7 support.

**13. Culture Awareness**

All cultures have their unwritten rules, and there are many examples of collaboration problems that stem from a lack of understanding of ethical and behavioral code. Note that the culture aspect includes organizational culture!

Since we are not all born social anthropologists, some specific training may be necessary to learn to “read” team members from a different culture, and to ensure respect and good working relations.

**14. Selected**

Working on a distributed team is demanding, and requires flexibility in work hours, ability to travel and stay at other locations, and good social skills for the collaboration within a very heterogeneous team.

Team members should be screened for personality and the team carefully built over time for maximum cohesion.

**15. Prepared**

Distributed teams have some added challenges, especially related to communication. The negative effect of not being co-located can be made significantly smaller by applying good practices from the start of the project, so make sure all are trained on distributed development from the start.

**16. Social Funds**

Team members cannot be expected to privately pay for social team events. These events are important for the bonding and thereby for team efficiency.

Make sure funding is allocated to build personal relations between the team members (team building, celebration of achieved milestones etc.).

**17. Mr. Mentor**

There is much to learn from colleagues that have managed distributed teams before, but this learning may need some support from the organization. Each individual does not necessarily know who to ask, or feel that they can “bother” colleagues repeatedly for advice.

By assigning a mentor with experience in distributed development to a new team, ongoing and on-the-job learning is made possible.

**18. Mr. Connector**

Distributed teams struggle with keeping on the same page between locations.

Instead of everyone on the team trying to be on top of all ongoing issues, designate a team member at each site to manage the flow of information, and who knows how to get answers to different kinds of questions.

**19. Living Process**

A shared development process adapted to the team’s needs becomes even more important with the increased challenge and need for formality.

Spell out the fundamental values, methodology and techniques used and keep it up-to-date for all team members (revisit at each iteration).

**20. Early Bonding**

Trust and personal connections need to be built early and are key to have a functional team.

Emphasize on building the team from the start of the project, and include social events.

**21. Smart Meetings**

If co-located, you can do frequent progress meetings, and have spontaneous get-togethers when needed. In a distributed setting meetings need to be organized up front. But your communication needs are still there.

By allocating common time and making team members available to each other at

agreed times during the week, you can mimic spontaneous contact (with a delay), and you ensure frequent communication in the team (several times per week) to avoid locations drifting apart on issues.

## 22. **Together**

Even with Smart Meetings, communication suffer by not being co-located.

Make sure the team physically meets frequent enough during development. For 1-2 year software projects, we have come to believe a 6-8 week frequency is the best compromise between all effects of traveling and the need to meet, each time staying together for about 10 days.

## 23. **Iteration Connect**

The most busy communication phase in a project is at the end of an iteration cycle, plus during the iteration assessment and the start of a new iteration.

By synchronizing the length of the iteration (or a multiple of iterations) with the frequency of the physical team meetings we found the meeting time was spent most effectively.

## 24. **Completion United**

The final phase of a project is again a time where good and fast communication is important. System testers doing the final verification face significant problems when the team is distributed.

We have found it amazingly effective to bring the development team together with all the testers and stakeholder/user representatives on a single site for completion.

## 25. **Flexibility**

The load of communication with other locations that frequently have to happen outside work hours is a big load on employees and on their families. The workday often gets very long, in reality working one normal day plus an extra shift early or late in the day. To balance the effort and get closer to a normal workload, redefine the team's work hours and give the team members flexibility outside common team time.

## 26. **Short Engagements**

Short-term assignments at the other location will enable team members to get to know other team members better.

## 27. **Conflict Management**

Choose and publish a pre-defined way of managing conflicts, possibly involving an independent party outside the core team.

## 28. **Full Credit**

It is amazing that this one needs to be written down, but experience tells us to do it: Make sure all team members get credited for success.

## 29. **Team Space**

Local team rooms need to accommodate visiting team members, VC equipment need to be easily available etc.

## 30. **Team Terminology**

Make sure you do not get local "dialects" and that everyone understand the lingo of the problem domain.

## 31. **Pilot Solutions**

Many of the practices we recommend have a certain cost associated, although we firmly believe they pay back multiple times over. But, there is management to convince.

To gain experience, try out a new solution on one team first. Be sure to record the results well.

# Business Plan Conception Pattern Language

Wim Laurier, Pavel Hruby\*, Geert Poels

Department of Management Information and Operational Management, Faculty of Economic and Business Administration,  
Ghent University, Tweekerkenstraat 2, 9000 Ghent, Belgium  
[wim.laurier@ugent.be](mailto:wim.laurier@ugent.be), [geert.poels@ugent.be](mailto:geert.poels@ugent.be)

\*CSC, Denmark  
[phruby@acm.org](mailto:phruby@acm.org)

**Abstract.** This paper provides generic guidelines for starting entrepreneurs. First, the basic features of a good business plan are addressed in a pattern language for creating an innovative business plan. Second, soft skills for starting entrepreneurs are discussed in a pattern language for interactions with ‘outside’ people, as these interactions are crucial for a validation and realization of a business idea. Together, these two pattern languages create a pattern language for business plan conception. This pattern language for business plan conception should provide a stable conceptual basis that supports starting entrepreneurs in adapting the business plan templates they find to the needs of their business idea instead of adapting their business idea to the templates they find, which should support a successful penetration of new ideas and entrepreneurs into an innovation requiring market.

**Keywords:** Patterns, Entrepreneur, Business Plan, Soft skills

# 1 Introduction

Topic	Content
<b>Problem</b>	How to start a business?
<b>Context</b>	When someone (e.g. you) has an innovative business idea.
<b>Forces</b>	Entrepreneurs that want to start a new business initiative often have the required technical expertise but not the money (i.e. funding), financial expertise and expertise in the section to get the business up and running.
<b>Solution</b>	Start writing a business plan and take care of your soft skills when interacting with other people.
<b>Resulting Context</b>	A business plan helps the entrepreneur to gain more relevant expertise and might convince people to support the entrepreneur financially or with additional expertise. Polished soft skills support this convincing process. (fig. 1)
<b>Design Rationale</b>	Markets require innovation, therefore new ideas and entrepreneurs should be able to penetrate the market successfully. In this process, interactions with new 'outside' people are crucial. A business plan is the ideal tool to assess the feasibility of such new ideas.
<b>Related Work</b>	<p>BUSINESS PATTERNS FOR PRODUCT DEVELOPMENT [1] discriminate four generic kinds of product setups.</p> <p>PATTERNS FOR BUILDING A BEAUTIFUL COMPANY [2] provide insight in the development of a business.</p> <p>A DEVELOPMENT PROCESS GENERATIVE PATTERN LANGUAGE [3, 4] addresses organization design and project management.</p> <p>Additionally, DESIGN PATTERN FOR CREATIVITY [5] can help to invent and evolve innovative business ideas. Online creativity portals (e.g. <a href="http://www.creax.net">www.creax.net</a>) and creativity consultants (e.g. <a href="http://www.creax.com">www.creax.com</a>) can also provide such help.</p>

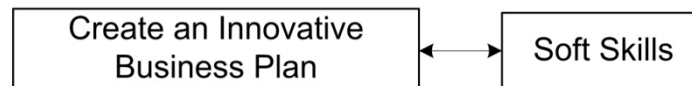


Fig. 1. Interaction between two pattern languages



## 2 Create an Innovative Business Plan

Topic	Content
<b>Problem Context</b>	How to create a good business plan for an innovative idea? Bringing a product to the market first implies high development costs and potentially high returns, whereas copying an existing business idea implies lower development costs lower returns but more certain returns as the market potential has been proven (i.e. if they buy the 'real deal', they will buy the copy also).
<b>Forces</b>	<ol style="list-style-type: none"> <li>1. When inexperienced business plan writers develop a business plan, they are inclined to adapt their idea to a business plan template they found, instead of adapting the template to fit their idea. (As has been demonstrated for other engineering disciplines [6])</li> <li>2. The collection and presentation of detailed technical information (e.g. manufacturing methods, logistics) for the real startup process is mostly supported and heavily influenced by third parties just like the business process itself.</li> </ol>
<b>Solution</b>	Collect, create (re)structure all information that is needed to create a realistic image of the future venture and its implementation process. Reiterate this information collection and creation process to keep the information up to date and improve on the idea and plan.
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. Regardless of business plan templates, the patterns in this section guide the entrepreneur in collecting all relevant information for the creation of a business plan. (fig. 2)</li> <li>2. The reiteration of the information collection and structuring process helps the entrepreneur to identify the influence of third parties and provide a good basis for negotiating with them.</li> </ol>
<b>Design Rationale</b>	Although many good business plan templates <sup>1</sup> exist for proven business idea's (e.g. starting a baker's shop or hairdresser's salon), it is hard or even impossible to find guidance for creating business plans for unique and innovative business ideas.
<b>Related Work</b>	The US Small Business Administration <sup>2</sup> provides guidance, templates and workshops for writing business plans.

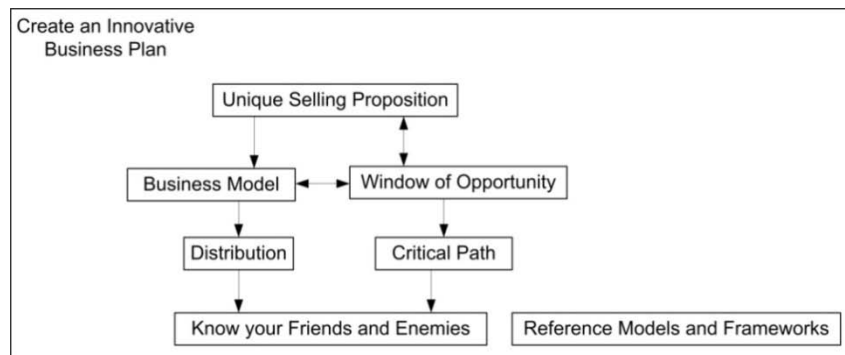


Fig. 2. Structure of the pattern language for creating an innovative business plan

<sup>1</sup> [http://www.bplans.com/sample\\_business\\_plans.cfm](http://www.bplans.com/sample_business_plans.cfm)

<sup>2</sup> <http://www.sba.gov/smallbusinessplanner/plan/writeabusinessplan/index.html>

## 2.1 Unique Selling Proposition (USP)

Topic	Content
<b>Problem</b>	How to demonstrate to potential investors that your venture deserves a place in the market.
<b>Context</b>	When you have an innovative idea, and want to CREATE AN INNOVATIVE BUSINESS PLAN.
<b>Forces</b>	Truly innovative selling propositions are unique by definition but the returns they will generate are highly unpredictable, which makes convincing potential investors tough.
<b>Solution</b>	List the benefits of your venture for each of your stakeholders and demonstrate or motivate why they will accept your offer and not that of a competitor. Identify the strengths and weaknesses of the idea, foresee the opportunities and threats in the market and mitigate for potentially negative effects.
<b>Resulting Context</b>	Motivating why customers will prefer your offer substantiates the expected returns. The mitigation scenarios compensate for the unpredictability.
<b>Design Rationale</b>	To attract investors, a venture needs to demonstrate or at least argument that it will be able to earn and defend its place in the market.
<b>Related Work</b>	<p>The following patterns describe how to construct USP's:</p> <ul style="list-style-type: none"> <li>- BUSINESS PATTERNS FOR PRODUCT DEVELOPMENT [1]</li> <li>- DESIGN PATTERNS FOR SOFTWARE COMPANIES [7, 8]</li> <li>- DESIGN PATTERNS FOR TECHNOLOGY COMPANIES [9]</li> <li>- BUSINESS STRATEGY DESIGN PATTERNS FOR TECHNOLOGY COMPANIES [10]</li> <li>- A FEW MORE BUSINESS DESIGN PATTERNS [11]</li> <li>- BUSINESS STRATEGY PATTERN FOR THE INNOVATIVE COMPANY [12]</li> <li>- THE PORTER PATTERNS [13]</li> </ul>

## 2.2 Business Model

Topic	Content
<b>Problem</b>	How will you use your limited resources to execute and realize your unique selling proposition?
<b>Context</b>	When you have developed a UNIQUE SELLING PROPOSITION.
<b>Forces</b>	The amount of funding that can be attracted is determined by the returns a business process can generate. Conversely, the scale of a business process, and consequently the returns it can generate, are constrained by the amount of funding that it can attract.
<b>Solution</b>	<ol style="list-style-type: none"> <li>1. Check whether everything is legal (e.g. do not violate intellectual property rights)</li> <li>2. Make or Buy <ol style="list-style-type: none"> <li>a. Determine the things that are specific for your USP, protect them against competitors (e.g. profit from first mover advantage, create confidentiality agreements, intellectual property rights, patents) and keep the execution of those activities under your control.</li> <li>b. Try to find partners for all non-USP-specific activities (i.e. outsourcing).</li> </ol> </li> </ol>
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. Legal activities mostly do not affect the investor's image negatively, which influences their motivation to lend money positively.</li> <li>2. Make or Buy <ol style="list-style-type: none"> <li>a. Performing activities in house requires specific equipment and expertise. The scale of these activities largely determines the need for funding.</li> <li>b. Performing activities out house reduces the amount of funding needed, since these activities can mostly be categorized as variable costs.</li> </ol> </li> </ol>
<b>Design Rationale</b>	Resources are scarce by definition [14] therefore they cannot be generated easily and disbursed without a proper motivation.
<b>Related Work</b>	The e3-value [15] tool ( <a href="http://www.e3value.com">www.e3value.com</a> ) provides help for visualizing business models and making a first profitability analysis.

### 2.3 Distribution

Topic	Content
<b>Problem</b>	How to reach your stakeholders and customers?
<b>Context</b>	When you have developed a USP and BUSINESS MODEL.
<b>Forces</b>	<ol style="list-style-type: none"> <li>1. The chosen venture location, virtual (e.g. market) or real (e.g. site), also determines the uniqueness of your proposition and the appropriateness of distribution strategies.</li> <li>2. Using other distribution channels than competitors may create competitive advantage, but may also prove to be risky (e.g. failure to profit from economies of scale, poor distribution quality) and expensive. However, choosing the same distribution channels as your competitors may constrain the uniqueness of your USP.</li> <li>3. Maintaining existing distribution channels is often more cost-efficient than finding new ones and keeping existing customers (informed) is cheaper than finding new prospects (or informing them about your product).</li> </ol>
<b>Solution</b>	<p>Elaborate an adequate acquisition and distribution strategy for products and information, which stands out in the 'noise' that is created by competitors.</p> <ol style="list-style-type: none"> <li>1) Targeting a market (niche) in which you can stand out.</li> <li>2) Select communication and sales strategies that are appropriate for the target market (niche).</li> </ol>
<b>Resulting Context</b>	Narrowing your USP towards a particular target audience, increases the potential market share but reduces the size of the targeted market.
<b>Design Rationale</b>	Your products and relevant information about them do not automatically reach your target audience.
<b>Related Work</b>	A PATTERN VOCABULARY FOR PRODUCT DISTRIBUTION [16] lists distribution strategies for products, not for information.

## 2.4 Window of Opportunity

Topic	Content
<b>Problem</b>	How to take maximum advantage of an opportunity?
<b>Context</b>	Innovative business ideas (i.e. USP and BUSINESS MODEL) create an opportunity.
<b>Forces</b>	<ol style="list-style-type: none"> <li>1. Customers are not always susceptible for the advantages your idea has to offer or the advantages may be bound to a specific time window (e.g. ice cream in the summer)</li> <li>2. Investors are happy to have their cash back as soon as possible and with the highest possible return, but what's in it for the entrepreneur?</li> </ol>
<b>Solution</b>	<p>Determine the window of opportunity meticulously. When there is no natural window of opportunity create one with a proper marketing strategy (e.g. media campaign).</p> <p>When the window of opportunity opens, saturate the market sufficiently fast to prevent competitors from entering the market (i.e. penetration pricing strategy), which would decrease your market share and hence volume.</p> <p>Maximize your profit by using the skimming technique (i.e. start with a high introduction price and lower your price over time) when competitors are not able to enter the market (e.g. product cannot be copied) before the window of opportunity closes.</p>
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. When customers are/have been made susceptible (i.e. awaking latent need) for the advantages your product has to offer, competitors can also take advantage of this susceptibility.</li> <li>2. Penetration strategy (Low price, High Volume), which requires extra funding for a longer period as the product volume that needs to be prefinanced is larger and the profit margin is lower. Skimming (High price, Low Volume), on the other hand, requires little funding and can be self-sustaining relatively early.</li> </ol>
<b>Design Rationale</b>	The net present value of revenues generated by an innovative idea needs to be maximized over the entire product lifecycle. [17]
<b>Related Work</b>	The TAKE NO SMALL SLIPS [4] pattern tackles how a project should be scheduled, not to miss the market window.

## 2.5 Critical Path

Topic	Content
<b>Problem Context</b>	How to plan, coordinate and control the implementation of your business idea.
<b>Forces</b>	When you have identified the WINDOW OF OPPORTUNITY and do not want to miss it.
	<ol style="list-style-type: none"> <li>1. Working fast leads to large negative cashflows (i.e. costs), on the other hand, working slow leads to a longer the critical path, which delays positive cashflows (i.e. revenue).</li> <li>2. The longer it takes to put your product in the market, the higher the chance competitors or copycats will bring a similar or better product to the market. The shorter the time to market, the larger the chance your product still has (minor) flaws.</li> </ol>
<b>Solution</b>	<ol style="list-style-type: none"> <li>1) Determine the sequence in which the implementation steps need to be executed, plan the process such that steps that can be executed in parallel are planned in parallel and discriminate process steps that are crucial for the timely execution of the project from the process steps that are not.</li> <li>2) Visualize the process and its constituting steps (e.g. PERT [18] and Gantt [19-21] chart)</li> <li>3) When the sequence of process steps that is crucial for the timely execution of the entire sequence (i.e. the critical path) has been identified; plan and monitor the execution of the critical path elements meticulously and prevent parallel path elements from interfering with critical path elements (i.e. plan them and limit the delay in their execution such that they do not delay the execution of critical path elements).</li> </ol>
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. A proper visualization of the process allows representing the trade-off between project cost and timing.</li> <li>2. The minimal length of the critical path indicates how fast competitors can catch up with the venture if they possess sufficient resources.</li> </ol>
<b>Design Rationale</b>	Time is money: Missing the window of opportunity (i.e. being too early or too late) likely means failure.
<b>Related Work</b>	The DE-COUPLE STAGES [4] and SIZE THE SCHEDULE [4] pattern present an approach for (re)designing a critical path.

## 2.6 Know your Friends and Enemies

Topic	Content
<b>Problem</b>	How to cope with parties that (will) have stakes in your business and opposing or common interests.
<b>Context</b>	When you need to interact with third parties (e.g. suppliers, customers, investors, neighbors, etc.) to execute your DISTRIBUTION strategy and CRITICAL PATH.
<b>Forces</b>	<ol style="list-style-type: none"> <li>1. Most entrepreneurs are only aware of the stakeholders they (plan to) interact with on a regular basis (e.g. suppliers, customers, debtors and creditors), while largely neglecting other stakeholders (e.g. neighbors, which can prevent the plant from expanding or force the enterprise to relocate their activities).</li> <li>2. Although entrepreneurs are aware of their stakes in other parties, they are mostly unaware of the stakes other parties have in their business.</li> </ol>
<b>Solution</b>	Make an exhaustive list of your (future) competitors (i.e. enemies) and collaborators (i.e. friends) and identify their stakes in your activity. Pay special attention to those that are both friend and enemy (e.g. suppliers, customers).
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. Making an explicit and exhaustive list of friend and enemies helps to identify and monitor the forces that may influence your strategy.</li> <li>2. Making the list of friends and enemies makes entrepreneurs aware of the stakes other people have in their business.</li> </ol>
<b>Design Rationale</b>	Being profitable (i.e. generating your own ROI) through value creation, is at the same time a collaborative and a competitive process [22], which makes an organization's relation to its environment ambiguous.
<b>Related Work</b>	IN BED WITH THE ENEMY [23] shows how opposing interest can become common interests. CUSTOMER INTERACTION PATTERNS [24] address how a specific kind of stakeholder (i.e. customers) should be dealt with. Porter's Five Forces model [25] identifies the major categories of enemies.

## 2.7 Reference Models and Frameworks

Topic	Content
<b>Problem</b>	How to identify structure and assess all internal, external, positive, negative, future, current and historical influences that concern your business?
<b>Context</b>	When you have identified the factors that may influence your business. (cf. KNOW YOUR FRIEND, KNOW YOUR ENEMY)
<b>Forces</b>	The information you collect about your environment is usually incomplete unstructured, but unstructured information is difficult to process and incomplete information gives an unprofessional impression.
<b>Solution</b>	Use existing frameworks and reference models to structure the information you collect and identify the lacunas in the information gathered.
<b>Resulting Context</b>	Structured information is easier to process and using well-known frameworks and reference model to structure the information amplifies this advantage. The structure of the frameworks and reference models also provides a blueprint for the information search process.
<b>Design Rationale</b>	An enterprise is not an island that operates independent of its environment [26]; therefore, the enterprise's influence on the environment and the environment's influence on the enterprise need to be addressed.
<b>Related Work</b>	<p>Well-known reference models and frameworks are:</p> <ul style="list-style-type: none"> <li>• SWOT [27] provides a template for representing all internal and external factors that influence the business' performance positively and negatively, in an efficient visualization where every kind of factor has a fixed location in the diagram [28].</li> <li>• Porter's Value Chain [29] gives a generic overview of business activities.</li> <li>• Porter's Five Forces [25] list the main threats a business faces.</li> <li>• McKinsey's 7-S Framework [30] list 7 critical success factors that a business needs to achieve its goals.</li> <li>• Product Lifecycle [17, 31] shows that the real value of innovation lies in a future adaptation by the market.</li> </ul> <p>These and other reference models and frameworks that are also applicable when a business has been established (e.g. BCG [32], GE Model[33], Ansoff matrix [34]) are collected in Kotler's work [35, 36].</p>

### 3 Soft Skills

Topic	Content
<b>Problem Context</b>	How to be convincing? When you have CREATED AN INNOVATIVE BUSINESS IDEA for which you need support and additional expertise or when such an idea has evolved into a business plan for which you need investors, customers and suppliers.
<b>Forces</b>	A sector mostly overlaps with a closed social network of peers. Therefore, occasions to meet the right people are rare if you are not part of the network of peers.
<b>Solution</b>	Take care of your soft skills when you have the opportunity to enter the network of peers that overlaps with the market you want to enter.
<b>Resulting Context</b>	When the peers perceive your presence as valuable, other opportunities to meet the network of peers will be offered to you. When your soft skills are taken care of, the chance that your presence will be perceived as comforting and valuable increases. The EYE CATCHER pattern addresses how to be noticed and remembered, the DREAM TEAM pattern supports the creation of a valuable team image and the CASH IS KING pattern helps to demonstrate the value of business ideas. The support and expertise of the network of peers should then make FINDING FUNDING easier. (fig. 3)
<b>Design Rationale Related Work</b>	When inexperienced entrepreneurs have a business idea it proves difficult to give their developing business sufficient momentum to turn it into a real business. PATTERNS FOR LEADING EFFECTIVE AND EFFICIENT MEETINGS [37, 38] addresses other soft skills for entrepreneurs that are not considered in this paper. LEARNING PATTERNS: A PATTERN LANGUAGE FOR ACTIVE LEARNERS [39] features one of the most important skill of an entrepreneur. The BUSY PERSON PATTERNS [40] then address the features of personal effectiveness and time management that are also crucial for entrepreneurs.

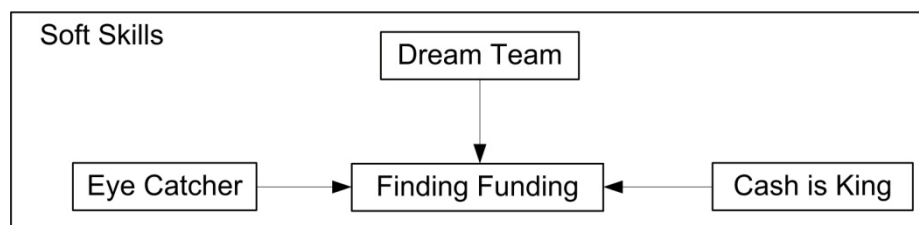


Fig. 3. Structure of the soft skills pattern language



### 3.1 Eye Catcher

Topic	Content
<b>Problem Context</b>	How to make sure that people remember you? The opportunities to meet important people are scarce. As they probably meet many people and you are not their priority, they tend to forget about your encounter.
<b>Forces</b>	It is hard to balance a professional image, which results from a good practice, with standing out in the noise, which requires something exceptional or surprising.
<b>Solution</b>	Choose a company name and design a logo that is easy to recall and unique at the same time, and use your logo in every document (e.g. name card, business plan), drop the company name in every conversation (e.g. when you introduce yourself) and do not economize on business cards.
<b>Resulting Context</b>	A good company name and logo do not affect your professional image negatively like eye-catching clothing or behavior can do.
<b>Design Rationale</b>	People need to remember who you are and what you stand for, before they will contact you.
<b>Related Work</b>	AIDA (i.e. Attention-Interest-Desire-Action)[41] and related [42] models, like the diffusion of innovation [31], see attention or awareness as a first step and condition sine qua non for communication or action.

### 3.2 Dream Team

Topic	Content
<b>Problem</b>	How the create a team that can make the venture flourish in an uncertain environment?
<b>Context</b>	When you need to convince people that your business plan is realistic and sustainable.
<b>Forces</b>	<ol style="list-style-type: none"> <li>1. Managing a venture requires a large amount of energy, motivation and dedication, and many different types of skill, knowledge and experience.</li> <li>2. Nobody is perfect.</li> </ol>
<b>Solution</b>	Create a team that incorporates as many desired skills and as much relevant knowledge, experience, motivation and energy as possible, but also recognize the shortcomings of the management team and develop strategies to mitigate the consequences.
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. As many hands make light work, a team is more than the sum of its members (i.e. synergy), has access to more information and a larger and more divers social network. A TEAM IMAGE convinces outsiders that the synergy between the members will work.</li> <li>2. A well-designed management team that knows its own qualities and shortcomings and knows when to look for external expertise. A CURRICULUM VITAE is an instrument to make these qualities and shortcoming explicit. The DOOR OPENER describes an essential quality that should be present in a venture starting team. (fig. 4)</li> </ol>
<b>Design Rationale</b>	No matter how meticulously a business initiative is planned, its predestined interactions with an uncertain environment will effect in uncertain outcomes. Therefore, a venture should have a management team that is able to cope with environmental changes.
<b>Related Work</b>	The 3 TO 7 HELPERS PER ROLE [4] addresses group dynamics. The REVIEW [4] pattern highlights one of the advantages of teamwork. The BEAUTIFUL PEOPLE [2] pattern can help to find the right team members.



Fig. 4. Structure of the Dream Team pattern language

#### 3.2.1 Curriculum Vitae

Topic	Content
<b>Problem</b>	How to convince people that you are the right person for the job?
<b>Context</b>	When you want to be part of the team.
<b>Forces</b>	No matter which diplomas you have or knowledge and skills you demonstrate, people tend to be reserved about your qualities in real life situations.
<b>Solution</b>	Present them your curriculum vitae
<b>Resulting Context</b>	A CV/resume shows prior experience in real life situations. Potential stakeholders value prior experience in the sector extremely high. Also prior success in other sectors might be valued, as your 'refreshing' outside view might compensate for the lack of experience in the sector.
<b>Design Rationale</b>	If people are not convinced that your team can handle the job, they will not support the initiative.
<b>Related Work</b>	Digitalized curriculum vitae's (e.g. [43]) are hot. Also many tips and tricks for writing CVs can be found on the net and in magazines [44-46].

## 3.2.2 Team image

Topic	Content
<b>Problem</b>	How to make obvious that a group works as a team?
<b>Context</b>	When other people (e.g. potential stakeholders) judge your team on its coherence and team spirit.
<b>Forces</b>	<ol style="list-style-type: none"> <li>1. People's judgment is mainly subjective.</li> <li>2. Different roles in a team might make it difficult to create a noticeable unity in the group.</li> </ol>
<b>Solution</b>	<p>Use Gestalt principles [47, 48] to appear as a unit:</p> <ul style="list-style-type: none"> <li>• Similarity: Wear similar clothing or a uniform (e.g. T-shirt with company logo)</li> <li>• Proximity: Stay relatively close to each other, do not scatter throughout the entire room and frequently consult each other (briefly) so that see that you know each other, without losing too much time that can be used for meeting new people.</li> <li>• Common-fate: Enter/leave a room simultaneously and use the same entrance/exit</li> <li>• Closure: Even when your team is scatters in a room and every team member is talking to a different person, team members should try to stay within each other's field of vision, so that they never turn their back on each other.</li> </ul>
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. Gestalt laws describe the rules that govern human perception.</li> <li>2. For the DOOR OPENER, following these guidelines (e.g. proximity) might be difficult (e.g. a Door Opener needs to talk to everyone in the room irrespective of his/her proximity to the team). Consequently, he/she might be perceived as an outsider. However, the DOOR OPENER's deviant behavior and appearance might make it easier for the other group members to build a team image (e.g. stick together).</li> </ol>
<b>Design Rationale</b>	People take team appearance as a proxy for team spirit.
<b>Related Work</b>	Chaturvedi [49] discusses how teams operate in practice in his pattern TEAM WHERE PEOPLE MATTERS – A PROJECT MANAGEMENT PATTERN.

### 3.2.3 Door opener

Topic	Content
<b>Problem Context Forces</b>	<p>How to get in touch with the right people?</p> <p>When you have been able to penetrate the right social network.</p> <ol style="list-style-type: none"> <li>1. The right people might come to you for the wrong reason.</li> <li>2. Wrong people might come to you for the right reason.</li> <li>3. Interesting people might not speak to you because you are already involved in a conversation.</li> <li>4. Interesting people might not speak to you because they do not know what your expertise and goals are.</li> </ol>
<b>Solution</b>	<p>Select one team member as a contact person. Make obvious to others who is the contact person:</p> <ul style="list-style-type: none"> <li>– Make him/her wear brighter colors than other team members.</li> <li>– Take advantage of conspicuous physiognomic and other characteristics that make a contact person stand out in a crowd (e.g. select and attractive and/or tall contact person).</li> <li>– When the team enters a room, building or other location, the contact person walks up front, and the contact person is the first to greet people (e.g. shake hands) or start a conversation.</li> </ul> <p>Dedicate the contact person to having short conversations with as many persons as possible, after which interesting contacts are introduced to team members with the right expertise to continue the conversation.</p>
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. When people start a conversation with the contact person, the contact person has the opportunity to introduce you properly and start the conversation on topic.</li> <li>2. Since the contact person is the first one to talk to people, he/she has the opportunity to bounce people that cannot contribute to your initiative, without damaging the image of the other team members.</li> <li>3. As the contact person is the first one to talk to people, he/she knows who you are talking to and has the opportunity to interrupt ongoing conversations and introduce more interesting people, while luring the less interesting people away (e.g. by introducing them to other people he/she met).</li> <li>4. The contact person serves as a kind of ‘table of content’ for the team’s expertise.</li> </ol>
<b>Design Rationale</b>	Getting in touch with the right people can help to speed up the maturation process of your venture.
<b>Related Work</b>	The GATEKEEPER [4] pattern shows the other side of the DOOR OPENER (i.e. bouncing irrelevant people with a smile).

### 3.2.4 Beauty Queen

Topic	Content
<b>Problem</b>	Who should play the DOOR OPENER role?
<b>Context</b>	As DOOR OPENERS are dedicated to short conversations, they might be perceived as superficial people.
<b>Forces</b>	<ol style="list-style-type: none"> <li>1. During social events, DOOR OPENERS should be entirely dedicated to their role.</li> <li>2. If the DOOR OPENER role is combined with expert roles, the person that combines these roles dominates the conversations and makes the other team members look unimportant.</li> </ol>
<b>Solution</b>	<p>Avoid combining the DOOR OPENER role with other roles by engaging a team member with these specific skills (e.g. beauty contest finalist and professional sales people are typically skilled DOOR OPENERS).</p> <p>If the DOOR OPENER role needs to be combined with an expert role, make sure that another team member can take over this role and only needs support from the DOOR OPENER/expert in exceptional cases.</p>
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. If a dedicated DOOR OPENER is engaged or the expert role can be fulfilled by another team member, the DOOR OPENER can focus on his/her role.</li> <li>2. If a dedicated DOOR OPENER is engaged, he or she does not have the expertise to dominate the conversation. If another team member can take over the DOOR OPENER's expert role, the team roles will look more balanced.</li> </ol>
<b>Design Rationale</b>	The roles of the team members need to be balanced so that all team members are perceived as equally relevant.
<b>Related Work</b>	The BEAUTY QUEEN pattern is a special case of the DOOR OPENER pattern.

3.3 Cash is King

Topic	Content
<b>Problem</b>	How to express benefits using one unifying measure?
<b>Context</b>	When your plan needs to be compared to other (competing) plans.
<b>Forces</b>	Finding the appropriate means to motivate people and expressing the benefits of your proposition is difficult, as different people might have different objectives.
<b>Solution</b>	Express the benefits of your proposition to stakeholders in monetary units and create financial projections (e.g. cashflow statements, annual accounts).
<b>Resulting Context</b>	<p>Money (i.e. cash) is a good measure since can be exchanged for virtually every other type of resource. However, the monetary zone in which your stakeholders reside determines the monetary unit (e.g. €, \$, £) in which benefits should be expressed, and taxes and exchange rates affect the benefits stakeholders actually receive (i.e. net benefit).</p> <p>The INCENTIVE pattern demonstrates how cash can be used to convince people. The ROADMAP THROUGH DEATH VALLEY identifies how much cash is needed to start a venture and the BEST-BASE-WORST CASE SCENARIO addresses volatility in both outcomes. (fig. 5)</p>
<b>Design Rationale</b>	Investors often like to assess the benefits of totally different business initiatives; therefore they require a unifying measure.
<b>Related Work</b>	Financial prognoses in business plans. (e.g. determining projected sales and acquisition cashflows by means of projected sales and acquisition volumes (e.g. market share) and projected market prices) [50]

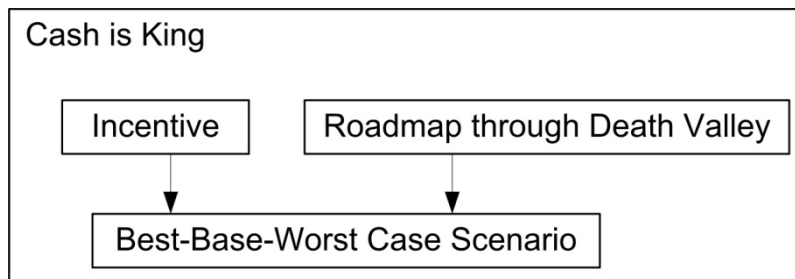


Fig. 5. Structure of the Cash is King Pattern Language

### 3.3.1 Incentive

Topic	Content
<b>Problem</b>	How to convince people to contribute to your initiative?
<b>Context</b>	When you have CREATED AN INNOVATIVE BUSINESS PLAN and built a DREAM TEAM.
<b>Forces</b>	<ol style="list-style-type: none"> <li>1. The larger the time interval between the contribution and the reward the more intense the reluctance is, but for the distributor, the cost of the recompense decreases over time.</li> <li>2. People prefer certain over uncertain outcomes (i.e. risk aversion). The perceived risk is determined by the available information (e.g. the experience of the assessor, the information in the business plan).</li> </ol>
<b>Solution</b>	Show every contributor that you will give him/her something he/she values more in return for his/her contribution, and make sure the aggregate of your trades (i.e. contribution for recompense) is sustainable (i.e. the aggregated value of the contributions is higher than the aggregated cost of the recompenses). Reward contributors that need to wait longer and bear more (perceived) risk with higher returns (i.e. risk premium).
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. Stakeholders that know what returns they may expect are more inclined to provide and spend money and take the 'risk'. As long as they value their reward higher than their contribution, their reluctance to contribute will diminish. If they decide to contribute, win-win situations will occur (i.e. each participant will benefit from the cooperation).</li> <li>2. A higher expected return balances the risk. However, there is a limitation to the power of money since business initiatives with a considerable financial return on investment might be considered unrewarding when they endanger the sustainability of the investor's earnings by damaging his/her image.</li> </ol>
<b>Design Rationale</b>	When you know people that have the assets (e.g. expertise, means) that you need to make your business a success, cooperation needs to be achieved. Although cooperation is often required to create successful initiatives, people are reluctant to contribute (i.e. spend/invest energy/effort/time/money) to an initiative that is not theirs. Therefore, all participants in/contributors to an initiative need to be rewarded. Consequently, the question "What's in it for me?" needs to be addressed for every single participant (e.g. investor, customer, supplier, team member, employee).
<b>Related Work</b>	The logic of the INCENTIVE pattern is also applied in the COMPENSATE SUCCESS pattern. [3] Risk aversion is the motivation for risk management (e.g. [51], [52]). Maslow [53] wrote a standard work on motivation of people in general (i.e. including non-financial incentives). So did Herzberg [54-56], in an economic context. Gossen posed saturation laws for stimuli. [57]

### 3.3.2 Roadmap through Death Valley

Topic	Content
<b>Problem Context</b>	<p>How to avoid illiquidity when a business is growing?</p> <p>Since (fixed) assets have a payback period, newly founded and fast growing (i.e. rapidly increasing assets) businesses temporarily consume more cash than they can generate. Even existing projects can grow so fast that they cannot provide their own funding.</p>
<b>Forces Solution</b>	<p>New business initiatives often require more funding than the entrepreneur can provide.</p> <p>To prevent insolvency, the project should also attract external funding, estimate the amount of cash that is required to implement the business plan properly and look for investor funding with the right maturity (i.e. match the average lifespan of your assets with that of your liabilities). Find a solid base (e.g. existing data) and a logical buildup for your prognoses, and compare the outcomes of different estimates using different methods and data sources.</p>
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. Cash that has been obtained from investors needs to be returned augmented with an ROI (i.e. an additional cash outflow). Consequently, external funding increases the amount of cash available at one moment, but decreases the potential amount of cash in the future, as the ability of a business process to generate (more) cash in the future is constrained by the amount of cash it has generated in the past and additional cash it can obtain from investors.</li> <li>2. When the average maturity of the assets and liabilities match, the debt structure is sufficiently stable to support the venture and sufficiently flexible to follow the changing need for funding during the different lifecycle stages of the venture. A planning that is too optimistic burns money too fast and makes stakeholders lose confidence, while a planning that is too pessimistic creates liquidity constraint when the enterprise grows faster than expected and gives competitors the opportunity to provide the products that you cannot deliver.</li> <li>3. The sensitivity analysis for financial milestones (e.g. break-even, reimbursement period, internal rate of return) serves as an implicit contract between investor and entrepreneur.</li> <li>4. Throughout the organization's lifecycle, potential returns on investment decrease considerably, while the level of certainty increases.</li> <li>5. In existing organizations, new business initiatives are solely evaluated in terms of generated cashflows (i.e. the contribution to the organization) when no external funding is required, whereas projects that require external funding (e.g. ventures) need to be complemented with projected annual accounts.</li> </ol>
<b>Design Rationale</b>	<p>Illiquidity means the end of the venture; therefore, solvency &amp; liquidity (i.e. make sure you will be able to pay the bills in the long and short term) are prerequisites for business continuity.</p>
<b>Related Work</b>	<p>A typical death valley curve, which shows the effect of initial losses on the finances of new ventures, can be found in [58].</p>



### 3.3.3 Best-Base-Worst Case Scenario

Topic	Content
<b>Problem Context</b>	How to let potential investors deal with the uncertainty that is inherent to the future? When a business plan has been created and the financial returns (i.e. INCENTIVES) for investors need to be estimated.
<b>Forces Solution</b>	Volatility of outcomes is hard to asses since not all volatility causing factors are known beforehand. Identify as much volatility causing factors as possible and estimate their aggregated potential positive and negative effect on the financial returns the venture can generate. Use these estimates to create a best, base and worst case scenario and if possible, calculate an expected return with these scenario outcomes and their respective likeliness.
<b>Resulting Context</b>	One cannot account for unknown factors; therefore it is sufficient to account for all factors that can be identified, as investors know that the financial outcomes are uncertain and want to know in the first place whether the entrepreneur has carefully thought the plan and its potential implications through.
<b>Design Rationale Related Work</b>	No matter how meticulously a business initiative is planned, its predestined interactions with an uncertain environment will effect in uncertain outcomes. Typical use of financial scenarios is demonstrated in [59].

### 3.4 Finding Funding

Topic	Content
<b>Problem Context</b>	How to convince investors to contribute to the success of your business? When starting an enterprise, you make a lot of cost before you can start reaping the benefits. The team member can probably not provide sufficient funding themselves.
<b>Forces</b>	Several investor profiles exist, each with objectives (e.g. expected return, duration and scale of participation) and expertise. But be aware that every investor is in the business for its own profit.
<b>Solution</b>	Create a clear and attractive document (i.e. business plan) that can serve as an implicit contract (e.g. concerning expected returns, management style) between investor and entrepreneur.
<b>Resulting Context</b>	<p>The liability mix (i.e. equity capital vs. debt), which is largely dependent on the origin of the funds (e.g. banks, professional investors, stock market, friends, fools and family) is mainly determined by the business lifecycle (i.e. seed, start-up, expansion, exit):</p> <p>Initial funding (e.g. seed money, start-up money) is often provided by the initiators, ‘friends, fools and family’ and potentially venture capitalists and business angels. Once a business is more established, professional investors (e.g. banks, leasing companies, venture capitalists, stock markets) play a more important role providing expansion and exit money. In specific cases, authorities can also provide money.</p> <p>Professional investors (e.g. banks) often require collateral securities or high risk premiums and sometimes joint management. Relatives provide money at favorable conditions but mostly have limited resources. Business angels can provide additional support and relevant experience. The longer the history of the enterprise, the lower the risk premium.</p> <p>Matching investor profile with the information provided in the business plan, will avoid unnecessary reiterations of the business plan or unnecessary rejections because the plan is deemed unsatisfying.</p> <p>The underlying pattern language for FINDING FUNDING then addresses techniques to address an AUDIENCE with ATTRACTIVE DOCUMENTS and ATTRACTIVE PRESENTATIONS. (fig. 6)</p>
<b>Design Rationale Related Work</b>	<p>The balance sheet of an enterprise consists of assets and an equal amount of liabilities (i.e. equity and debt). Consequently, an enterprise requires funding in each phase of its lifecycle.</p> <p>The ROADMAP THROUGH DEATH VALLEY pattern provides the venture profile that can be matched with investor profiles.</p> <p>Timmons and Spinelli [58] provide an overview of the cost of capital and typical amount of funding per type of investor and shows typical capital structures during an enterprise’s lifecycle. Kotler [60] discusses a marketing approach to finding funding.</p>

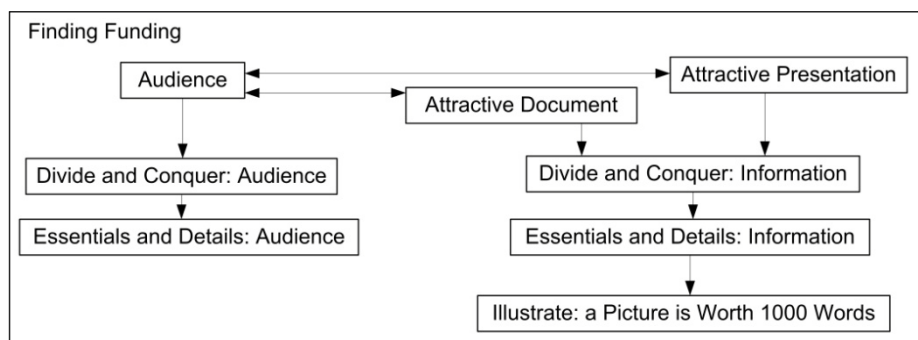


Fig. 6. Structure of the Finding Funding pattern language

### 3.4.1 Audience

Topic	Content
<b>Problem</b>	How to find your audience?
<b>Context</b>	When you have an idea of what you want to achieve and how to achieve it, but do not know who to target to move your idea forward.
<b>Forces</b>	<ol style="list-style-type: none"> <li>1. There is a competition to get in touch with people that can really help you, because they are scarce and can also help others.</li> <li>2. People that have managed to get in touch with them will fence their valuable connections off.</li> </ol>
<b>Solution</b>	Talk to as many people as possible, take your time to find out whether people are interesting, move on when they are not and go back to places where you meet interesting people.
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. When you visit the right locations often, people will know your ideas and tell you who you need to talk to.</li> <li>2. When people think they know you, their motivation to fence of people decreases.</li> </ol>
<b>Design Rationale</b>	Inventors are often so preoccupied <sup>3</sup> with their idea that they do not value a wide and diverse social network until they really need it.
<b>Related Work</b>	The INTROVERT-EXTROVERT pattern [61] helps introverted people to find the right ‘tone of voice’ for addressing an audience, The DOOR OPENER introduces them to the right audience. Social networks in cyberspace are a hot topic.[43, 62]

### 3.4.2 Divide and Conquer: Audience

Topic	Content
<b>Problem</b>	How large should an audience be?
<b>Context</b>	When you have a message that you want to communicate to an audience.
<b>Forces</b>	Finding an optimal size or granularity for groups of people is a complex task, since there is a trade-off between economies of scale (larger audiences generally require fewer resources per capita) and the specificity of your message, which follows your success rate (specific information has a higher success rate but destined for a smaller audience, generic information has a lower success rate but is destined for a larger audience).
<b>Solution</b>	Match the required resources (i.e. cost) per capita with the expected value per capita, and make sure that the expected value per capita exceeds the cost per capita.
<b>Resulting Context</b>	<p>Valuable audiences are divided in smaller groups and provided them with the specific information they want. The higher cost per capita is compensated with a higher success rate.</p> <p>Less valuable audiences are divided in larger groups and provided with more generic information. The lower success rate is compensated with a lower cost per capita.</p>
<b>Design Rationale</b>	You cannot convince the entire world at once.
<b>Related Work</b>	<p>Kotler [36] lists several Market (i.e. audience) Segmentation criteria.</p> <p>This pattern has a dual pattern (i.e. DIVIDE AND CONQUER: INFORMATION)</p>

<sup>3</sup> Some are convinced that good wine needs no bush; others are just shy or scared that someone will steal their idea.

### 3.4.3 Essential & Details: Audience

Topic	Content
<b>Problem</b>	How to structure large groups of people?
<b>Context</b>	When you have an audience that is too large to address at once.
<b>Forces</b>	Large groups of people are hard to manage and address, as the diversity of the group and hence the heterogeneity of its interests increases, which requires the generality of the information to increase, with the number of members.
<b>Solution</b>	Discriminate essential (e.g. most rewarding prospects) from peripheral people (e.g. least rewarding prospects) and focus (your message) on the essential people (first).
<b>Resulting Context</b>	This essentiality criterion defines a priority in which groups should be addressed.
<b>Design Rationale</b>	Too little essential people may lead to inappropriate omissions, leading to missed opportunities.
<b>Related Work</b>	Too many essentials may dilute focus and cause the message to be too generic. This pattern has a dual pattern (i.e. ESSENTIALS AND DETAILS: INFORMATION)

### 3.4.4 Attractive Document

Topic	Content
<b>Problem</b>	How to motivate people to process as much of the information contained in your documents as possible.
<b>Context</b>	When you are creating documents (e.g. business plan, name card) that will be distributed among your (potential) stakeholders (e.g. investors, customers, suppliers).
<b>Forces</b>	1. Reading your documents is often not the reader's priority. 2. Many fonts look attractive but prove hard to read.
<b>Solution</b>	Create well-structured documents with sober but efficient and consistent use of lay-out elements. <ul style="list-style-type: none"> <li>• Limit the number of fonts used in the document, use them consistently (e.g. one for titles, one for text, one for captions, one for sidebars ...)</li> <li>• Be careful (and sparing) with the use of italic and bold.</li> <li>• Use professional looking fonts (e.g. Arial, Times)[63]</li> <li>• Use between 45 and 65 characters per line[64]</li> <li>• Use left-justified text [64]</li> <li>• Use serif fonts for text and sans-serif fonts for titles [64]</li> <li>• Use sparse line spacing [65]</li> <li>• Avoid having a short line (e.g. one word) at the end of a paragraph.[63]</li> </ul>
<b>Resulting Context</b>	1. When documents are attractive, the perceived effort of processing them is reduced, which means that this attractiveness is a kind of intrinsic INCENTIVE. 2. When the body of the document uses fonts that are especially designed for readability, and fancy fonts are used for highlighting certain parts of the document (e.g. titles, side boxes), the reading efficiency will increase. However, dyslectic people seem to prefer sans-serif fonts [66].
<b>Design Rationale</b>	You can improve your chances to get your message across by making your presentations superior. It won't make your arguments better, but it will ensure that readers grasp and retain your points more easily. That's a valuable advantage, which you should seize.[64]
<b>Related Work</b>	Business Plan Templates [50] show examples of business plans with a structure and content that is typical for a certain type of venture.

### 3.4.5 Attractive Presentation

Topic	Content
<b>Problem Context</b>	How to motivate people to be attentive during the entire presentation? When you are preparing presentations (e.g. business plan) for (potential) stakeholders (e.g. investors, customers, suppliers).
<b>Forces</b>	Although a business plan largely consists of detailed technical and financial information, an oral presentation is not ideally fit for presenting detailed information.
<b>Solution</b>	Sketch the plan in broad outlines (i.e. idea's and bottom line) in the presentation, save the details for the question round after the presentation and prepare (slides) for questions. Create well-structured presentation with sober but efficient and consistent use of lay-out elements [67] and manage your body language. <ul style="list-style-type: none"> <li>• Be enthusiastic</li> <li>• Know your audience and highlight important points for them</li> <li>• Limit the number of words per slide</li> <li>• Tell people three times (i.e. tell them what you will tell, tell it, tell them what you have told)</li> <li>• Remind the audience of the presentation structure and show the advances in the narrative thread (e.g. by repeating the presentation outline and highlighting what which topics have been addressed).</li> </ul>
<b>Resulting Context</b>	<ol style="list-style-type: none"> <li>1. An enthusiastic presenter may motivate people to be attentive</li> <li>2. Reminding people of the presentation structure and highlights may help to regain the attention of people whose attention you have lost.</li> <li>3. A sober presentation puts the emphasis on the content</li> </ol>
<b>Design Rationale</b>	The audience needs to be attentive, to be susceptible to your message.
<b>Related Work</b>	PATTERNS FOR TEACHING SOFTWARE IN CLASSROOM [68] addresses a specific kind of presentation skills (i.e. teaching software) for which certain guidelines (e.g. SHOW IT RUNNING) can be generalized for other presentation types (e.g. presentation of a product). A PATTERN LANGUAGE FOR SCREENCASTING [69] focuses on a particular type of presentation.

### 3.4.6 Divide & Conquer: Information

Topic	Content
<b>Problem</b>	How communicate large amounts of (complex) information to an audience?
<b>Context</b>	When you are creating a message that you intend to communicate to an audience.
<b>Forces</b>	Finding an optimal size or granularity for information is a complex task as there is a trade-off between the quality of the content and the size of the message, which influences the reception of the message negatively.
<b>Solution</b>	Use short and simple sentence structures, structure your information clearly (i.e. create a table of content and use section and subsection titles), bring every message (i.e. piece of information) as concise as possible without being incomplete and avoid messages that are too short (e.g. half/double the size of the average message in the document) as they may indicate bad structure (e.g. belongs is part of another message) or incompleteness (e.g. the idea is not fully developed).
<b>Resulting Context</b>	A clear document structure allows you to represent all required information in the document and divide it in small and effective messages. The structure also allows the reader to look for (or skip) messages that are (not) relevant.
<b>Design Rationale</b>	Transferring information is the essence of every type of communication.
<b>Related Work</b>	This pattern has a dual pattern (i.e. DIVIDE AND CONQUER: AUDIENCE)

### 3.4.7 Essentials and Details: Information

Topic	Content
<b>Problem</b>	How to structure large pieces of information?
<b>Context</b>	When you have a message that is too large to be communicated at once.
<b>Forces</b>	Large messages are hard to communicate, as they require more effort from the recipient, which may hamper communication through fatigue. On the other hand, it is hard to substantiate convincing arguments in a short message.
<b>Solution</b>	Discriminate essential from peripheral information and communicate the essential information first. (e.g. bring the essential information in the first section of a text or the first sentence of a paragraph).
<b>Resulting Context</b>	Bringing essential information first allows people to scan a document more easily, without losing sight of the overall message of that document.
<b>Design Rationale</b>	Large messages require more effort from the recipient; hence a larger incentive is required.
<b>Related Work</b>	This pattern has a dual pattern (i.e. ESSENTIALS AND DETAILS: INFORMATION). The executive summary (i.e. a 1-5 page summary of a document), which is an essential part of a business plan [50], is a typical example of the bundling of essential information.

### 3.4.8 Illustrate: A picture is worth 1000 words

Topic	Content
<b>Problem</b>	How to make documents and presentations visually attractive?
<b>Context</b>	When you want your message to make a lasting impression.
<b>Forces</b>	Documents that contain too much solid text may appear dull, whereas text that contains too many pictures, diagrams and tables loses its consistency
<b>Solution</b>	Include relevant diagrams, tables and pictures to illustrate your text, but try to avoid having more than one illustration per page and having two facing pages of text without illustration.
<b>Resulting Context</b>	Having graphics to illustrate a text is insufficient to increase attractiveness; the graphics also need to be perceived as attractive and relevant.
<b>Design Rationale</b>	Some information is hard to communicate with text and numbers (e.g. the design of your product).
<b>Related Work</b>	The TURNING ME ON TURNING ME OFF pattern [70] provides guidelines for interactive graphics. Moody [71] describes essential characteristics of good diagrams.

#### Acknowledgements

An earlier version of this paper was presented at the 14<sup>th</sup> European Conference on Pattern Languages of Programs (EuroPLoP). We are grateful for the comments of the writing workshop participant. Especially, Allan Kelly, Lise Hvatum, Nora Ludewig, Albena Antonova, Elissaveta Gourova and Lisa ... We also like to thank Andreas Fiesser for his efforts shepherding prior versions of this paper.

#### References

1. Kelly, A.: Business Patterns for Product Development. EuroPLoP 2008, Irsee, Germany (2008)
2. Rising, L., King, C.: Patterns for Building a Beautiful Company. (2002)
3. Coplien, J.O.: A generative development-process pattern language. Pattern languages of program design. ACM Press/Addison-Wesley Publishing Co. (1995) 183-237
4. Coplien, J.: A Development Process Generative Pattern Language. PLoP/94, Monticello (1994)
5. Georgiakakis, P., Retalis, S.: Design Patterns for Creativity. 14th European Conference on Pattern Languages of Programs, Irsee, Germany (2009)
6. Ball, L.J., Ormerod, T.C., Morley, N.J.: Spontaneous analogising in engineering design: a comparative analysis of experts and novices. Design Studies **25** (2004) 495-508
7. Kelly, A.: Design Patterns for Software Companies (Product Development). VikingPLoP 2007, Bergen, Norway (2007)
8. Kelly, A.: Design Patterns for Software Companies (Product Development). EuroPLoP 2007, Irsee, Germany (2007)
9. Kelly, A.: Design Patterns for Technology Companies. EuroPLoP 2006, Irsee, Germany (2006)
10. Kelly, A.: Business Strategy Design Patterns for Technology Companies. VikingPLoP 2005, Espoo, Finland (2005)
11. Kelly, A.: A few more business design patterns. EuroPLoP 2005, Irsee, Germany (2005)
12. Kelly, A.: Business Strategy Pattern for the Innovative Company. VikingPLoP 2004, Uppsala, Sweden (2004)
13. Kelly, A.: The Porter Patterns. EuroPLoP 2004, Irsee, Germany (2004)
14. Ijiri, Y.: Theory of accounting measurement. American Accounting Association, Sarasota, Fla. (1975)
15. Gordijn, J.: Value-based requirements Engineering: Exploring innovative e-commerce ideas. Exact Sciences, Vol. Phd. Free University of Amsterdam, Amsterdam (2002) 292
16. Kelly, A.: A Pattern vocabulary for product distribution. 14th European Conference on Pattern Languages of Programs, Irsee, Germany (2009)
17. Mahler, A., Rogers, E.M.: The diffusion of interactive communication innovations and the critical mass: the adoption of telecommunications services by German banks. Telecommunications Policy **23** (1999) 719-740
18. Malcol, D.G., Roseboom, J.H., Clark, C.E., Fazar, W.: APPLICATION OF A TECHNIQUE FOR RESEARCH AND DEVELOPMENT PROGRAM EVALUATION. Operations Research **7** (1959) 646
19. Wilson, J.M.: Gantt charts: A centenary appreciation. European Journal of Operational Research **149** (2003) 430-437
20. Jones, C.V.: THE THREE-DIMENSIONAL GANTT CHART. Operations Research **36** (1988) 891
21. Gantt, H.L.: Organizing for work. Harcourt, Brace and Howe, New York, (1919)
22. Brandenburger, A., Nalebuff, B.: Co-opetition. Doubleday, New York (1996)

23. Weiss, M.: In Bed with the Enemy. In: Hvatum, L., Schummer, T. (eds.): 12th European Conference on Pattern Languages of Programs. Universitätsverlag Konstanz, Irsee, Germany (2007) 159-171
24. Rising, L.: Customer Interaction Patterns. (1997)
25. Porter, M.E.: THE FIVE COMPETITIVE FORCES THAT SHAPE STRATEGY. Harvard Business Review **86** (2008) 78-93
26. Gilman, G.: The manager and the systems concept. Business Horizons **12** (1969) 19-28
27. McNutt, K.: SWOT Before You Start. Nutrition Today **26** (1991) 48-51
28. Larkin, J.H., Simon, H.A.: Why a Diagram is (Sometimes) Worth Ten Thousand Words. Cognitive Science **11** (1987) 65-100
29. Porter, M.E., Millar, V.E.: How information gives you competitive advantage. Harvard Business Review **63** (1985) 149-160
30. Peters, T.J., Waterman, R.H.: In search of excellence : lessons from America's best-run companies. Harper and Row, New York (N.Y.) (1982)
31. Rogers, E.M.: Diffusion of innovations. Free Press of Glencoe, New York, (1962)
32. Day, G.S.: Diagnosing the Product Portfolio. Journal of Marketing **41** (1977) 29-38
33. Kerin, R.A., Varadarajan, P.R., Mahajan, V.: Contemporary perspectives on strategic market planning. Allyn and Bacon, Boston (Mass.) (1990)
34. Ansoff, H.I.: Strategies for Diversification. Harvard Business Review **35** (1957) 113-124
35. Kotler, P., Keller, K.L.: Marketing management. Pearson Prentice Hall, Upper Saddle River, N.J. (2009)
36. Kotler, P., Armstrong, G.: Principles of marketing. Pearson/Prentice Hall, Upper Saddle River, N.J. (2008)
37. Haase, M., Miedl, M.: Patterns for Leading Effective and Efficient Meetings. In: Hvatum, L., Schummer, T. (eds.): 12th European Conference on Pattern Languages of Programs. Universitätsverlag Konstanz, Irsee, Germany (2007) 53-95
38. Haase, M.: Patterns for Leading Effective Meetings. 10th European Conference on Pattern Languages of Programs, Irsee, Germany (2005)
39. Iba, T., Miyake, T., Naruse, M., Yotsumoto, N.: Learning Patterns: A Pattern Language for Active Learners. 16th Conference on Pattern Languages of Programs, Chicago, Illinois, USA (2009)
40. Kile, J.F., Little, D.J., Shah, S.: Busy Person Patterns. 13th Conference on Pattern Languages of Programs, Portland, OR USA (2006)
41. Strong, E.K.: The Psychology of Selling and Advertising. pp. xi. 468. McGraw-Hill Book Co.: New York (1925)
42. Lavidge, R.J., Steiner, G.A.: A Model for Predictive Measurements of Advertising Effectiveness. Journal of Marketing **25** (1961) 59-62
43. Bardon, D.: online SOCIAL networking for business. Online, Vol. 28. Information Today Inc. (2004) 25-28
44. Williams, M., Kochhar, A., Tennant, C.: An object-oriented reference model of the fuzzy front end of the new product introduction process. International Journal of Advanced Manufacturing Technology **34** (2007) 826-841
45. Tips on CV writing. Travel Trade Gazette UK & Ireland (2008) 95-95
46. DO IT RIGHT: JAZZ UP YOUR CV. Management Today (2009) 16-16
47. Ellis, W.D.: A Source Book of Gestalt Psychology. Prepared by W. D. Ellis, etc. [By various authors.]. pp. xiv. 403. Kegan Paul & Co.: London (1938)
48. Smith, B., Ehrenfels, C.: Foundations of Gestalt theory. Philosophia Verlag, München (1988)
49. Chaturvedi, M.: Team where People Matters - A Project Management Pattern. In: Hvatum, L., Schummer, T. (eds.): 12th European Conference of Pattern Languages of Programs. Universitätsverlag Konstanz GmbH, Irsee, Germany (2007) 149-157
50. SBA: How To Write a Business Plan. U.S. Small Business Administration (2008)
51. Cockburn, A.: Toward a risk management catalog. Computer **29** (1996) 28-30
52. Myers, B.L., Melcher, A.J.: ON THE CHOICE OF RISK LEVELS IN MANAGERIAL DECISION-MAKING. Management Science **16** (1969) B-31-B-39
53. Maslow, A.H.: Motivation and personality. Harper, New York, (1954)
54. Herzberg, F.: The managerial choice : to be efficient and to be human. Dow Jones-Irwin, Homewood, Ill. (1976)
55. Herzberg, F.: The motivation to work. Wiley, New York, (1959)
56. Herzberg, F.: One more time : how do you motivate employees? Harvard Business Press, Boston, Mass. (2008)
57. Riedle, H., Beckmann, M.: Hermann Heinrich Gossen, 1810-1858. Ein Wegbereiter der modernen ökonomischen Theorie. Econometrica: Journal of the Econometric Society **23** (1955) 223-224
58. Timmons, J.A., Spinelli, S.: New venture creation : entrepreneurship for the 21st century. McGraw-Hill/Irwin, Boston, Mass. (2007)
59. Sahlman, W.A.: Note on Free Cash Flow Valuation. Harvard Business Publishing (1987) 43
60. Kotler, P., Kartajaya, H., Young, S.D.: Attracting investors : a marketing approach to finding funds for your business. Wiley, Hoboken, New Jersey (2004)
61. Bergin, J.: Introvert-Extrovert. 7th European Conference on Pattern Languages of Programs, Irsee, Germany (2002)



62. Athitakis, M.: The Right Connection. *Associations Now*, Vol. 5. American Society of Association Executives (2009) 22-25
63. Smasal, C.: Graphic Design At Work. *Office Pro* **67** (2007) 11-14
64. Barth, S.: Digital Designs. *EContent* **31** (2008) 32-36
65. Moriarty Sandra, E., Scheiner Edward, C.: A Study of Close-Set Text Type. *Journal of Applied Psychology* **69** (1984) 700-703
66. Litterick, I.: Typefaces for Dyslexia. Vol. 2009. *dyslexic.com* (2003)
67. Tessler, F.N.: Polish Your Presentations. *Macworld* **23** (2006) 68-69
68. Schmolitzky, A.: Patterns for Teaching Software in Classroom. In: Hvatum, L., Schummer, T. (eds.): 12th European Conference on Pattern Languages of Programming. Universitätsverlag Konstanz GmbH, Irsee, Germany (2007) 37-51
69. Chen, N., Rabb, M.: A Pattern Language for Screencasting. 16th Conference on Pattern Languages of Programming, Chicago, Illinois, USA (2009)
70. Kohls, C., Windbrake, T.: Turning me on, turning me off. 13th European Conference on Pattern Languages of Programs, Irsee, Germany (2008)
71. Moody, D.: What Makes a Good Diagram? Improving the Cognitive Effectiveness of Diagrams in IS Development. *Advances in Information Systems Development* (2007) 481-492

# Software Design Reviews

December 31st, 2009

Nora Ludewig  
(nludewig@gmx.de)

© 2009 Nora Ludewig. Copyright retained by author.  
Permission granted to Hillside Europe for inclusion  
in the CEUR archive of conference proceedings and  
for Hillside Europe website.

---

## Introduction

This paper describes patterns for software design reviews.

The experiences leading to these patterns were made in the context of software development for embedded systems. However, they contain practices that can be useful in every kind of software design review. I became aware of these patterns while I was learning to apply the review method DRBFM to software development. DRBFM stands for Design Review Based on Failure Mode and is a review method invented by Toyota and now slowly spreading into other organizations. Initially, it was developed for proving the robust design of mechanical products. But it turns out that DRBFM is a powerful way to look at software designs, too.

DRBFM focuses on the separation between functionality and solution. The method helps you to check whether a product still fullfills the requirements in a robust way when you implement changes in one or several parts of the product. Interactions between different parts are looked at in particular.

Software for embedded systems typically is connected very closely to the “real world”. There are sensor signals that have to be transformed into numerical values, maybe the system has a processor working with integer arithmetics, and the system always has time constraints because there are limited reaction times specified. Due to this close connection of software and hardware there are many constraints on the software such as accuracy of the conversion of sensor signals or runtime and memory consumption. In the design review the software must not be analysed as an isolated product but as one part of a system.

In the field of embedded software development, reviews are often held after every development phase. There are reviews of the specification, of the functional design, of the code, of the software and system tests. All the different review methods are based on the assumption that it is helpful if someone different from the developer examines a product and brings in their expertise. Therefore the main parameter defining the success of the review is

the knowledge, the communication and the behaviour of the review participants [1]. This paper describes how you as a developer can contribute to the success when you are preparing, organizing and moderating a software design review.

In some of the patterns, I talk about “changes” or “modifications”. This results from the approach of DRBFM always comparing a product as it was before with a new (“changed” or “modified”) solution. So I assume there is always some basic version of a product, artifact or piece of code to start with. We are not re-inventing the software from the beginning, but there is some comparable product or some frame we can start our work from.

The thirteen patterns are divided into four groups:

<b>Deciding what to review</b>	Risk over Size
<b>Preparing the Review</b>	Continuously Prepare the Review Review the Final Design Don't Think - Describe your Solution! Make Implicit Requirements Explicit Look at the System Free Requirements from Solution
<b>Running the Review</b>	You are the Entertainer Every Question is Okay Do not Justify, just Listen Do it your Way - Don't stupidly fill in Templates It is not about Form
<b>After the Review</b>	Make the Outcome Concrete

With these patterns I would like to address developers, engineers and students working in embedded software development. There is a strong focus on the developer who is preparing and running the review. Nevertheless they can be helpful to review participants, too. Also they could be interesting for managers who want to improve the quality and the output of design reviews performed in their organisation.

---

## Deciding what to review

### Risk over Size

**Context:** You are designing a new software part or a change in an existing part and a design review is required by the process.

**Forces:** In software development, not only the final product counts but also the development time and effort. In most cases, developers have to make a trade off between quality and adherence to deadlines. The availability of time and expertise or the mere size of a software module determine the content and the intensity of reviews. So a rather common algorithm might be looked at very carefully while a very innovative software part might be more or less neglected.

The developer of new software usually has some influence on the intensity and the extent of the review that is done on the design. But if he sets the focus improperly, maybe big risks will be overlooked.

**Problem:** When you have to focus your review effort due to time or capacity restrictions, which parts of the development should you consider in the review?

**Solution:** Find out where is the biggest risk and put most of your effort on this part.

**Implementation:** A design resulting in a high number of lines of code is not automatically riskier than a small change in an existing software part. You should watch out for changes with a high degree of innovation or changes in complex interfaces. [2]

If you have used a commonly known solution or you have developed a module similar to one you have developed before, the risk is not as high as there is for completely new algorithms or changes in interfaces with cross connections to many other parts of the software.

Of course it is difficult to find an impartial technique leading to decisions comparable to decisions in other reviews. It is hard to measure risk which does not only depend on the properties of the software but also on the properties of the developer and his or her environment. By discussions with colleagues about your development and how it impacts the system, you will develop a sense of how intense the review should be.

**Examples: 1.** You are using a filter algorithm looking very complicated at first sight. The code for it is rather big. But in fact you have a lot of experience with this algorithm because you have used it many times before. Moreover the algorithm does not have many boundary conditions or dependencies. Therefore the review of this part of the design does not have to be elaborate.

2. You are changing an interface used as an input for a timer from a 32-bit-variable to a 16-bit variable. Maybe you have tested the new function in the target environment and you have shown that the behaviour is the same as before. Unfortunately you did not consider the fact that now there is a variable overflow much faster and that then the timer will fail. This example shows that especially when there are changes in interfaces it is good to review with a rather high effort.

---

## Preparing the Review

### Continuously Prepare the Review

**Context:** You are planning all the different tasks of the design phase, including the design review. You want to allocate time for the review preparation.

**Forces:** The review often is perceived as a point in time. That is the point when the review meeting takes place.

During the design phase, you have to keep in mind all the requirements and constraints to your product as well as the process you have to follow. Preparing for the review is another task you have to plan for during the design phase. The review preparation will consume a certain amount of time and energy in addition to the core design tasks. For this reason, review preparation often is neglected.

If you take care of the review only very shortly before the review meeting, you will prepare and organize the review in a hurry, forgetting about important points, inviting the wrong people, neglecting presentation rules.

Consequences of these forces are unprepared reviews, unsatisfying results and unmotivated participants.

**Problem:** When should you start to prepare the design review and how should you allocate the preparation efforts?

**Solution:** Start preparing the review from the beginning of the design phase and continue all through the design process.

**Implementation:** Take notes about concerns or questions you have. Talk to the colleagues you would like to invite to the review. Prepare the information you want to give to the review participants before or during the review.

Set the date for the review meeting as soon as you know when you will have finished the design. REVIEW THE FINAL DESIGN and plan some time after the review for doing some rework if necessary. Early planning will increase the chance that you get the experts you want to have in the meeting and they will have some time to prepare for the review.

**Example:** You design a software module for the control of a machine. This module has the engine speed as an input variable. In fact there are two variables representing the engine speed available in the software. A raw signal NEng\_unfilt and a filtered signal NEng\_filt.

At the beginning of the design you are not sure yet which signal is best for your application. You are taking the raw signal for the first draft version. As you have started to prepare for the review from the very beginning of the design phase, you always have in mind that you need to clarify this point until the design is finished. So there is no risk that the first choice for the draft version will stay in the solution just by chance.

Moreover, when you present the input values at the review meeting, you can say which variable for the engine speed you have chosen and, very important, why.

## Review the Final Design

**Context:** You want to set the time for the review meeting.

**Forces:** A review meeting held at the wrong time can lead either to discussions about what is really the concrete solution and by this to a kind of developers workshop. Or it can lead to a hurried meeting where the developer tries to dismiss as many points as possible in order to avoid rework.

If you hold the review meeting too early, there is a danger that not all the necessary information will be available or some details even are not defined yet. You will have to review alternatives or theoretical concepts. This can result in endless discussions.

On the other hand, if the review meeting is held too late, you will not have the time to implement the measures you have decided on in the meeting. Maybe you will even try to avoid some findings in the meeting because you know that will not have the time to do the rework identified to reduce the risks.

**Problem:** When is the best time to hold the review meeting?

**Solution:** Hold the review meeting only after you have collected all the facts the review participants need to examine the design but keep enough time after the review to change the design where necessary.

**Implementation:** As you CONTINUOUSLY PREPARE THE REVIEW, you can look for the best time for the review meeting from the beginning of the development. You can already talk with the colleagues you would like to participate in the meeting so that you find out when persons that are very important for the review success are on vacation et cetera.

As the delivery date normally is fixed, you can estimate how much rework will be necessary after the review in the worst case and then you can calculate the date when the meeting has to be held the latest [3]. The review participants also need some time for the preparation of the review. Taking this into consideration you know when your design has to be finished. So by calculating backwards from the delivery date you make sure that you have enough time for the review and the rework and that you also go into the review with a real design and not only a design idea.

**Example:** Here you can look at the same example as in CONTINUOUSLY PREPARE THE REVIEW. Imagine you are still not sure yet whether you should take the filtered or the unfiltered signal as an input value. Therefore you do not put NEng\_filt or NEng\_unfilt into your design, you just write down NEng. If you are doing the review now, there is a high risk that some of the participants will have NEng\_filt in mind and some will think about NEng\_unfilt. They will be talking about different things without noticing the problem. If they notice this

open point, the risk is high that the review will not be a design review on the complete design but a discussion about which of the two variables would be the better one for your application. Of course this might be a helpful discussion but it is not the goal of the review.

## Don't Think – Describe your Solution!

**Context:** You are preparing the information **about your design** you want to give to the review participants before and during the review meeting.

**Forces:** When developing software, you do not just write code. You also define new interfaces, your software consumes memory and clock cycles in the processor, you might call library functions, maybe you use new development tools. This is often overlooked. As a developer you tend to focus on the changes you consider to be important. Changes that do not affect the functionality are easily ignored. However, things that seem unimportant to you might affect the overall system or parts of the system interacting with your module.

But you want to avoid to drown important changes in unimportant “stuff” and you think that a comprehensive list of changes might overwhelm the review participants. So your idea is to make a selection of the most important aspects in advance.

**Problem:** When collecting the information the reviewers need for analyzing your development, how do you decide on the relevance of the information?

**Solution:** When preparing for the review, describe every part of your solution, even if it seems unimportant to you. Avoid any judgement on the importance or the risks of individual aspects. Try to be as neutral as possible.

**Implementation:** A checklist can be really helpful to identify all the changes that were made.

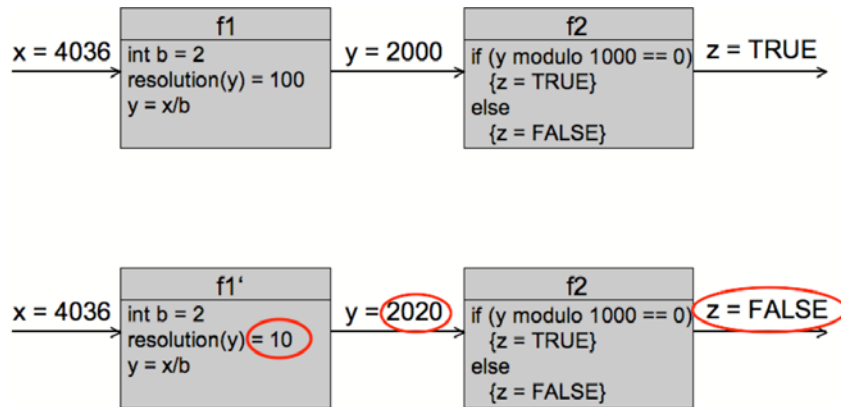
If you have filled out a checklist before, you can show this checklist to the review participants. If there is a very high number of changes, you can, later in the review meeting, make suggestions to put some details aside or summarize some minor changes in one common heading. But you have to justify why it is safe to do so to the review participants.

If safety is a concern, you should rather err on the side of caution and mention too many changes rather than too few.

**Example:** The precision of the output value of a function is increased while the functionality itself remains unchanged. To the developer, this change seems to be unimportant, as his function is now providing a more precise value than before. Unfortunately, there is another function which uses this value as an input and performs a modulo operation on it. As now the resolution is higher, the results of the modulo operation will be completely different. The function fails.



If in the review this change had been mentioned, the potentially fatal consequences could have been found.



## Make Implicit Requirements Explicit

**Context:** You are preparing the information **about the requirements** to the design you want to give to the review participants before and during the review meeting.

**Forces:** When judging whether a product meets the requirements everybody makes his own assumptions about implicit requirements

You have designed your new software module based on the specification of the customer. The customer wrote the specification from his point of view. Often, he has no information about the processes or constraints in your company. He might not know all the technical details of the system your software will be integrated into. But these details can have significant influence on your choices of specific solutions during the design phase.

On the other hand, the customer has his own processes and constraints. His company's philosophy might be completely different from the philosophy of your company, a fact that you, both the customer and the developer, often are not fully aware of.

For these reasons you can almost be sure that the requirements are not complete from your point of view. You will design the software based on the requirements you got from the customer, completed with assumptions on what the customer wants to have but did not say explicitly.

**Problem:** How can you make sure that you as the developer, the review participants, and the customer have the same understanding of the requirements?

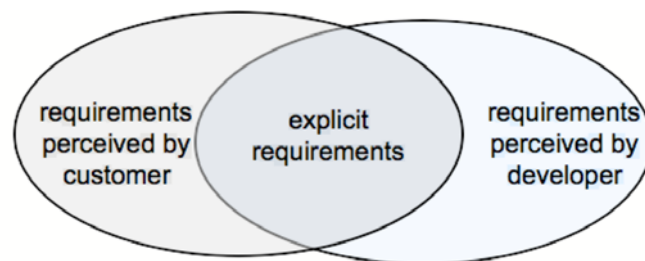
**Solution:** Specify clearly what you think are the requirements to the product even if these requirements were not mentioned explicitly by the customer.

**Implementation:** Put yourself in the place of the customer and think of what he or she wants the product to do or not to do. What are the different pos-

sibilities of using the product and how can they be combined? List all the aspects and circumstances of use. Think about what might be obvious to the customer but not to you. Then, think about what might be obvious to you but not to the customer.

You can take these assumptions as a basis for discussion. Chances are high that in the review meeting there is an expert who can confirm or correct these assumptions.

When you are wondering which requirements are obvious to the customer but not to you, of course it is a good idea to simply ask the customer. But even then, there will be implicit requirements remaining implicit if you do not find them and make them explicit.



When looking for implicit requirements it might be helpful to use checklists. In DRBFM for example, you are invited to find, for each module, the “basic function” which describes the actual purpose of the module, “additional functions” which focus on the on the look and feel of a product, “harm prevention” which avoids that the product can cause harm to people, or the environment and “self protection” which avoids that the module itself is damaged or destroyed.

Of course you should not wait until the design review to talk to the customer about the requirements. If there is information missing at the beginning of the design phase you should get answers to the open points as quickly as possible. Still, the design review is a good opportunity to present and discuss the requirements.

**Example:** The customer ordered a new application for a mobile phone: Photos taken with the camera integrated into the phone can be sorted into different albums. He has given you a specification where all the different possibilities of sorting, opening and modifying the albums are described. As it seems obvious to him, he has not specified the fact that the basic function of the phone, namely the possibility to place and receive calls must not be disabled by the new application. If you do not integrate this requirement into your review, you might only find out that your design is good for sorting photos and so fulfills the requirements you limited the module to.

## Look at the System

**Context:** You are preparing the information **about the environment** of your module you want to give to the review participants before and during the review meeting.

**Forces:** When you are developing only a small part of a complex technical product you tend to find a solution fitting this small part but maybe carrying risks for the whole product. If you do not consider this in the review you will not find these risks.

You know best about all the obstacles you had to overcome and about the reasons why you have chosen this specific solution. Digging deeply into the subject for a long time, you have probably lost an overall view on the product. As a consequence, there is a risk that you have created a locally optimized solution which works well if you regard it for itself but which does not really fit into the greater context. [4]

In software development, especially in large projects, there is a risk that a developer who is working on a specific part of the software for a long time loses sight of the real function of the product.

Moreover, in embedded systems, software is not an end in itself. Often there are many interfaces of one software function to other functions and to sensor signals or device drivers. It is difficult to keep in mind all interactions all the time during development.

**Problem:** **Where should you make the cut when you have to decide how much of your software module's environment you should keep in mind for the review?**

**Solution:** **Ask yourself what is the function of your development for the whole product and document this function during review preparation.**

**Implementation:** When preparing for the review it is important to step back and look at the whole system and at all the interfaces that your software module has with the overall system. These are inputs and outputs as well as environmental conditions.

Find answers to the following questions:

1. Why was there a need for this development?
2. Why did the requirement come up right now?
3. Have there been other attempts to fulfill the requirements? Why were they successful or why did they fail?

**Example:** The software is getting a sensor signal as an input and calculates some output value based on it. In order to find out if the output value meets the accuracy requirements you have to get some information about the accuracy of the input value. You read the technical data sheet of the sensor. But this is not the only information you should take into consideration. You should also ask yourself: Is it possible, in certain circumstances, that the sensor signal does not have the specified accuracy, although the sensor works properly? When might this be possible? For example, due to the signal processing from the raw

signal to the signal received by your module. Maybe there is some filtering or some hysteresis applied you did not take into your consideration. Also: what do I do (in my module) if the accuracy is not as good as I expect it? Can my module deal with that?

## Free Requirements from Solution

**Context:** During review preparation phase, you are describing the requirements to your design.

**Forces:** When thinking about the requirements, a developer never is unbiased. He or she always has the solution in mind. Thus there is a high risk that in the review there will be no clear separation between solution and requirements description.

The problem with this is that if you do not clearly state what your product should do you cannot tell whether the design is applicable or not. Also, if you and the other review participants think into a certain direction from the beginning, you limit the chances that you will find the best possible solution. The review should lead you back to the question what you really need this part of the system for.

Also, boundary conditions are essential because they define the scope you have to look at. Many designs work under certain conditions, but in the review, it has to be checked if these are the conditions that will prevail in the real system.

**Problem:** When you have to describe the requirements the design is to be reviewed against, how can you make sure that you are not biased by the design work you have already done?

**Solution:** Ask yourself and your colleagues what your development should really do. Why was it requested by the customer, what does he expect it to do?

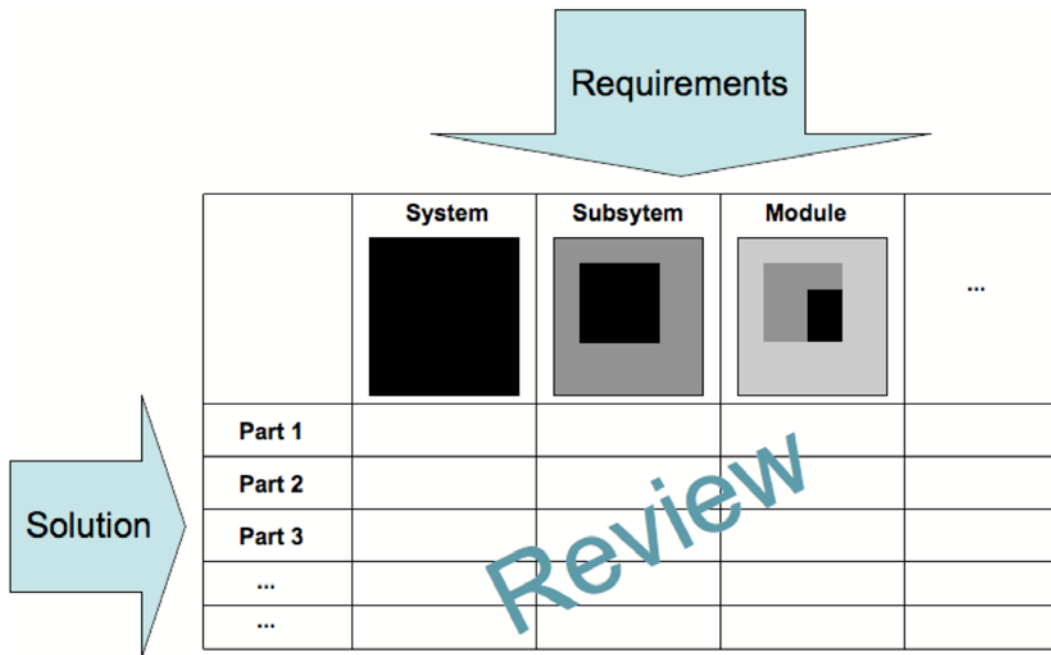
**Implementation:** When describing the requirements, make your product a black box and describe its behaviours seen from outside. Divide your description into two parts:

First, describe the function. What is really the job of the module? How does the customer want your module to behave? What does the system your module will be integrated into “expect” from your module?

Second, describe the boundary conditions. This can be environmental conditions the product still has to work correctly in. It can also be requirements on precision, calculation speed or control quality.

If you have a rather complex design with different parts, it makes sense to break down the requirements into smaller pieces. In other words, you have to open the black box and make several smaller black boxes out of it. This sounds contradictory to the the pattern itself, but it is not as long as you keep the boxes black at each level of granularity you have chosen and you make sure that there is one black box at the same level of your whole design. Having this

highest level, you get the view on the complete design and the interfaces to the system.



**Example:** You are describing the functions of a software part modelling the temperature at a specific location of the system (virtual sensor). This software consists of several subparts. So the overall function would be “calculating temperature at position x”. Describing the function of one of the subparts you might say “signal range check of sensor input S1”. That means you are looking into your top-level black box “temperature calculation” and you are delimiting the solutions to those solutions using S1 as an input. The key point is that for this granularity level you must not say “signal range check with algorithm a” but only “signal range check with boundary conditions c1, c2 and c3”.

---

## Running the Review

### You are the Entertainer

**Context:** You are moderating the review meeting.

**Forces:** Maybe some of the review participants see the review rather as an inevitable evil and so they come into the meeting without any motivation and without any expectation. They probably have attended a lot of boring review meetings before.

There is a big risk that, when some people with this attitude meet, this will turn into a self-fulfilling prophecy. They expect to have a boring meeting and so they make the meeting boring. This blocks all the creativity and concentration and finally leads to nonsatisfying results.

Even if your colleagues come to the review meeting with positive expectations, they can become unmotivated very quickly if you are presenting and moderating the review in a boring or confusing manner.

If you are not able to arouse the review participants' interest for your work, there will be not only a lot of time wasted, but there is also a higher possibility that not all the risks and errors will be found.

**Problem: How can you motivate the review participants to contribute to a good review outcome?**

**Solution: Make the review fun! During the review meeting, you are the entertainer! You are responsible to create an atmosphere where all the review participants are comfortable and really want to contribute to the analysis of the product.**

**Implementation:** At the beginning, you should make sure that all the participants know each other and know you. This helps everybody to get some orientation in the group.

Second you should give a short overview of the review so that everybody knows how long the meeting will take, what will be the steps to take and, very important, what should be the outcome of the meeting.

When you are presenting your work, tell a story to the audience. Give information about the motivation, about your ideas for a solution, about which obstacles you had to overcome, which actions you took in order to make the outcome robust and what is the final result. It does not mean that you should manipulate the information in order to create a breathtaking story or to show what a great developer you are. The story does not have to be thrilling and it does not need to have a happy ending. But it has to be clear, logic and complete. By this you make sure that the review participants can follow. Also it creates an open atmosphere and motivates people to account for the story in order to make it a successful one.

Use the fact that IT IS NOT ABOUT FORM in order to arrange the meeting so that the participants remember it as some very interesting and motivating hours.

**Example:** You have designed a software module for an automobile engine control. For the development you had to do measurements at very low temperatures in a climate chamber in the middle of summer. So you came out of the climate chamber with a complete arctic dress in July.

In the review meeting, show a photo of you in your winter dress to the colleagues. Instantly they will pay attention and they will be inspired to be creative.

## Every Question is Okay

**Context:** During the review meeting, you get a lot of questions from the review participants.

**Forces:** Maybe one of the participants asks a question you or a colleague consider to be trivial or irrelevant. As always, the time for the review is limited, and you want to get to the crucial points quickly. So you are maybe not answering all the questions elaborately.

But there is a high chance that there are more review participants who do not know the answer to the question and who just did not dare to ask.

If not every review participant gets the chance to understand the facts, they will probably have problems to follow the remainder of the review. You will not be able to benefit from their expertise. So you are wasting their time and the outcome of the review will probably not be as good as it could be.

Sometimes, we also tend to declare questions irrelevant, because we unconsciously know that we do not have a clear picture of that point in our mind.

**Problem: How should you deal with questions you perceive to be superfluous?**

**Solution: Answer all the questions and do not (even not implicitly) declare them as stupid or the answer as obvious.**

**Implementation:** You should not make excuses when questions are coming up. If you do not know the answer, be honest and suggest that you will find out after the review meeting. You should mention this task in the action plan in order to MAKE THE OUTCOME CONCRETE.

If you have the impression that the answer to the question is obvious to most of the other review participants, invite them to give the answer.

**Examples:** It is often the basic questions that cannot be completely answered: Are the assumptions made for defining the boundary conditions correct? Are we really sure to have chosen the best algorithm?

## Do not Justify, just Listen

**Context:** You are in the review meeting and you are getting feedback about your work from the review participants.

**Forces:** When you are doing the review, experts from different fields will look on your work from many different points of view. Most of them will probably know that they were invited to the review because they are considered experts in their fields. So they will be proud to give their opinion and to ask you why you did it this way and not that way.

You will have a different perspective on your work. You might think: „How easy is it to question my solution. I have spend long nights at the desk, struggling for finding a solution at all“. In other words, you might feel offended. So you prevent an open discussion and an open mind for the best solution.

**Problem:** How can you make sure that you get the maximum benefit out of the review participants' feedback?

**Solution:** Listen to the feedback carefully and make sure you are getting the point. Do not justify your work or presentation. Appreciate the questions and the advice as they will improve your product and your future work.

**Implementation:** If it was so difficult to find a solution and many boundary conditions have left you no choice, take the questions as a feedback. You should have illustrated the problems and challenges more clearly in your review presentation. Then the experts could have found out if their alternative solution really would have been possible.

Explain as long as there are issues that need clarification for the reviewers, but avoid discussing who is right or why you had no choice but had to do it that way. When participants are starting to repeat remarks and you have the impression that they just want to demonstrate their expertise stop the discussion and continue with the next point.

Give the review participants the feeling that you have got their point. If you have not, ask them to explain their question or remark again.

Keep in mind that, after the review, it is you who is deciding what to do with the opinions of the experts. So you might say, „This time, I won't change my solution as it is still robust. For the next time, I will keep in mind what expert E said.“

So, a design review is not just a method to examine products for their robustness. It is also a way of exchanging ideas, spreading knowledge, building networks and improving communication skills.

**Example:** One of the review participants says, “I don't think that you have identified all the input parameters to your software model. In my opinion, input IN243 has to be added.”

Your answer could be “Thank you for this idea! Could you please explain why you think that this is an important input? Do you know if in model m95 which is similar to mine this input is also used?”



## Do it your Way - Don't stupidly fill in Templates

**Context:** In your company, you have to follow a certain review process. Maybe you have to take into consideration templates, checklists, or fill databases with information about your design.

**Forces:** As you want to fulfill the requirements of the processes, you might spend a lot of energy to force your review into the process.

A defined review process is necessary in larger companies, but it always carries the danger that developers become annoyed about all the administrative work they have to do. So the review participants will fill in all the templates and forms, but not before they have switched off their brain. The problem can also occur when template categories or questions do not match your context.

When you are trying to stick to the rules slavishly you might not be able any more to have a review where the real content and not the form is discussed. So you won't find the risks.

**Problem:** How can you avoid to be paralyzed by your organization's review rules?

**Solution:** Adapt the processes and the templates to your solution and not vice versa. As long as you communicate this clearly and you can give clear the reasons, this is okay.

**Implementation:** Maybe you can change the checklist or template categories, insert new rows, or columns. Adapt the review and its templates to your needs and not vice versa.

If you leave out process steps or some chapters in a template you have to give reasons for this decision. If you add something this is usually not seen as leaving the process but rather as an add-on, so you will not have any problems with this.

This pattern is not an invitation to cheat. Of course you have to stick to the rules. But if you bother to think about the templates and the meaning of the process steps you will find your way to use them as helpful support tools. Moreover you can give feedback to the process developers and help creating a leaner and more useful process in the long run.

**Example:** In your company you have to use a template where you have to fill out a chapter "changes in logic". In fact you have changed something in a model representing the physical behaviour of the system. You do not really find a category where this change fits into and you wonder if this should be put into the chapter "changes in logic". In order not to confuse anybody you add a chapter "changes in physical models" and describe the change there. In the chapter "changes in logic" you put a dash.

## It is not about Form

**Context** Preparing for the review, you have collected all the different pieces of information about the requirements and boundary conditions. Together with your solution, you want to present them to the other review participants. Maybe you have also made some simulations or you even already have a running prototype of the software you are developing.

**Forces:** As YOU ARE THE ENTERTAINER, you want to have something nice to show to the review participants. You want to make sure that every review participant can contribute to the outcome of the review. So you maybe try to get as much information and demonstrations as possible to the review participants.

Structures and relations in large software projects are usually complex and not easy to understand. Even visualisations which were very simple in the beginning can become confusing very quickly.

When you present your work in a perfect form with a lot of „special effects“, it is more difficult for the review participants to really understand what you did and to get in a discussion. They have to „believe“ because they cannot retrace your steps of work.

**Problem: How should you present your work in order to facilitate a creative and open discussion?**

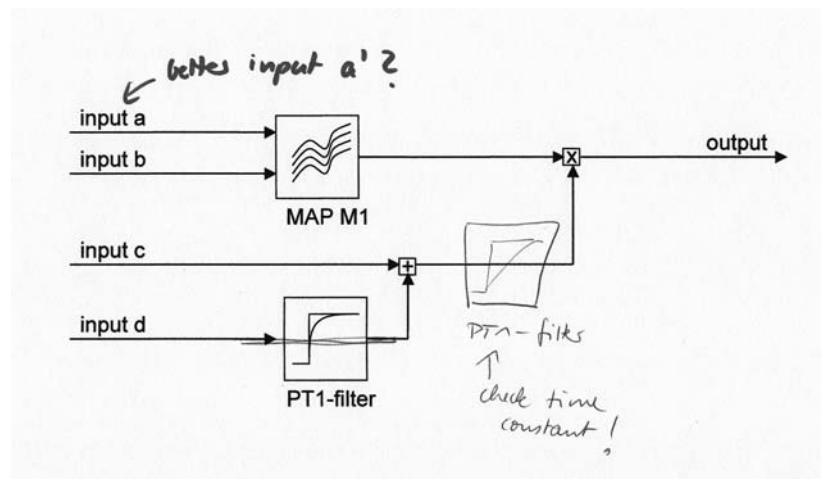
**Solution: Limit technical means and „show effects“ to the really necessary. Do not waste your time with creating perfectly designed presentations, put the time into the SW design instead.**

**Implementation:** It is good to give the audience a motivating start and to visualise as much as possible throughout the review meeting. But although YOU ARE THE ENTERTAINER, you should avoid making the design review a design show. Keep it as simple as possible.

Make drawings on the white board as you explain things instead of letting a power point slide suddenly pop up completely filled.

In many cases you can print out the facts and the figures you want to talk about, hang the printouts to the wall and then discuss with the other review participants while standing in front of them. This might seem to be old-fashioned and requires a little more preparation than taking the laptop to the meeting room, but it gives the review participants the possibility to really interact with each other. If they want to know a number you have presented ten minutes ago, they do not have to ask you to go back in the slides. They can look it up themselves in the printouts. Another advantage is that everybody can take out a pencil and show his ideas to everyone on the printouts. This can be very useful if he or she suggests a small modification in one of the figures.

Example:



---

## After the Review

### Make the Outcome Concrete

**Context:** The review meeting is over. In the meeting you have found some need for rework, for example you have to get more information on a specific topic, you have to do some tests or measurement. You and your colleagues will have to accomplish these tasks.

**Forces:** Once the review team gets to the point where the need for validation or enhancement is identified, there is a great temptation to say „Well, we have found the critical point now, and we will think about what to do exactly about it later.“

Maybe, you are even defining the necessary actions in detail in the review meeting, but you are only writing down some notes or keywords or you do not decide on a responsible and a due date.

Even if everything is documented still somebody could miss the information because he or she is not among the review participants.

**Problem: How can you make sure that everybody can continue their work with the review outcome?**

**Solution: Give a detailed and comprehensible description of all the actions defined in the review. For each action, write down a due date and a responsible. If the responsible is not present, write down who is responsible to give the information to this person.**

**Implementation:** If you are planning to do measurements or tests in defined conditions write down not the measurements themselves but also all the boundary conditions. Give success criteria of tests or measurements in advance, too.

Even when you think that you did not get any concrete outcome from the review, you certainly have some outcome that can and should be made concrete in order to allow an elaborate progress of the project. This could be anything, even “soft” points like a common understanding of the problem or the awareness about some missing information. Write down a responsible and a due date for the communication of this outcome to the management or other departments in your company.

If not all the actions decided upon are absolutely crucial agree on a priority of the actions and also write it down.

**Examples:** Do not say, „Maybe further evaluation of the SW behaviour in extreme conditions necessary,“ but

1. „Testing of SW version 12.3.4 on processor SmCa3.5. Stimulate temperature sensor input temp\_1 with 50 and user interface input user\_3 with 933 and measure display output for 3 hours while increasing the environment

temperature from -20 to +50 °C. Output has to be between 11 and 19.8. Responsible: Peter. Due date: Oct 31<sup>st</sup> 2009.” or

2. „Peter and Heidi will meet in order to define the exact measurements. Responsible: Heidi. Due date: Sep 24<sup>th</sup> 2009.” or

3. “Write down common understanding of temperature problem with SW version 12.3.4 on processor SmCa3.5 and present it to management in next team manager meeting. Responsible: Bob. Due date: Sep 10<sup>th</sup> 2009.”

---

## Acknowledgements

Many thanks to my sheperd Klaus Marquardt for his great support, his patience with me as a patterns beginner and his concrete feedback with many examples!

Also I would like to thank the members of EuroPloP 2009 workshop C, namely Allan Kelly, Alben Antonova, Lise B. Hvatum and Wim Laurier. Their questions and feedback helped me a lot to improve my paper.

---

## References

- [1] Tom Gilb, Dorothy Graham, *Software Inspection*, Addison-Wesley, 1993
- [2] Karol Frühauf, Jochen Ludewig, Helmut Sandmayr, *Software-Prüfung – Eine Anleitung zum Test und zur Inspektion*, vdf Hochschulverlag an der ETH Zürich, 5. Auflage, 2004
- [3] Mark Prince, Andy Schneider, *Patterns for the End Game*, Europlop 2004, <http://hillside.net/europlop/europlop2004/Papers/wwc/C3.pdf>
- [4] Jeffrey K. Liker, *The Toyota Way*, McGraw-Hill, 2004

## **Design Patterns for Practical Creativity Towards Innovation**

Petros Georgiakakis & Symeon Retalis

ITisART - Interactive Technologies for ART and Education  
4, Platonos Street, 18535, Piraeus, Greece  
[petros@itisart.com.gr](mailto:petros@itisart.com.gr)

University of Piraeus, Department of Digital Systems,  
80 Karaoli & Dimitriou, 18534 Piraeus, Greece  
[retal@unipi.gr](mailto:retal@unipi.gr)

### **Fostering Creativity**

According to Wikipedia creativity is “a mental and social process involving the generation of new ideas or problem solutions, or new associations of the creative mind between existing ideas or concepts”. Creativity towards innovation in product development settings are often used interchangeably. Innovation and creative product development refer to completely new as well as significantly improved products, processes or methods.

Creativity techniques have been developed by both academia and industry to shift a person's mental state into one that fosters creativity. Michalko (2006) in his book presents the various creativity techniques that have been proposed in the Oslo Manual which is the foremost international source of guidelines for the collection and use of data on innovation activities in industry (OECD, 2005). Several other web resources contain guidelines and examples of creativity techniques (e.g. <http://www.brainstorming.co.uk/tutorials/> or [http://www.mindtools.com/pages/main/newMN\\_CT.htm](http://www.mindtools.com/pages/main/newMN_CT.htm)) in effort to train and guide individuals and teams which technique to choose and how apply them. Of course, it is well known that “design patterns are potentially more general than existing guidelines, use more specific examples, deliberately scope their context of application, and explicitly reflect particular values” (Dearden and Finlay, 2006).

Thus, the scope of this paper is to propose two design patterns for two well known creativity techniques, namely the Six Hats technique and the SCAMPER technique. The ultimate goal is to create a pattern language about the various creativity techniques that promote collaboration for innovation. This is actually the context of our current research and development work within the EU funded idSpace project (<http://idspace-project.org>). In the idSpace project we investigate the collaborative development process of innovative products, and it should be best supported by a groupware system, a system that is computer-based and supports two or more users engaged in a common task, and that provides an interface to a shared environment. In our attempt to aid the groupware systems designers, we needed to cooperate with experts who know about creativity, innovative product development processes and computer supported collaborative

corporative learning, as well as with application domain specialists environment (e.g. aerospace industry) who apply creativity techniques into their working environment (e.g. aerospace industry). As an effect, we faced the challenge of creating a common language that will give designers recommendations about the functionality of the idSpace groupware system. In general, people within a discipline often have trouble communicating their ideas and decisions to other specialists (Borchers, 2001). One of the challenges in groupware system design, which is a multidisciplinary task, is to develop effective techniques for making specialists' knowledge and assumptions more explicit, and easier for the specialists from other disciplines to understand and refer to. In the next sections we present two design patterns about well known creativity techniques. A full list of the available strategies that could be transformed into patterns can be found in the State of the Art in Tools for Creativity (idSpace D2.1, 2009).

### **What are the Flow Design Patterns about creativity techniques**

According to Goodyear (2005) and Hernández Leo et al. (2005) pedagogical strategies that foster collaboration can be described via design patterns. Actually, various pedagogical design patterns have been published. Creativity techniques that foster collaboration can be also described via design patterns and especially via the specific type of design patterns called flow design patterns (FDP). This is due to the fact that creativity techniques (e.g. Six-Hats, SCAMPER, TRIZ, Disney etc.) are considered to be task-oriented. Thus the FDPs can clearly explain the set of tasks that are needed to be performed by people who participate into the creativity process. The term "flow pattern" was originally coined by Hernández Leo et al. (2005) to portray coordination and sequencing of tasks of a (learning) process. Thus, the FDP define the sequence of the tasks that the technique dictates as well as other elements needed for the various tasks, such as the duration of a task, the use of a particular tool for a given task and so on.

The structure of a FDP includes elements such as design problem's description, the related context and a documented solution suggestion for this problem with concrete examples. Table 1 shows in detail the format of a FDP which is similar to the suggestion of CLFP (Collaborative Learning Flow Patterns) by (Hernandez-Leo et. al, 2005). We have chosen this specific format since it is detailed and situated by Hernández Leo et. al, that it is very useful and fully understandable by practitioners which are the final recipients of our FDPs. Since our design patterns are not system patterns we are using a more actor playing description, a description that the FDPs can attribute.

**Table 1. The flow design pattern format**

<b>Element</b>	<b>Explanation</b>
Name	Name of the FDP
Context	Environment type in which the FDP could be

	applied (state an example from a real-world learning activity capable of being structured according to the FDP) as well as the forces that create the problem
Example	One example that states the need of the FDP
Forces	The contradictory considerations that must be taken into account when choosing a solution to a problem.
Problem	Problem to be solved by the FDP
Solution	Description of the proposal by the FDP for solving the problem
Types of Tasks	Types of tasks, together with their sequence, performed by the actors involved in the activity.
Types and structure of Groups	Description of the types of groups of learners identified and how they are related
Consequences	A description of the results, side effects, and trade offs caused by using the pattern
Related Patterns	Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns
Examples – Known uses	Examples of real usages of the pattern
References	References that correlate with the pattern
Related Patterns Thumbnails	Thumbnail of the related patterns

### The flow design pattern of the “Six Thinking Hats” technique

**Name:** Six Thinking Hats



#### **Context**

In everyday life (academic, professional, or political), problem solving and/or decision making is often a result of collaboration within a group of people with different thinking and planning styles. Group members contribute according to their personality, inner strengths and thinking styles. Some people think strategically, methodically and with great discipline, trying to foresee possible consequences, while others often “listen to their hearts”. Some people think from a very rational, positive viewpoint, showing resistance to change, while at the same time they don't make creative leaps. Other people are used to a more intuitive approach to problem solving which makes them engage with



passion into new ideas that might not be very realistic based on socio-technical constraints of the context. It is necessary, however, to hear all voices and examine a problem and possible solutions from various perspectives in order to achieve pluralism in depicting the ideal solution.

**Example:** Let's assume that a town council is trying to decide whether or not several local school buildings should be combined into a new one, and what the options are for the use of the old buildings if those become vacant. The decision makers have to analyze all options, critically determine the advantages and disadvantages of the suggested solutions, and do a risk assessment of the outcomes before ending up with creative final solutions.

### **Forces**

Six Thinking Hats creativity method can be used when there are time constraints. The formation of groups (applying a hat) is very easy while at the same time there is no need of specific competencies of the participants. It is also proposed when the need of several different opinions can be heard instead of getting as outcome one and final decision. The complexity of this method is minor since no preparation from the participants is needed.

### **Problem**

How can groups make sure that all possible (practical as well as emotional) perspectives have been examined during a problem solving or a decision making process?

### **Solution**

**Use a technique where you can approach the solution to your problem from different perspectives.**

“Using the Six Thinking Hats technique for looking at a problem, decisions and plans will mix ambition, skill in execution, sensitivity, creativity and good contingency planning.” as de Bono stated (1992),

This technique assists groups in creating a complete and concrete view of the problem to be solved, by considering diverse thinking styles and incorporating multiple views. Groups are able to discuss the full complexity of their decisions, and identify possible drawbacks or benefits which might not, otherwise, be noticed.

### **Types of Tasks**

The collaboration process is broken down into six “Divisions”, each corresponding to a thinking style and represented by a “Thinking Hat”. Members have to perform their thinking within each division. The guidelines for each division are:

- White Hat: be neutral, objective, and unbiased
- Red Hat: be intuitive, emotional, and instinctive
- Black Hat: be pessimistic & judgmental; think of disadvantages
- Yellow Hat: be optimistic, and hopeful; think positively
- Green Hat: be creative, think out-of-the box (new perspectives)
- Blue Hat: manage, coordinate, summarize, facilitate

In the following paragraphs all divisions will be described in detail.

**White Hat:** Members who are working on the problem under the White Hat need to collect data, group those, and interpret information objectively and accurately. The objectives of the White Hat are:

- Exposition of statistical data
- Concentration on actual facts (and not opinions or beliefs)
- Acknowledgement of incomplete or inaccurate knowledge

- Suggestion of solutions that logically result from the data

Questions asked from a White Hat's perspective are:

1. What are known facts, data, and other information on hand?
2. What are the unknown facts, data, and other information on hand?
3. What additional information is needed?
4. What is there to be learned from this procedure?
5. What is the methodology for obtaining the facts and data needed to reach a solution?
6. Based strictly on the data and information collected, what are the possible, logically-derived solutions?

Red Hat: Members who are working on the problem under the Red Hat think with their "heart". They need to use their intuition and instinct to evaluate the situation, its outcomes, and the possible solutions (as those get proposed by the other divisions).

The objectives of the Red Hat are:

- Adoption of intuitive reactions
- Awareness and evaluation of others' feelings
- Promotion of emotional views
- Exposition of implied advantages of different approaches
- Exposition of implied disadvantages of different approaches
- Exposition of contradicting outcomes

Questions asked from a Red Hat's perspective are:

1. What is my initial reaction to a suggestion?
2. How do I feel about a decision I might make?
3. Do I believe I am making the right choice?
4. Does anything inside me tell me there is a better option?

Black Hat: Members who are working on the problem under the Black Hat need to concentrate on the dangers and flaws of each approach, and emphasize the worst case scenarios for any proposed solution. The objectives of the Black Hat are:

- Identification of negative outcomes and their consequences
- Identification of flawed or weakly-supported contingency plans
- Consideration of inadequate resources
- Elimination of pitfalls and non-beneficial ideas

Questions asked from a Black Hat's perspective are:

1. What is a serious flaw of this recommendation?
2. What is a major drawback to this way of thinking?
3. What are the odds of failure?
4. What could be potential worst-case scenarios?
5. Are necessary recovery resources in place?

Yellow Hat: Members who are working on the problem under the Yellow Hat need to bring forward optimistic ideas which may provide opportunities for success. The objectives of this division are:

- Identification of benefits of recommendations
- Evaluation of opportunities within proposed solutions

- Assessment of good-case scenarios
- Assessment of feasibility of recommendations
- Promotion of enthusiasm and motivation

Questions asked from a Yellow Hat's perspective are:

1. What is the best way to approach the issue?
2. What is a reasonable and realistic way to make things work?
3. What are the positive outcomes of each idea?
4. What are the long-term benefits of each action?

Green Hat: Members who are working on the problem under the Green Hat need to vision the problem in a new, open and unrestricted way, in order to generate creative and unusual ideas. The objectives of the Green Hat problem solving approach are:

- Promotion of expanded and elaborate thinking
- Application of extended rules (beyond reality restrictions)
- Envision of creative and non-habitual solutions
- Consideration of new perspectives

Questions asked from a Green Hat's perspective are:

1. What alternative solutions are possible?
2. Could a recommendation be done in another way?
3. What is an unusually unique way of looking at the issue?
4. What would constitute "outside-the-box" thinking in this case?
5. What if...?

Blue Hat: Members who are working on the problem under the Blue Hat need to maintain focus. They act as arbitrators between divisions, directors of the problem solving process, and coordinators of the group. The objectives of the Blue Hat are:

- Maximization of efficiency and effectiveness of thinking
- Facilitation and direction of the thinking process
- Determination of agenda, goals, and responsibilities
- Organization of ideas and recommendations

Questions asked from a Blue Hat's perspective are:

1. What is the best way to define the actual problem?
2. What are the goals?
3. What are the desired outcomes of the solution-seeking process?
4. What is the most effective way of moving forward?
5. What is the optimal way out of the current circumstances?

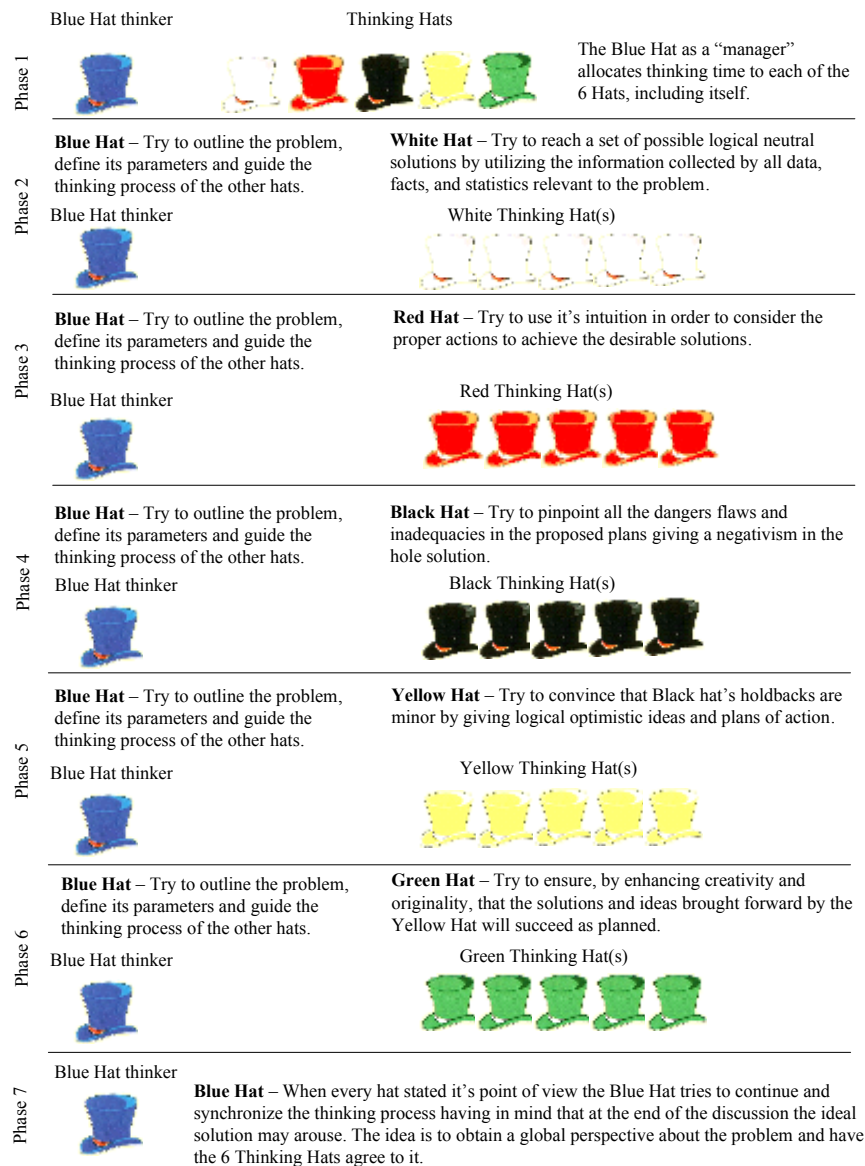
### **Types and structure of Groups**

The Six Thinking Hats technique can be used effectively in practical creativity group meetings, as it provides a way of understanding and accepting different thinking styles that people approach problem solving with.

You can use Six Thinking Hats with other persons or on your own. When it is being used among groups it has the benefit of defusing the disagreements that can happen when people with different thinking styles discuss the same problem. There is not a specific number of participators needed. Of course if there are six (1 blue hat and the other 5 hats)

or eleven (1 blue hat and the other 10 divided into couples for representing the 5 hats) etc. then the division into the Hats groups and the role playing is easier. The role (type of hat) that are given to each participant, even if it doesn't reflect their opinion, is not far beyond simple thinking, since the hat drives thinking to a specific point of view than vague thoughts around the problem.

The proposed – according to de Bono (1992) - order of thinking would transition in the following manner:



Several proposals have been made for changing the specific flow and replacing or excluding types of hats. These alterations of the De Bonos's initial proposed flow are more problem oriented. For example when the problem is Strategic Planning a proposed flow could be: Blue, Yellow, Black, White, Blue, Green, Blue ([http://en.wikipedia.org/wiki/De\\_Bono\\_Hats](http://en.wikipedia.org/wiki/De_Bono_Hats)). Of course these flows need further evaluation.

## **Consequences**

There are several theoretical reasons of why should someone use the Six Thinking Hats technique. The key point that is behind each hat is that the motive that each colored hat gives to the participant of that process is that a hat is a direction to think rather than a label for thinking. By this point the Six Thinking Hats technique encourages parallel thinking as well as full-spectrum thinking. Additionally this technique separates ego from performance ([http://www.mycoted.com/Six\\_Thinking\\_Hats](http://www.mycoted.com/Six_Thinking_Hats))

## **Related Patterns**

- FACILITATOR [1], WELL-CHOSEN RESOURCES [2]

## **Examples**

- MindTools: “Directors of a property company”  
[http://www.mindtools.com/pages/article/newTED\\_07.htm](http://www.mindtools.com/pages/article/newTED_07.htm)
- the MiddleWeb Listserv: “moving to a new city”  
<http://www.middleweb.com/MWLresources/dyckarticle2.html>
- Six Thinking Hats Testimonials:  
[http://www.debonoonline.com/Six\\_Thinking\\_Hats.asp](http://www.debonoonline.com/Six_Thinking_Hats.asp)
- Six Thinking Hats as Applied in Six Sigma by Tata Consultancy Services:  
[http://www.debonoconsulting.com/Six\\_Thinking\\_Hats\\_as\\_Applied\\_in\\_Six\\_Sigma.asp](http://www.debonoconsulting.com/Six_Thinking_Hats_as_Applied_in_Six_Sigma.asp)

## **References**

- De Bono, E. (1992), Serious Creativity: Using the Power of Lateral Thinking to Create New Ideas, Harper Collins, New York, NY.,
- IQ Matrix: 6 Thinking Hats: Solving Life’s Complex Problems  
<http://blog.iqmatrix.com/mind-map/edward-de-bono-6-thinking-hats-mind-map>
- MindTools: Six Thinking Hats, Looking at a Decision from All Points of View  
[http://www.mindtools.com/pages/article/newTED\\_07.htm](http://www.mindtools.com/pages/article/newTED_07.htm)
- the MiddleWeb Listserv: Turn a Sad Goodbye into a "Problemtnuity"  
<http://www.middleweb.com/MWLresources/dyckarticle2.html>

## **Related Patterns Thumbnails**

[1] NAME: FACILITATOR

The FACILITATOR is enriching discussions by generating cognitive conflicts. The role of the facilitator is to potentially enhancing the learning outcomes of collaboration.

[2] NAME: WELL-CHOSEN RESOURCES

During a discussion it is essential to guide stakeholders giving them accurate and exact resources in order to produce fruitful ideas.

## The flow design pattern of the “SCAMPER” technique

Name: SCAMPER



### Context

Creative problem solving processes are often characterized by the divergent nature of human thought and action. Divergent thinking involves being able to merge, or combine, unusual ideas. Researchers have acknowledged the importance of recombination of ideas, or previous effective problem solutions, as central to the process of producing a creative and innovative solution to a given problem.

### Example

We can imagine food manufacturers that often produce modifications of their flagship products in order to stimulate new consumers and increase their sales.

### Forces

The use of SCAMPER technique is being used when a pluggable solution to a specific problem is already known. Its outcomes are new solutions for different perspective. This creativity method needs only one participant which doesn't play any roles. It has no complexity, since the only module of it is the answer to one or a number of ready made questions on the already working solution in order to produce a new working outcome.

### Problem

How can we help people to generate new, more complex or complicated ideas (or problem solutions) based on one existing idea or based on several simple ideas or existing solutions?

### Solution

**SCAMPER is a problem solving technique which can be used to spark the creativity of people that try to solve specific problems and helps them reuse existing ideas or effective existing solutions to similar problems.** The main idea behind SCAMPER technique, which has been developed by Bob Eberle (1997), is that everything new is a modification of something that already exists.

Each letter in the acronym SCAMPER represents a different way someone can visualize the components of a problem and how new ideas can be triggered from existing ideas/cases:

S = Substitute

C = Combine

A = Adapt

M = Modify

P = Put to Other Uses

E = Eliminate (or Minify)

R = Rearrange (or Reverse)

### Types of tasks

The SCAMPER technique provides the stakeholders with a set of directed questions in order to direct them coming up with new ideas. These questions should not be posed in normally brainstorming sessions but should be typically used to direct in ways that deliver new ideas. These questions drive the whole discussion between the participants and lead them to minimise the number of out of point proposals as well as the time consumed to bring new ideas upfront.

#### Substitute

Try to imagine the consequences of replacing part of the problem, product or process with something else. Changes can also concern people, things, places, emotions, ideas. New ideas may come up by looking for replacements.

Typical questions:

- How can place, time, materials or people be substituted?
- What can be substituted to make an improvement?
- Swapping two things makes a difference?

#### Combine

Creativity thinking often, in order to reveal new ideas, involves the combination of unrelated ideas, goods, or services. In that case the creation of different products or processes can be achieved by the combination two or more parts of the product or the processes themselves

Typical questions:

- What combinations of components can be made (e.g. materials, features, processes, people, products etc.)?
- Can a synergy being build and where?

#### Adapt

You don't have to "re-invent the wheel" each time you want to solve a problem. The adaptation of an existing idea/solution can be the answer to your dilemma.

Typical questions:

- Can I change a part of the product?
- This change can be done in exchange for what?
- In what degree can the characteristics of a component be changed?

#### Modify

Try to imagine the consequences of increasing or reducing in scale, changing the shape, modifying the attributes of the product. Try to imagine what will bring up the changing of some parts or whole of the current situation. Rearrange the subclasses of it in an unusual way often drives into alternative products/processes.

Typical questions:

- What will occur if a feature or a component will be boosted or on the other hand downgraded?
- What will occur by the modification in the hole or sectional of a process?

#### Put to Other Uses

Imagine how the current product or solution or process can be exploited in other uses, for other purposes. Reuse the existing knowledge to expand the affordances of it. Discover other markets for your product.

Typical questions:

- Are there any other markets that the product can be used in?

- Are there any new potential customers that might be able to use it?

#### Eliminate (or Minify)

Imagine the outcome of a possible elimination – minimization of various clusters of the process or specific parts of the product. Narrow your thoughts by gradually trimming the processes, the ideas, the objects to the way that leads to most important function, to the most substantial part.

Typical questions:

- What will be the outcome of a possible removal of a specific component or part of it?
- Is there any other way(s) beyond the “beaten track” to lead to the desirable solution?

#### Rearrange (or Reverse)

Imagine the side-effects or the result that a possible reverse, a different order, in the way a product is being manufactured or a process is being executed. Picture the outcome from different angles and come up with new ideas.

Typical questions:

- What will happen if the process runs the other way round?
- What can be done to achieve the exact opposite effect?
- What are the sequences of reversing the way it is used?
- What are the sequences of reversing the order it is done?

### **Types and structure of Groups**

SCAMPER can be used either in person (which is not its strength) or in group brainstorming sessions in which there are time constraints and lack of innovative ideas. SCAMPER is actually a checklist of structured questions that promotes innovation and creativity leading stakeholders to think of changes that can be made to an existing idea (sometimes product) to create a new one. These proposed changes can be considered either as direct suggestions or as starting points for lateral and divergent thinking.

### **Consequences**

SCAMPER is ideal for use to identify possible new products. Of course during the brainstorming session many of the ideas that are generated may be unfeasible or may not suit the equipment used by the manufacturer, but some ideas could be good starting points for discussion of new products (<http://www.mycoted.com/SCAMPER>).

### **Related Patterns**

- BRAINSTORMING [3], MIND MAPPING [4]

### **Examples**

One well known example of the application of this technique is the case of MacDonald’s founder Ray Kroc. [<http://www.flatworldknowledge.com/beta-0.1/principles-management/developing-mission-vision-and-/creativity-and-passion>]

We can easily identify many of the ideas he used based on the SCAMPER technique:



- P = Put to other uses: selling restaurants and real estate instead of simply hamburgers
- R=Rearrange: having customers pay before they eat
- E=Eliminate: letting customers serve themselves, avoiding the use of waiters

Another example can be found in the book "Mind performance hacks, Tips & tools for overclocking your brain" from Ron Hale-Evans, O'Reilly, 2006, ISBN 0596101538, 9780596101534, in page 85 when the SCAMPER technique is being used in the traditional card game Rummy (<http://www.pagat.com/rummy/rummy.html>)

One more example can be found in Design and Discovery, Understanding the Design Process in which we can find a case study that students were introduced to the SCAMPER technique by examining an everyday item the water bottles: ([http://www97.intel.com/en/DesignDiscovery/Curriculum/DesignProcess/Session2/S2\\_KeyConcepts.htm](http://www97.intel.com/en/DesignDiscovery/Curriculum/DesignProcess/Session2/S2_KeyConcepts.htm))

### **References**

- <http://www.brainstorming.co.uk/tutorials/scampertutorial.html>
- [http://www.mindtools.com/pages/article/newCT\\_02.htm](http://www.mindtools.com/pages/article/newCT_02.htm)
- <http://litemind.com/scamper/>

### **Related Patterns Thumbnails**

#### [3] BRAINSTORMING

BRAINSTORMING is a method of group interaction in both educational and business settings. It is designed to generate a large number of ideas for the solution of a problem.

#### [4] MIND MAPPING

MIND MAPPING is a technique used to represent words, ideas, tasks, or other items linked to and arranged around a central key word or idea. This technique is being used to generate, visualize, structure, and classify ideas, and as an aid in study, organization, problem solving, decision making, and writing.

### **Acknowledgments**

This work has been partially supported by the ISTFP7 idSpace project: Tooling of and training for collaborative, distributed product innovation -2008-216199

### **References**

Borchers, J. (2001). A Pattern Approach to Interaction Design. John Wiley & Sons, Chichester, UK, 2001.

- Davinia Hernández Leo, Juan I. Asensio-Pérez, Yannis A. Dimitriadis (2005). Computational Representation of Collaborative Learning Flow Patterns using IMS Learning Design. *Educational Technology & Society* 8(4): 75-89 (2005)
- De Bono, E. (1992). *Serious Creativity: Using the Power of Lateral Thinking to Create New Ideas*, Harper Collins, New York, NY (1992).
- Dearden, A. and Finlay, J. (2006). Pattern Languages in HCI: A Critical Review, *Human-Computer Interaction*, 21(1), 49-52.
- Eberle, R (1997). *Scamper: Creative Games and Activities for Imagination Development*. New York: Prufrock Press.
- Garzotto, F., Retalis S. (2009). A Critical Perspective on Design Patterns for e-Learning. In L. Lockyer, S. Bennett, S. Agostinho, B. Harper (eds.) *Handbook of Research on Learning Design and Learning Objects: Issues, Applications and Technologies*. Idea Group Inc.
- Goodyear, P. (2005), "Educational design and networked learning: patterns, pattern languages and design practice", Retrieved on Jan 2009 from [www.ascilite.org.au/ajet/ajet21/goodyear.html](http://www.ascilite.org.au/ajet/ajet21/goodyear.html)
- IdSpace (2009). State of the Art in Tools for Creativity, idSpace D2.1 project deliverable, URL: <http://hdl.handle.net/1820/1658> (accessed March 2009)
- Michalko, M. (2006). *Thinkertoys: A Handbook of Creative-Thinking Techniques*. (2nd ed.). Berkeley, CA: Ten Speed Press.
- OECD (2005) *The Measurement of Scientific and Technological Activities Oslo Manual: guidelines for Collecting and Interpreting Innovation Data, 3rd Edition (Complete Edition - ISBN 9264013083) Science & Information Technology*, pp.1-166

# PLITS: A PATTERN LANGUAGE FOR INTELLIGENT TUTORING SYSTEMS

Dina Salah, The American University in Cairo, Computer Science Department, dsalah@aucegypt.edu  
Amir Zeid, The American University of Kuwait, Division of Sciences and Engineering, azeid@auk.edu.kw

## Abstract

The design and implementation of Intelligent Tutoring Systems (ITS) is a very complex task, as it involves a variety of organizational, administrative, instructional and technological components. In addition, there are no well established methodologies or development tools for ITS implementation. Therefore systematic, disciplined approaches must be devised in order to leverage the complexity of ITS implementation and achieve overall product quality within specific time and budget limits (Vladan Devedzic & Harrer, 2004).

The goal of patterns within software community is to provide software developers with solutions to recurring software problems. However, the concept of patterns has received surprisingly little attention so far from researchers in the field of ITS (Vladan Devedzic & Harrer, 2004).

In this research work, we tried to mine patterns by reverse-engineering some of the existing ITS systems. These identified patterns were semantically organized and categorized to form the basic core of a PLITS: A Pattern Language for Intelligent Tutoring Systems.

## 1. Introduction

A pattern describes a problem that occurs over and over again in our environment and then describes the core of the solution to that problem in such a way that you can use this solution a million times over without ever doing it the same way twice (Alexander, Ishikawa, & Silverstein, 1977). Moreover, a pattern language is a collection of such solutions which, at every level of scale, work together to resolve a complex problem into an orderly solution according to a pre-defined goal. Pattern Languages include rules and guidelines that suggest the order and granularity for applying each pattern in the language (Vladan Devedzic & Harrer, 2004).

ITSs incorporate built-in expert systems in order to monitor the performance of a learner and to personalize instruction on the basis of adaptation to the learner's learning style, current knowledge level, and appropriate teaching strategies (Liegle & Woo, 2000). The classical ITS architecture is composed of the following components illustrated in figure (1):

- **Expert Model:** This model contains the domain knowledge.
- **Pedagogical (Tutor) Model:** This model provides the knowledge infrastructure necessary to tailor the presentation of the teaching material according to the student model.
- **Domain Model:** This model contains the knowledge about the actual teaching material.
- **Student Model:** This model stores details about the student's current problem-solving state and long-term knowledge progress, essential for adapting the material to the student's characteristics
- **Communication (User Interface) Model:** This model is responsible of user interaction.

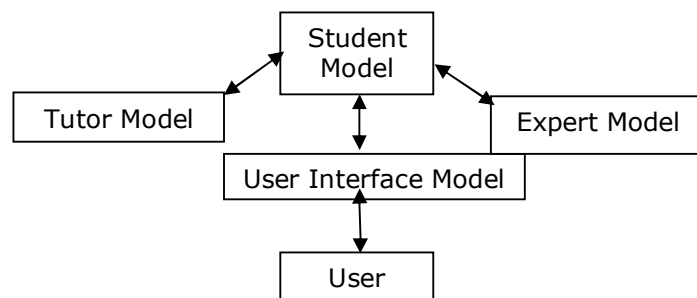


Figure (1): ITS Architecture (El-Sheikh & Sticklen, 1998)

A recent analysis of a number of existing ITS architectures has revealed that many ITS designers and developers use their own solutions when faced with design problems that are common to different systems, models, and paradigms. However, a closer look into such solutions and their comparison often shows that different solutions and the contexts in which they apply also have much in common, just like the corresponding problems do. In all such cases we can talk of the existence of patterns (Vladan Devedzic & Harrer, 2004).

## 2. The Proposed Pattern Language for Intelligent Tutoring Systems (PLITS)

This research, presents PLITS: A Pattern Language for Intelligent Tutoring Systems. PLITS is not a mere collection of patterns that can be used in the implementation of ITSs it includes rules and guidelines that explain how and when to apply its patterns to solve a problem which is larger than any individual pattern can solve.

In this research work, we first identified the functional requirements of ITS which will be summarized in table (1) and then tried to discover these features in a number of real ITSs that are broadly used and are listed in table (2). If these features were indeed found in existing ITSs, then these features were considered widely adopted and applicable and were therefore included in our pattern language for ITSs. In table (3) we map these Functional Requirements to their corresponding ITS patterns

**Table (1): ITS Functional Requirements and Related ITS Model**

<b>Functional Requirement</b>	<b>ITS Model</b>
Curriculum Composition	Domain Model
Lesson Presentation Planning, Exam Generation	Tutor Model
Curriculum Instantiation	Domain Model
Collective Student Information Instantiation	Tutor Model
Students Registration and Access Control	Student Model
Student Model Standardization	Student Model
Determining and Achieving Student Goals	Student Model
Initializing Student Model	Student Model
Update and Maintenance of Student Model	Student Model
Representing Student Status Per Topic	Student Model
Representing Student Status Per Exercise	
Student Exam Grade Computation	Tutor Model
Monitoring Student Progress and Updating Students' Progress Reports	Tutor Model
Creating and Customizing Curriculum	GUI Model
Guiding Students Through a Complex Task	GUI Model

Table (2) provides a list with the ITSs that were searched for patterns

**Table (2): ITSs that were searched for Patterns**

Name	Domain
SQLT-Web (Mitrovic, 2003)	Database Learning(SQL)
Cyberphysique (Nkambou & IsaBelle, 1998)	Physics
Web Passive Voice Tutor-Web PVT(Virvou & Tsiriga, 2001)	Language Learning
Verb Expert (Fum, Giangrandi, & Tasso, 1989)	Language Learning
CAPIT: An ITS for Capitalization and Punctuation(Mayo, Mitrovic, & McKenzie, 2000)	Language Learning
Intelligent Language Tutoring System for Grammar Practice(Heift, 2001)	Language Learning
An Interactive Course Support System for Greek (Heift, Toole, McFetridge, Popwich, & Tsiplakou, 2000)	Language Learning

## 2.1 PLITS Pattern Categories

While formulating PLITS we aimed to follow a comprehensive approach to cover all aspects related to ITS implementation in order to provide a road map for any ITS developer or designer. To provide this comprehensive approach a number of pattern categories were used in PLITS including access, instructional, design, adaptive and interaction patterns. Although some of those patterns were not intended for ITSs, yet we found that they can be useful in ITS implementation.

1. **Access Patterns:** Access Patterns are concerned with the ways that users may access the various resources (Paris Avgeriou, Retalis, & Papasalouros, 2003).
2. **Instructional Patterns:** Instructional Patterns are concerned with the various tasks that tutors perform in order to create and edit courses and learning resources (Paris Avgeriou, Retalis, & Papasalouros, 2003).

Both Instructional Patterns and Access Patterns are used in the Learning Management Systems domain. However, we found out that they can be useful in ITS implementation if they were modified to suit the ITS developers and designer needs.

3. **Design Patterns:** Design Patterns can be divided into the following subcategories:
  - **Creational Patterns:** Creational Patterns abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented (Gamma, Helm, Johnson, & Vlissides, 1995).
  - **Structural Patterns:** Structural Patterns are concerned with how classes and objects are composed to form larger structures. Structural class patterns use inheritance to compose interfaces or implementations (Gamma, Helm, Johnson, & Vlissides, 1995).
  - **Behavioral Patterns:** These patterns are concerned with algorithms and the assignment of responsibilities between objects. These patterns describe patterns of communication between objects or classes (Gamma, Helm, Johnson, & Vlissides, 1995).
4. **Adaptive Patterns:** Adaptive instruction can be defined as real-time modification of the instructional curriculum, learning environment to suit different student characteristics ("ELEN Project").

Adaptive Patterns are patterns that are used in Adaptive Learning Systems. However, we found that there is a number of adaptive learning patterns that can be of great use to the designers and developers of ITSs if they were modified to suit the ITS developers and designer needs.

More details on adaptive patterns for ITS implementation can be found in (Salah & Zeid, 2009).

5. **Interaction Patterns:** Interaction Patterns are focused on solutions to problems that end-users have when interacting with systems. The patterns take an end-user perspective which leads to a format where usability is the essential design quality (Trætterberg & Welie, 2000).

It is important to note that some of the discovered patterns were based on existing patterns; however, we modified some of these patterns in order to suit the needs of ITS implementation. All modified patterns have new names that begin with the word “New”.

## 2.2 PLITS Pattern Template

Documenting a pattern is the first step to build a pattern language. The GOF pattern template was used since it is very complete and provides straightforward guidelines for implementing the patterns into software by including more implementation details rather than generic solutions.

## 3 Building an Intelligent Tutoring System

The process of building a typical ITS is composed of four phases; building the Domain Model, Student Model, Tutor Model and the Graphical User Interface Model. However, this is the typical sequence followed, nevertheless, it is not mandatory to strictly follow this order and some ITS developers use a different order for building ITSs.

In section 3 we will highlight the patterns that can be useful in every phase, and then in section 4 we will present PLITS our proposed pattern language for ITSs.

### 3.1 Phase One: Building the Domain Model

The domain model contains the knowledge about the actual teaching material, since the teaching material is usually composed of a number of topics. Each topic is composed of a number of learn items, each topic has a number of exercises related to it and each learn item has a number of examples related to it, furthermore, since the curriculum is shared by all students but yet is tailored to every student current knowledge level thus a single curriculum instance and global student access is needed.

This led us to identify two patterns that are needed in building the domain model; the Whole Part Pattern and the Singleton Pattern.

#### Pattern 1: The Whole Part Pattern

**Classification:** Design Patterns

**Intent:** Helping with the aggregation of components that together form a semantic unit.

In ITSs teaching material is composed of a number of topics. Each topic is composed of a number of learn items, each topic has a number of exercises related to it and each learn item has a number of examples related to it.

**ITS Applicability:** The Whole Part Pattern can be applied in the domain model implementation in aspects in ITS design related to composing topics, composing lessons and composing curricula.

#### Known Uses

1. **GET-BITS (Vladan Devedzic, Jerinic, & Radovic, 2000)** : GET-BITS is an object-oriented model of ITSs .GET-BITS Uses a variant of the Whole Part Pattern in the framework's lesson presentation planner and remedial actions planner which is a part of the student's knowledge examination and assessment.
2. **SimQuest (Joolingen, King, & Jong, 1997)** : SimQuest is an intelligent biomedical simulator for medical training utilizing the expertise of professionals in the fields of 3D graphics programming, clinical training, game and simulation design, medical illustration, human factors and mechanical engineering. SimQuest focuses on the use of advanced simulation and gaming technologies for medical training. SimQuest uses the Whole Part Pattern in its lesson presentation planner to represent the teaching material and its constituent parts.

**Related Patterns:** Course Creation and Customization, Singleton, Adapter, New Student Model Initialization.

**References:** The Whole Part Pattern is one of the Pattern Oriented Software Architecture patterns (Buschmann et al., 2000).

## Pattern 2: Singleton

Since this pattern is one of the GOF patterns we will only list the pattern template items that shows its usefulness in the ITS field.

**Classification:** Design Patterns (Creational Pattern).

**Intent:** Ensuring that a class has only one instance, and provide a global point of access to it.

**ITS Applicability:** The Singleton Pattern can be applied in the Domain Model implementation in ITS design in Curriculum Instantiation since we need a single curriculum instance and global student access to it.

**Related Patterns:** Whole Part, Observer.

**References:** The Singleton Pattern is one of the Gang of Four patterns (Gamma, Helm, Johnson, & Vlissides, 1995).

## 3.2 Phase Two: Building the Student Model

The following patterns are needed in building the student model.

### Pattern 3: Registration-Authentication-Access Control

**Classification:** Access Patterns

**Intent:** Providing a standard registration mechanism for every user of the system. This can be achieved through building a web interface related to a database with user data and providing a mechanism for user authentication.

**ITS Applicability:** ITSs are large, multi-user systems. Due to security, privacy and institutional policy reasons, students' access to the resources of the ITS must be restricted to authorized students only. How can all the different students' access rights and privileges be effectively managed?

**Known uses**

**1. SQLT-Web (Mitrovic, 2003):** An ITS for teaching SQL (Structured Query Language).

**2. Cyberphysique (Nkambou & IsaBelle, 1998):** An ITS based on the World Wide Web that teaches physics.

Both (Mitrovic, 2003) and (Nkambou & IsaBelle, 1998) acquire information about a student through a login screen. Individual student models are stored permanently on the server, and retrieved for each student's session.

**Related Patterns:** User Model Definition.

**References:** The Registration-Authentication-Access Control Pattern is used in learning management systems; however, we discovered that this pattern can be useful to ITS developers and designers.

### Pattern 4: User Model Definition

The User Model Definition Pattern is one of the patterns that were used in adaptive systems ("ELEN Project"), however, we discovered that this pattern can be useful to ITS developers and designers in creating and maintaining a student model and providing mechanisms to modify application features based on that in order to offer the student the best possible learning experience.

More details on the User Model Definition Pattern and its usage for ITS implementation can be found in (Salah & Zeid, 2009).

### Pattern 5: User Goals

The User Goals Pattern is one of the patterns that were used in Adaptive systems ("ELEN Project"), however, we discovered that this pattern can be useful to ITS developers and designers in determining both short term and long term educational goals for each student which is an important component of the student model description.

More details on the User Goals Pattern and its usage for ITS implementation can be found in (Salah & Zeid, 2009).

#### **Pattern 6: New Student Model Initialization**

This New Student Model Initialization Pattern was based on the User Model Initialization Pattern that is used in Adaptive systems ("ELEN Project"), however, we discovered that this pattern can be modified to suit ITS developers and designers needs.

The original Student Model Initialization Pattern ("ELEN Project") deals with determining and providing the information needed to initialize the student model before all interactions in adaptive systems. It proposes that this can be done by one of the following methods:

- **User driven.** The user specifies explicitly what stereotype he belongs to.
- **Inferred by rules.** These rules indicate which user model elements and values can activate a stereotype.
- **Speculated by rules .**If the user does not specify his knowledge level then it is assumed to be average.

However, we modified the Student Model Initialization Pattern and introduced the New Student Model Initialization Pattern by initializing the student model with two types of knowledge:

- **Knowledge that is acquired from the students:** This knowledge should be determined with the guidance of both the User Model Definition Pattern and the User Goals Pattern.
- **Knowledge that can be acquired automatically from the system:** This knowledge can be acquired by implementing an entry exam that uses an Exam Generator Component. This component can use the pool of questions that was filled earlier by the teachers through Course Creation and Customization Pattern that will be discussed in section 3.4, and randomly choose a set of questions that represent different difficulty levels thus can accurately measure current student knowledge level per topic (stereotype).

More details on the New Student Model Initialization, its usage for ITS implementation and modifications made to the original pattern can be found in (Salah & Zeid, 2009).

#### **Pattern 7: New Student Model Maintenance**

The New Student Model Maintenance Pattern was based on the User Model Maintenance Pattern that is used in Adaptive systems ("ELEN Project"), however, we discovered that this pattern can be modified to suit ITS developers and designers needs.

The original User Model Maintenance Pattern ("ELEN Project") deals with the methods for capturing and maintaining changes that occur to student model elements as a results of interacting with the system. It proposes that this can be done by one of the following methods:

- Using a questionnaire form that indicates the amount of benefit the user gained from using the system.
- Interactive update of the user model by showing a pop-up form requesting the user to answer a question.

However, we modified the Student Model Maintenance Pattern and introduced the New Student Model Maintenance Pattern by capturing and maintaining changes that occur to student elements by using an exam generator component to conduct a per topic exam. This can occur after the student is presented by the topic learning material. The exam generator component can use the pool of questions supplied by the subject instructor to randomly select a number of questions that can test



the current student knowledge level and according to the results of this exam the user model maintenance module can update the student model to reflect his current status and the topics covered and his knowledge level per topic.

More details on the New Student Model Maintenance and its usage for ITS implementation and modifications made to the original pattern can be found in (Salah & Zeid, 2009).

## **Pattern 8: Adapter**

**Classification:** Structural Patterns

**Intent:** Convert the interface of a class into another interface that clients expect. Adapter lets classes work together that could not otherwise because of incompatible interfaces (Gamma, Helm, Johnson, & Vlissides, 1995).

### **Participants**

In the context of ITS we have the following participants:

- Client: Collaborates with objects conforming to the Target interface.
- Adaptee: Defines an existing interface that needs adapting.

In ITSs the Adaptee participant can be represented by two classes; Topic class and Exercise class.

- Adapter: Adapts the interface of Adaptee to the Target interface. It also acts as the Target.

In ITSs the Adapter participant can be represented by two classes; StudentTopic class and the StudentExercise class.

**ITS Applicability:** The Adapter Pattern can solve two problems for ITSs:

#### **1. Representing each student's specific status per topic?**

Any ITS deals with subject topics, however, there is a problem with the Topic class. This problem occurs because this class doesn't match the domain specific interface that the application requires because it doesn't reflect the student specific status per topic.

**This class doesn't represent the following information:**

- The student stereotype per topic; whether he is a Beginner, Intermediate, Advanced or Expert student.
- What learning item is currently being learned inside the topic by the student?
- How many times the student viewed a certain topic?
- What are the exams that he took on a certain topic?
- Student grades on every exam concerning this topic.

That is where the Adapter Pattern fits in ITSs Design. An Adapter class is needed that is capable of converting the interface of a class (Topic) into another interface that the client expects (StudentTopic). And to provide us with the functionality that the adapted class (Topic) doesn't provide.

#### **2. Representing each student's specific status per exercise?**

Any ITS deals with exercises, however, there is a problem with the Exercise class. This problem occurs because this class doesn't match the domain specific interface that the application requires because it doesn't reflect the student specific status per exercise.

**This class doesn't represent the following information:**

- Did the student view this exercise or not?
- The student answer on this exercise and whether he answered correctly or not, this will help to understand any misconception that the student might have.
- The student score per exercise.
- The number of student tries for solving this exercise.
- Did the student use out all his tries in this exercise or not yet?
- The examples that is related to this exercise, this means that if the student answers incorrectly he is shown some teaching material to revise the learning items that is covered by this exercise, this teaching material includes some examples so a record should be kept of the examples that is related to this exercise.

That is where the Adapter Pattern fits in ITSs. A StudentExcercise class is needed to act as an Adapter that is capable of converting the interface of the Exercise class into another interface that the client expects. And to provide us with the functionality that the adapted class (Exercise) doesn't provide.

### **Modifications Made to the Adapter Pattern Implementation**

The main difference between our implementation and the classic Adapter Pattern implementation is that the Adapter Pattern has two extra participants Target; which defines the domain specific interface that Client uses and Client; which collaborates with objects conforming to the Target interface.

**Related Patterns:** New Student Model Initialization, Whole Part, Master Slave, and New Student Model Maintenance.

**References:** This pattern is one of the GOF Patterns (Gamma, Helm, Johnson, & Vlissides, 1995).

### **3.3 Phase Three: Building the Tutor Model**

The following patterns are needed in building the tutor model.

#### **Whole Part Pattern**

The Whole Part Pattern can be used in the Tutor Model in the Lesson Presentation Planner and the Exam Generator Component.

#### **Pattern 9: Master Slave**

**Classification:** Design Patterns

**Intent:** The Master Slave Pattern supports fault tolerance and computational accuracy. A master component distributes work to identical slave components and computes a final result from the results that these slaves return.

**ITS Applicability:** In ITSs the Master Slave Pattern can be used to compute the exam results of each student according to their grade in each particular exercise within the exam.

**Related Patterns:** Adapter, Observer, New Student Model Maintenance, and New Student Model Initialization.

**References:** The Master Slave Pattern is one of the POSA patterns (Buschmann, et al., 2000)

#### **Pattern 10: Singleton Pattern**

ITS designers and developers can benefit from the Singleton Pattern in the Tutor Model in the instantiation of collective student information.

#### **Pattern 11: Observer Also Known as Dependents, Publish Subscribe**

**Classification:** Behavioral Patterns

**Intent:** The Observer Pattern defines a one to many dependencies between objects so that when one object changes state, all its dependents are notified and updated automatically (Gamma, Helm, Johnson, & Vlissides, 1995).

**ITS Applicability:** In ITSs an Observer class is needed to observe the status of all students and update the reports with the new student status as he navigates through the learning path.

**Related Patterns:** Master Slave, Singleton.

**References:** The Observer Pattern is one of the Gang of Four Patterns (Gamma, Helm, Johnson, & Vlissides, 1995).

### 3.4 Phase Four: Building the Graphical User Interface Model

The following patterns are needed in building the graphical user interface model.

#### Pattern 12: Course Creation and Customization

**Classification:** Instructional Patterns

**Intent:** How can the instructors be assisted in building on-line courses in ITS so that some of the tasks they need to perform can be automated in order to decrease the time and effort of performing those tasks?

**ITS Applicability:** ITSs must provide instructors with appropriate tools for creating and customizing a course. Course creation can be based on design templates with pre-set interfaces, content structure and features.

#### Known Uses

1. **Cyberphysique (Nkambou & IsaBelle, 1998):** An ITS based on the World Wide Web that teaches physics. This ITS provides an authoring environment dedicated to teachers and pedagogical designers.
2. **Eon Tools(Murray, 1998):** "Eon" is the name for a suite of authoring tools for building ITSs. Eon includes tools for authoring all aspects of intelligent tutors, including the learning environment, the domain knowledge, the teaching strategies, and the student model. Curriculum can be easily extended or modified to update information and theories, and new curriculum can be uploaded over the World Wide Web.

**Related Patterns:** Whole Part.

**References:** The Course Creation and Customization Pattern is one of the patterns that were used in learning management systems (Paris Avgeriou, Papasalouros, Retalis, & Skordalakis, 2003), however, we discovered that this pattern can be modified to suit ITS developers and designers needs.

The following patterns represent user interface patterns that focus on solutions to problems that end users have when interacting with systems. These patterns focus on usability as an essential design quality. Usability can be measured through one of the following usage indicators: Learn-ability, user guidance, memor-ability, speed of performance, error rate, satisfaction, task completion. Each pattern should state the impact on these usage indicators. If a pattern does not improve at least one usage indicator, it is *not* a user interface design pattern (Trætterberg & Welie, 2000).

#### Pattern 13: Wizard

**Classification:** User Interface Patterns (Interaction Patterns).

**Intent:** Students sometimes need to perform an infrequent complex task consisting of several subtasks which ranges between 3 to 10 tasks where decisions that need to be made in each subtask may not be known to the user.

**Usability Principle:** User Guidance (Visibility) (Trætterberg & Welie, 2000).

**ITS Applicability:** ITSs should be designed in order to take the user through the entire task one step at the time. The user steps through the tasks and is shown which steps exist and which have been completed (Trætterberg & Welie, 2000). When the complex task is started, the user is informed about the goal that will be achieved and the fact that several decisions are needed. The user can go to the next task by using a navigation widget such as a button. If the user cannot start the next task before completing the current one, feedback is provided indicating the user cannot proceed before completion for example by disabling a navigation widget. The users are given feedback about the purpose of each task and the users can see at all times where they are in the sequence and which steps are parts of the sequence. When the complex task is completed, feedback is provided to show the user that the tasks have been completed and optionally results have been processed.

Users that know the default options can immediately use a shortcut that allows all the steps to be done in one action. At any point in the sequence it is possible to abort the task by choosing the visible exit (Trätteberg & Welie, 2000).

**Known Uses**

1. **CAPIT (Mayo, Mitrovic, & McKenzie, 2000):** An ITS that teaches the rules of English capitalization and punctuation. As shown in figure (6) instructions relevant to the current problem are clearly displayed at the top of the screen. Immediately below the instructions, and clearly highlighted, is the current problem. There are also navigation buttons to move back or next to assist student in navigating without overwhelming him with plenty of lists or options.



**Figure (6): CAPIT Main User Interface (Mayo, Mitrovic, & McKenzie, 2000)**

2. **The German Tutor (Heift, 2001):** An ITS for teaching German. After the student finishes answering the current question he can go on to the next exercise with the "Weiter" (next) button. The German Tutor utilizes the Wizard Pattern by using navigation buttons to move to next exercise to assist students in navigating without overwhelming them with plenty of lists or options. Figure (7) illustrates the Wizard Pattern in the context of the dictation exercise in the German Tutor.



**Figure (7): Dictation Exercise in the German Tutor (Heift, 2001)**

**Related Patterns:** User Goals.

**References:** The Wizard Pattern is one of the Interaction Patterns (Trätteberg & Welie, 2000). Table (3) maps our proposed ITS functional requirements to the corresponding ITS patterns. This summarized description can be used as a roadmap for ITS developers and designers.

#### 4. PLITS –A Pattern Language for Intelligent Tutoring Systems

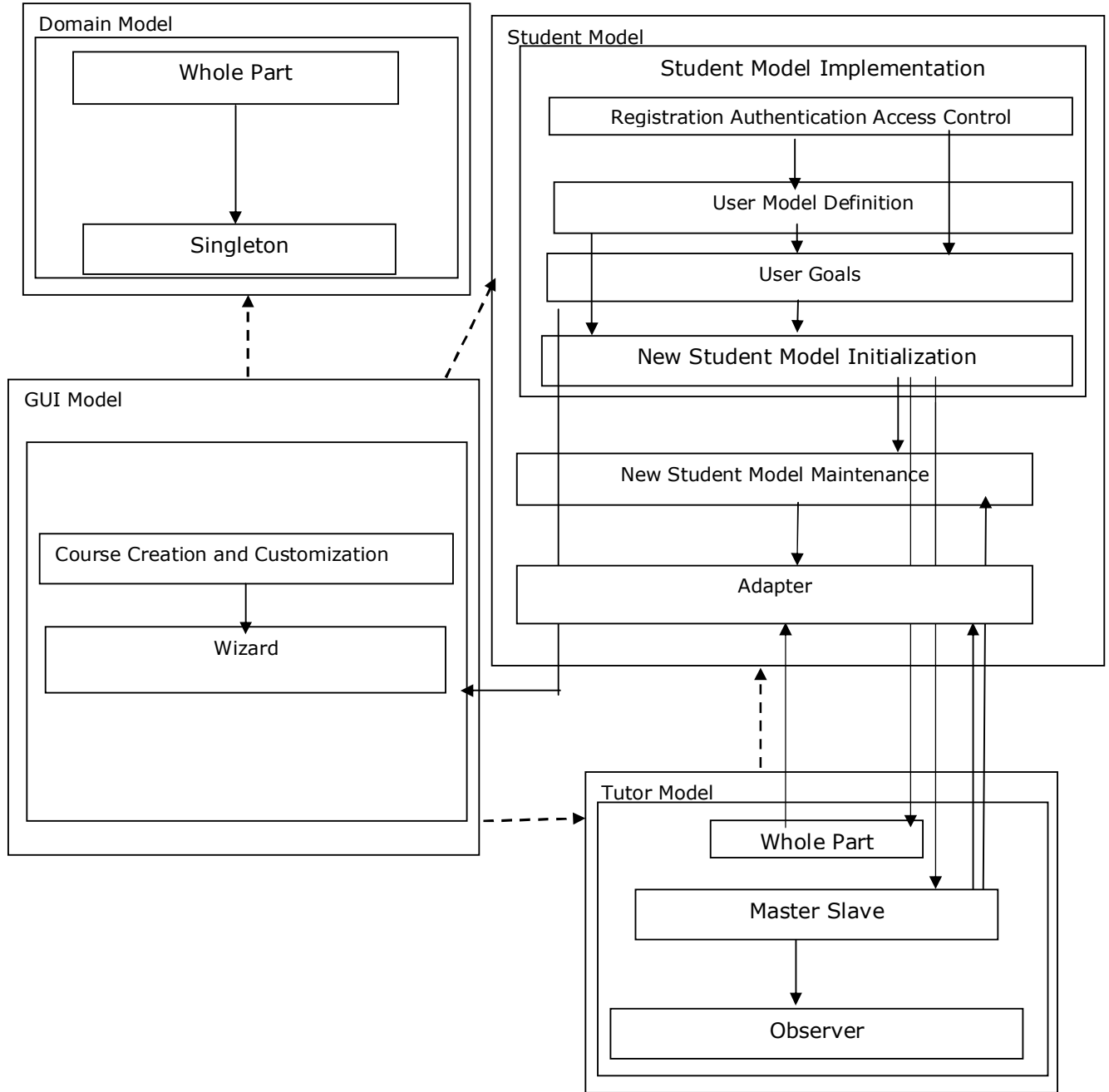
**Table (3): Mapping Between the Functional Requirements and the Corresponding ITS Patterns**

#	Category	Pattern Name	Functional Requirement	ITS Model
1	Design Pattern	Whole Part	Curriculum Composition	Domain Model
			Lesson Presentation Planning, Exam Generation	Tutor Model
2	Creational Pattern	Singleton	Curriculum Instantiation	Domain Model
			Collective Student Information Instantiation	Tutor Model
3	Access Patterns	Registration-Authentication-Access-Control	Students Registration and Access Control	Student Model
4	Adaptive Patterns	User Model Definition	Student Model Standardization	Student Model
5	Adaptive Patterns	User Goals	Determining and Achieving Student Goals	Student Model
6	Adaptive Patterns	New Student Model Initialization	Initializing Student Model	Student Model
7	Adaptive Patterns	New Student Model Maintenance	Update and Maintenance of Student Model	Student Model
8	Structural Patterns	Adapter	Representing Student Status Per Topic	Student Model
			Representing Student Status Per Exercise	
9	Design Pattern	Master Slave	Student Exam Grade Computation	Tutor Model
10	Behavioral Patterns	Observer	Monitoring Student Progress and Updating Students' Progress Reports	Tutor Model
11	Instructional Patterns	Course Creation and Customization	Creating and Customizing Curriculum	GUI Model
12	Interaction Patterns	Wizard	Guiding Students Through a Complex Task	GUI Model

Figure(8) represents the Pattern Language Approach. In figure (8) we used two types of arrows:

- > The dotted arrow is used to indicate the occurrence of a relationship between the ITS Models.
- > The solid arrow is used to indicate the occurrence of a relationship between patterns. This relationship is either a precedence or dependency.

**Figure (8): PLITS –A Pattern Language for ITS**



## 5. Conclusion

ITS complexity can be overcome by creating a pattern language for ITSs that can help software developers resolve recurring problems encountered throughout all of software development process of any ITS. In this way, designers of new or existing ITSs, especially inexperienced designers, can take advantage of previous design expertise and save precious time and resources.

In this research work, we showed that one cannot talk of patterns in the ITS domain only in the context of ITS architectures. On the contrary, there are many kinds of other patterns that can be helpful in ITS implementation. This research started out by surveying the existing ITSs and their components in search for some possible common design decisions, common interactions among components, and common generalized principles underlying superficially different designs. We extracted patterns from numerous known examples, systems, architectures, designs, etc.

We investigated the use of design patterns, access patterns, instructional patterns, adaptive patterns and interaction patterns in ITS implementation. As a result of this research we formulated PLITS a pattern language for intelligent tutoring systems implementation that includes rules and guidelines which explain how and when to apply its patterns to solve a problem.

## 6. Directions for Future Research

**Suggested directions for future research include the following:**

1. Establishing an initiative for constructing a repository of patterns for ITSs in order to attract more researchers to deposit their own patterns. That would strengthen the pattern language and offer a wealthy pool of patterns for inexperienced designers of an ITS.
2. **Introducing Web Services into ITS implementation:** Web Services are self-contained, modular applications that provide a set of functionalities (for instance; ITS expert Model) to anyone that requests them. The main characteristic of Web Services is that they interact with the applications that invoke them, using web standards such as WSDL (Web Service Definition Language), SOAP (Simple Object Access Protocol) and UDDI (Universal Description, Discovery and Integration). Basing learner modeling on web standards has the advantage of enabling the dynamic integration of applications distributed over the Internet, independently of their underlying platforms (Kabassi & Virvou, 2003).

## References

1. Alexander, C., et al. (1977). *A Pattern Language*: Oxford University Press.
2. Avgeriou, P., et al. (2003). Towards a Pattern Language for Learning Management Systems. *Educational Technology & Society*, 6(2), 11-24.
3. Avgeriou, P., et al. (2003). *Patterns For Designing Learning Management Systems*. In Proceedings of the European Pattern Languages of Programming (EuroPLOP), Irsee, Germany
4. Buschmann, F., et al. (2000). *Pattern-Oriented Software Architecture: A System of Patterns*: John Wiley & Sons.
5. Devedzic, V., & Harrer, A. (2004). Common Patterns in ITS Architectures. *Künstliche Intelligenz*, 18(3), 17-21.
6. Devedzic, V., et al. (2000). The GET-BITS Model of Intelligent Tutoring Systems. *Journal of Interactive Learning. Research*, 11(3/4), 411-434.
7. El-Sheikh, E., & Sticklen, J. (1998). *A Framework for Developing Intelligent Tutoring Systems Incorporating Reusability*. In Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Methodology and Tools in Knowledge-Based Systems. Springer-Verlag. Retrieved.
8. ELEN Project. Retrieved May, 2004, from <http://www2.tisip.no/E-LEN/outcomes.php>
9. Fum, D., et al. (1989). *Tense generation in an Intelligent Tutor for Foreign Language Teaching: Some Issues in the Design of the Verb Expert*. In Proceedings of the Fourth

- Conference on European Chapter of the Association for Computational Linguistics. Manchester, England. Association for Computational Linguistics. Retrieved.
10. Gamma, E., et al. (1995). *Design Patterns, Elements of Reusable Object-Oriented Software*. (First ed.): Addison Wesley Professional.
  11. Heift, T. (1998). *An Interactive Intelligent Tutor over the Internet*, In Proceedings of the Proceedings of ED-MEDIA 1998, World Conference on Educational Multimedia, Hypermedia & Telecommunications, Charlottesville, VA. Association for the Advancement of Computing in Education (AACE)
  12. Heift, T. (2001). Intelligent Language Tutoring System for Grammar Practice. Retrieved May, 2004, from <http://zif.spz.tu-darmstadt.de/jg-06-2/beitrag/heift2.htm>
  13. Heift, T., et al. (2000). *An Interactive Course Support System for Greek*. Bourdeau, J. & Heller, R. (eds). In Proceedings of the ED-MEDIA 00, World Conference on Educational Multimedia, Hypermedia & Telecommunications, Montreal Canada. Association for the Advancement of Computing in Education (AACE)
  14. Joolingen, W., et al. (1997). *The SimQuest Authoring System for Simulation-Based Discovery Learning*. In Proceedings of the Artificial Intelligence in Education, Tokyo. IOS Press
  15. Kabassi, K., & Virvou, M. (2003). Using Web Services for Personalized Web-based Learning *Educational Technology & Society*, 6(3), 61-71.
  16. Liegle, J., & Woo, H.-G. (2000). *Developing Adaptive Intelligent Tutoring Systems: A General Framework and Its Implementations*. In Proceedings of the ISECON Philadelphia, PA, USA
  17. Mayo, M., et al. (2000). *CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation*. In Proceedings of the International Workshop on Advanced Learning Technologies, Palmerston North, New Zealand
  18. Mitrovic, A. (2003). An Intelligent SQL Tutor on the Web. *International journal of Artificial Intelligence in Education*, 13(2-4), 173-197.
  19. Murray, T. (1998). Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design, *Journal of the Learning Sciences*.
  20. Nkambou, R., & IsaBelle, C. (1998). Cyberphysique: A Web Based Distance Learning Environment. *Global Education on the Net*, 1, 252-256.
  21. Salah, D., Zeid, A. (2009). Adaptive Patterns for Intelligent Tutoring Systems. In Proceedings of the EuroPLOP 2009. Germany.
  22. Trätteberg, H., & Welie, M. (2000). *Interaction Patterns in User Interfaces*. In Proceedings of the 7th. Pattern Languages of Programs Conference Illinois, USA
  23. Virvou, M., & Tsigira, V. (2001). *Web Passive Voice Tutor: An Intelligent Computer Assisted Language Learning System over the WWW*. In Proceedings of the Proceedings of the IEEE International Conference on Advanced Learning Technologies. IEEE Computer Society. Retrieved.

"Copyright retained by author(s). Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website."



## **Adaptive Patterns for Intelligent Tutoring Systems**

Dina Salah, The American University in Cairo, Computer Science Department, dsalah@aucegypt.edu  
Amir Zeid, The American University of Kuwait, Division of Sciences and Engineering, azeid@auk.edu.kw

### **Abstract**

The complexity of design and implementation of Intelligent Tutoring Systems (ITS) is caused by the lack of a clear road map or implementation methodology. This has led us to investigate the role of patterns in ITS implementation in order to provide software developers with solutions to recurring ITS design problems.

In this research work, we highlight the role of adaptive patterns in intelligent tutoring system implementation. We explain how those patterns were used in building The Arabic tutor, an Intelligent Language Tutoring System over the World Wide Web for teaching a subset of the Arabic Language that combines the flexibility and intelligence of Intelligent Tutoring Systems with the availability of the World Wide Web applications.

The implementation process of the Arabic Tutor was our proof of concept for the validity and usefulness of adaptive patterns in ITS implementation.

### **1. Introduction**

ITSs incorporate built-in expert systems in order to monitor the performance of a learner and to personalize instruction on the basis of adaptation to the learner's learning style, current knowledge level, and appropriate teaching strategies (Liegle & Woo, 2000). The classical ITS architecture is composed of the following components (El-Sheikh & Sticklen, 1998):

- Expert Model: This model contains the domain knowledge.
- Pedagogical (Tutor) Model: This model provides the knowledge infrastructure necessary to tailor the presentation of the teaching material according to the student model.
- Domain Model: This model contains the knowledge about the actual teaching material.
- Student Model: This model stores details about the student's current problem-solving state and long-term knowledge progress, essential for adapting the material to the student's characteristics.
- Communication (User Interface) Model: This model is responsible of user interaction.

A recent analysis of a number of existing ITS architectures has revealed that many ITS designers and developers use their own solutions when faced with design problems that are common to different systems, models, and paradigms. However, a closer look into such solutions and their comparison often shows that different solutions and the contexts in which they apply also have much in common, just like the corresponding problems do. In all such cases we can talk of the existence of patterns (Vladan Devedzic & Harrer, 2004).

Adaptive instruction can be defined as real time modification of the instructional curriculum, learning environment to suit different student characteristics (“ELEN

Project”). Adaptive patterns are patterns that are used in Adaptive Learning Systems. However, we found that there is a number of adaptive learning patterns that can be of great use to the designers and developers of ITSs if they were modified to suit the ITS developers and designers needs.

It is important to note that some of the discovered patterns were based on existing patterns; however, we modified some of these patterns in order to suit the needs of ITS implementation. Any modified pattern has a name that begins with the word “New”.

## **2. Adaptive Patterns for Building the Student Model in Intelligent Tutoring Systems**

The process of building a typical ITS is composed of four phases; building the Domain Model, Student Model, Tutor Model and the Graphical User Interface Model. However, this is the typical sequence followed, nevertheless, it is not mandatory to strictly follow this order and some ITS developers use a different order for building ITSs.

In the following section we will highlight the adaptive patterns that can be useful in building the student model.

### **2.1 Pattern 1: User Model Definition**

**Classification:** Adaptive Patterns

**Intent:** An ITS enriches its functionality by maintaining a student model and providing mechanisms to modify application features based on that. These modifications can be deciding to move to a higher difficulty level, showing the student more detailed feedback, etc., thus resulting in a personalized instruction. Standardization of the student model is an important issue, because through it we can greatly enhance the student model’s portability. This will allow learners to use several different ITS(s) and to carry their personal model with them, providing the systems with the same image of themselves, without that leading to compatibility problems. We need a student model that is small, compact and flexible. What information should an ITS keep for the student in order to offer him the best possible learning experience?

**ITS Applicability:** In an ITS setting the items that should be included in the user model definition have to be directly related to the user as a learner – anything that would be considered useful to better adapt to the learner’s particular characteristics. A complete user model definition should be comprised of the following elements:

1. **Demographic data**, which are relevant to the particular ITS (e.g. as age, gender, etc.)
2. **Student goals**, which are related to the long term and short term learning goals.
3. **Student preferences**, which includes the mode of delivery, accessibility requirements, or assessment.
4. **Student knowledge**, which includes the knowledge level about concepts to be learned and weaknesses and strengths on particular areas, sections or points of the concepts.
5. **Usage data**, which includes information like which pages were viewed, in what order, etc.

6. The **Stereotype** that applies to the student, which essentially is the group of learners he belongs to based on some predefined presuppositions in terms of knowledge level, learning and cognitive styles.

**Known Uses**

1. **An Interactive Course Support for Greek** (Tudor Heift, Toole, McFetridge, Popwich, & Tsiplakou, 2000): An ITS that teaches the Greek language.
2. **The Web Based German Tutor** (Trude Heift, 1998): A web based ITS for teaching German.
3. **Web Passive Voice Tutor** (Web PVT) (Virvou & Tsiriga, 2001): An ITS for teaching English passive voice.

Both of (Tudor Heift, Toole, McFetridge, Popwich, & Tsiplakou, 2000) and (Trude Heift, 1998) has three student stereotypes beginner, intermediate and advanced. Feedback messages are customized to suit current student knowledge level where beginners receive more detailed feedback than advanced students. Whereas (Virvou & Tsiriga, 2001) has four student stereotypes novice, beginner, intermediate and expert. Nevertheless, all the above mentioned ITSs take into consideration the student stereotype, knowledge, short term goals and usage data, but none of them take into consideration student demographic data, long term goals or preferences.

**Related Patterns:** User Goals, New Student Model Initialization

**References:** The User Model Definition Pattern is one of the patterns that were used in adaptive systems ("ELEN Project"); however, we discovered that this pattern can be useful to ITS developers and designers.

## 2.2 Pattern 2: User Goals

**Classification:** Adaptive Patterns

**Intent:** The student model description is a part of the student model component of an ITS and it should include student goals. What information should be considered as student goals in a student model in an ITS?

**ITS Applicability:** Any ITS needs to include specific student goals in the student model in order to facilitate adaptation, and capture the real intent of the learner with respect to the learning material.

**Student goals can be divided into two categories:**

**Long-term goals:** Educational goals that are valid for a longer period of time and require significant effort to be met, these goals are usually determined by the learners.

**Short-term goals:** Educational goals that are valid for a shorter period of time and require relatively moderate effort to be met. These goals are usually determined by the Tutor Module component of an ITS.

**Known uses**

1. **An Interactive Course Support for Greek** (Tudor Heift, Toole, McFetridge, Popwich, & Tsiplakou, 2000): An ITS that teaches the Greek language to foreigners
2. **The Web Based German Tutor** (Trude Heift, 1998): A web based ITS for teaching German.
3. **Web Passive Voice Tutor** (Web PVT) (Virvou & Tsiriga, 2001): An ITS for teaching English passive voice.

All the ITSs mentioned above accomplish short term goals by tailoring teaching material content and sequence to suit current student knowledge level.

**Related patterns:** User Model Definition, New Student Model Initialization.

**References:** The User Goals Pattern is one of the patterns that were used in Adaptive systems ("ELEN Project"); however, we discovered that this pattern can be useful to ITS developers and designers.

### 2.3 Pattern 3: New Student Model Initialization

**Classification:** Adaptive Patterns

**Intent:** ITSs initialize the student model before all interactions. What is the minimum amount of information needed to start the system? What kind of information and what amount is the student willing to provide?

**ITS Applicability:** ITS developers and designers need to initialize the student model with two types of knowledge; knowledge that is acquired from the students and knowledge that can be acquired automatically from the system.

The User Model Initialization Pattern initializes the student model with the student stereotype through one of the following methods:

- **User driven.** The user specifies explicitly what stereotype he belongs to.
- **Inferred by rules.** These rules indicate which user model elements and values can activate a stereotype.
- **Speculated by rules.** If the user does not specify his knowledge level then it is assumed to be average.

#### Known Uses

1. **Web PVT** (Virvou & Tsiriga, 2001): The student is initially assigned to one of the four distinct stereotypes, namely novice, beginner, intermediate and expert, according to her/his performance on a preliminary test.

2. **Verb Expert** (Fum, Giangrandi, & Tasso, 1989): At the beginning of each session, the tutor starts the interaction with the student by presenting him by an exercise on a given topic. The Student Modeller compares the answer of the student with that of the expert Module in order to identify the errors and to formulate some hypotheses about their causes in order to initialize the student model.

**Related Patterns:** User Model Definition, New Student Model Maintenance, User Goals.

**References:** This pattern was based on the User Model Initialization Pattern that is used in Adaptive systems ("ELEN Project"), however, we discovered that this pattern can be modified to suit ITS developers and designers needs.

We modified the solution suggested by the User Model Initialization Pattern in the New Student Model Initialization Pattern by accurately initializing the student model using an exam generator component. More details can be found in section 3.3.

### 2.4 Pattern 4: New Student Model Maintenance

**Classification:** Adaptive Patterns

**Intent:** During the course of interaction with ITSs many things about the student can be changed, e.g. current student knowledge level. Thus, the student model must be adapted to the new realities. How should the system capture those changes so as to maintain a good student model ("ELEN Project")?

**ITS Applicability:** ITS designers should define the conditions that govern the maintenance of the student model. The designer should define the scope of the

maintenance changes by defining the reason for updates. The reason is then quantified in terms of choice of elements to undergo change.

The Student Model Maintenance Pattern suggests maintaining an accurate student model through one of the following methods:

- Using a questionnaire form that indicates the amount of benefit the user gained from using the system.
- Interactive update of the user model by showing a pop-up form requesting the user to answer a question.

We modified the solution suggested by the Student Model Maintenance Pattern by using an exam generator component to conduct a per topic exam. This can occur after the student is presented by the topic learning material. The exam generator component can use the pool of questions supplied earlier by the subject instructor to randomly select a number of questions that can test the current student knowledge level and according to the results of this exam the user model maintenance module can update the student model to reflect his current status and the topics covered and his knowledge level per topic.

#### **Known uses**

1. **Web PVT** (Virvou & Tsiriga, 2001): A web based ITS for teaching the English passive voice. It records information about which concepts the student has mastered and to what extent. In addition, it records the kinds of error the student has made during past interactions as well as the most suitable explanation of each category of error. The information from the long term student model forms an individual model of the student, which together with the active stereotype are used in order to provide adaptive navigation support and perform intelligent analysis of the student's solutions to exercises.

2. **An Interactive Course Support for Greek** (Tudor Heift, Toole, McFetridge, Popwich, & Tsiplakou, 2000): A web based ITS for teaching Greek. The student model is a representation of the current skill level of the student. For each student the student model keeps score across a number of error types, or nodes, for example, grammar or vocabulary. The score for each node increases and decreases depending on the grammar's analysis of the student's performance. The amount by which the score of each node is adjusted is specified in a master file and may be weighted to reflect different pedagogical purposes.

**Related Patterns:** New Student Model Initialization.

**References:** This pattern was based on the User Model Maintenance Pattern that is used in Adaptive systems ("ELEN Project"), however, we discovered that this pattern can be modified to suit ITS developers and designers needs.

### **3. Using Adaptive Patterns in Building the Arabic Tutor**

#### **3.1 Pattern 1: User Model Definition**

A student model is the image that the system has about the learner. The closer it is to the learner's real characteristics and needs, the better the personalization. The User Model Definition Pattern was used in the Arabic Tutor; the user model included short term and long term student goals, student knowledge level on particular topics or learn items, usage data, and student stereotype.

### 3.2 Pattern 2: User Goals

User Goals Pattern was used in handling both long term and short term goals in the Arabic Tutor.

#### **Long Term Goal Handling in the Arabic Tutor:**

Students who use the Arabic Tutor are required to determine their goal from using the system. This is achieved by allowing the students to choose the topics that are of interest to them. These topics may be dependent on each other. Each topic may have successors, predecessors, or topics that can be taught in parallel with this topic.

#### **Short Term Goal Handling in the Arabic Tutor:**

Short term goals are goals that are valid for a shorter period of time, for example, if the long term goal of a certain student was “to learn topic X”, then the short term goal is the presentation of the appropriate learn items and examples that he needs to study or view in order to gain the required knowledge to master the chosen topic. Short term goals in the Arabic Tutor are usually determined by the Tutor Model component.

The short term goal handling in the Arabic Tutor begins after the student signs up and determines his long term goals from using the system. The results of the entry exam per topic will be used to control the teaching material that will be viewed by the student. This is illustrated in table (1):

**Table (1): Teaching Material According to Student Level.**

<b>Student Stereotype</b>	<b>Topic Explanatory Text</b>	<b>Learning Item Explanatory Text</b>	<b>Number of Learn Item Examples</b>
Beginner	Yes	Yes	6
Intermediate	No	Yes	4
Advanced	No	Yes	2
Expert	No	No	1

In this phase, the student needs to perform a complex task consisting of several subtasks where decisions that need to be made in each subtask may not be known to the student. This complex task is represented by the student long term goal which is mastering a certain topic. This long term goal can be divided into a number of short term goals. Thus the complex task can be divided into a number of subtasks represented in the learn items that are included inside this topic and that needs to be studied in a particular order to preserve the predecessor and successor relationship between learn items.

### 3.3 Pattern 3: New Student Model Initialization

The New Student Model Initialization Pattern was used in the Arabic Tutor in initializing the student model with two types of knowledge:

- **Knowledge that is acquired from the students:** This knowledge was determined with the guidance of both the User Model Definition Pattern and the User Goals Pattern.
- **Knowledge that can be acquired automatically from the system:** This knowledge was acquired by implementing an entry exam that uses an Exam Generator Component that randomly choose a set of questions that represent different difficulty levels thus can accurately measure current student knowledge level per topic (stereotype) instead of speculation or inferring as suggested in the User Model Initialization Pattern.

### 3.4 Pattern 4: New Student Model Maintenance

The New Student Model Maintenance Pattern was used in the Arabic Tutor by implementing an exam generator component to conduct a per topic exam. This occurred after the student is presented by the topic learning material. The exam generator component used the pool of questions supplied by the subject instructor earlier to randomly select a number of questions that can test the current student knowledge level and according to the results of this exam the user model maintenance module updated the student model to reflect his current status and the topics covered and his knowledge level per topic. This exam consists of 12 questions that vary in the difficulty level according to the current mastery level of student per topic. In addition, according to the level of the student that was determined in the entry exam he will be presented by a tailored feedback message in case he answered the question incorrectly.

Table (2) highlights the per exam configurable parameters in the Arabic Tutor.

**Table (2): The Per Topic Exam Configurable Parameters in the Arabic Tutor**

Student Level	Number of Exercise per difficulty level			Topic Explanatory Text	Learn Item Explanatory Text	Number of Remedial examples
	Difficulty Level 1	Difficulty Level 2	Difficulty Level 3			
<b>Beginner</b>	12	0	0	Yes	Yes	3
<b>Intermediate</b>	0	12	0	No	Yes	2
<b>Advanced</b>	0	6	6	No	Yes	1
<b>Expert</b>	0	0	12	No	No	0

#### 4. Conclusion

In this research work, we showed the role of adaptive patterns in implementing the user model in ITS, we used the example of the Arabic Tutor, a web based intelligent language tutoring system for teaching a subset of the Arabic Language as our proof of concept for the validity and usefulness of adaptive patterns in ITS implementation.

We aim to provide designers of new or existing ITSs with a road map for user model implementation within the ITS domain.

However, one cannot talk of patterns in the ITS domain only in the context of adaptive patterns. On the contrary, there are many other kinds of patterns that can be helpful in ITS implementation; design patterns, access patterns, instructional patterns, and interaction patterns. Moreover, ITS complexity can be overcome by creating a pattern language for ITSs that can help software developers resolve recurring problems encountered throughout all of software development process of any ITS. In this way, designers of new or existing ITSs, especially inexperienced designers, can take advantage of previous design expertise and save precious time and resources.

In (Salah & Zeid,2009) we proposed PLITS, a Pattern Language for Intelligent Tutoring Systems that utilizes design, access, instructional, adaptive and interaction patterns in ITS implementation. PLITS includes rules and guidelines which explain how and when to apply its patterns to solve a problem.

#### References

1. Devedzic, V., & Harrer, A. (2004). Common Patterns in ITS Architectures. *Künstliche Intelligenz*, 18(3), 17-21.
2. El-Sheikh, E., & Sticklen, J. (1998). A Framework for Developing Intelligent Tutoring Systems Incorporating Reusability. In *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Methodology and Tools in Knowledge-Based Systems*. Springer-Verlag. Retrieved.
3. ELEN Project. Retrieved May, 2004, from <http://www2.tisip.no/ELEN/outcomes.php>
4. Fum, D., et al. (1989). Tense generation in an Intelligent Tutor for Foreign Language Teaching: Some Issues in the Design of the Verb Expert. In *Proceedings of the Fourth Conference on European Chapter of the Association for Computational Linguistics*. Manchester, England. Association for Computational Linguistics. Retrieved.
5. Heift, T. (1998). An Interactive Intelligent Tutor over the Internet, In *Proceedings of the Proceedings of ED-MEDIA 1998, World Conference on Educational Multimedia, Hypermedia & Telecommunications*, Charlottesville, VA. Association for the Advancement of Computing in Education (AACE)
6. Heift, T., et al. (2000). An Interactive Course Support System for Greek. Bourdeau, J. & Heller, R. (eds). In *Proceedings of the ED-MEDIA 00, World Conference on Educational Multimedia, Hypermedia & Telecommunications*, Montreal Canada. Association for the Advancement of Computing in Education (AACE)



7. Liegle, J., & Woo, H.-G. (2000). Developing Adaptive Intelligent Tutoring Systems:A General Framework and Its Implementations. In Proceedings of the ISECON Philadelphia, PA, USA
8. Salah,D.,Zeid,A (2009). A Pattern Language for Intelligent Tutoring Systems. In Proceedings of the EuroPLOP 2009.Germany.
9. Virvou, M., & Tsiriga, V. (2001). Web Passive Voice Tutor: An Intelligent Computer Assisted Language Learning System over the WWW. In Proceedings of the Proceedings of the IEEE International Conference on Advanced Learning Technologies. IEEE Computer Society. Retrieved.

"Copyright retain by author(s). Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website."

# A Pattern Language for Online Trainings

**Christian Kohls**

**Knowledge Media Research Center, Tuebingen, Germany**

**c.kohls@iwm-kmrc.de**

## **Abstract**

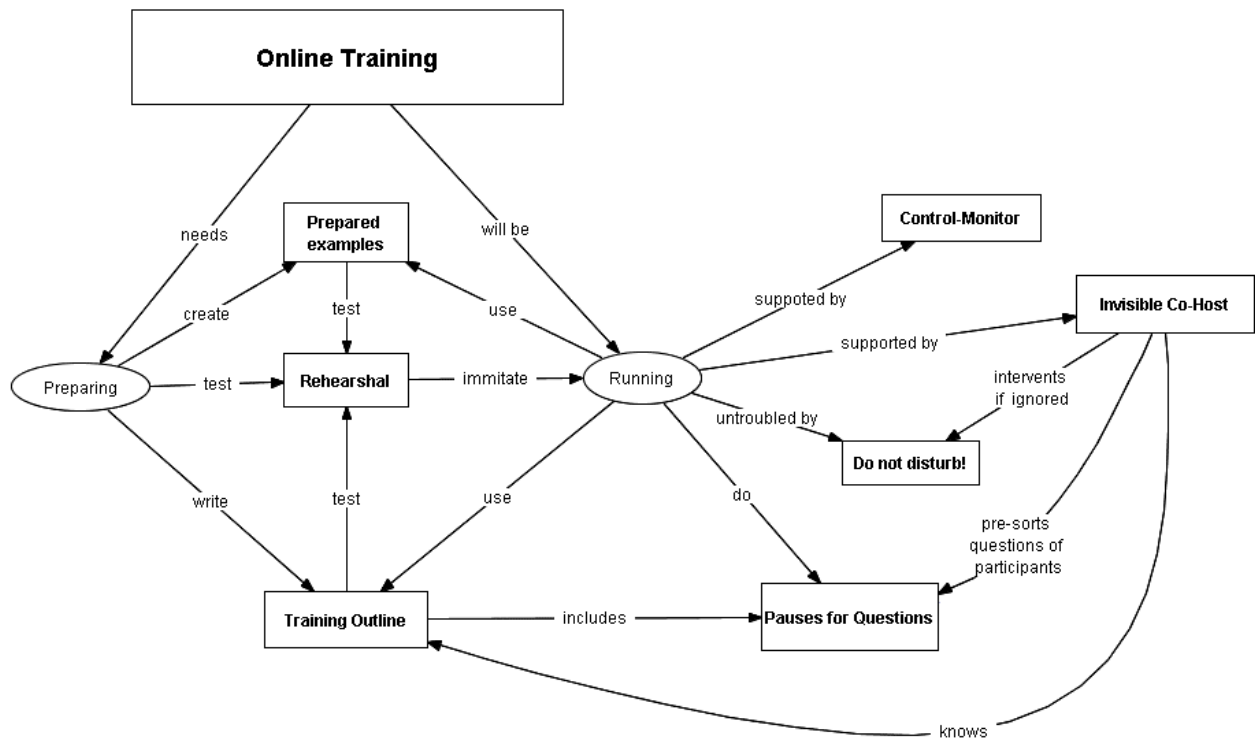
This is a small pattern language for running online trainings. The language starts with an entry pattern ONLINE TRAINING. All other patterns are supporting this pattern and help to make online trainings more alive. Some of the patterns are not exclusive for online uses, for instance PREPARED EXAMPLE or TEST RUN. These patterns could be part of a pattern language for trainings as well. They are included in this collection because they are important for online settings particularly. Another example is PAUSE FOR QUESTION. This should be applied in classroom education as well. But the present pattern description takes special forces into account, i.e. online you have to plan *more* pauses due to the lack of any non-verbal feedback.

## Mining Ground

The patterns ground in experiences of online training events of the German information portal e-teaching.org. Since spring 2006 there have been 14 online trainings, hosted by the author and guest trainers. There are usually between 30-50 participants attending the live event. Each event is recorded and can be accessed later. The trainings are open educational resources and can be accessed at<sup>1</sup>:

<http://www.e-teaching.org/community/communityevents/schulung/>

Besides our own experiences, we discussed best practices of using the conference system Adobe Connect at a user meeting. Many of the patterns we applied have been in use by others independently, i.e. Invisible CO-HOST, CONTROL MONITOR and of course TEST RUNS. Though we have gathered our experiences using a particular conferencing system, they are not limited to that system. We have tested other systems as well and their functions are quite comparable. The present patterns should be useful for any of the standard web conference systems.



The pattern map shows the relation between the patterns and how they support each other. The map clarifies that some patterns are applied in advance to prepare for the training while other patterns are more important when the actual live training takes place.

### Acknowledgement

The author wants to thank his shepherd Nuno Flores for all his help and very thoughtful suggestions. I enjoyed working with Nuno and I think I have not only learned new ideas for this paper but improved my pattern writing style in general. I appreciate all the time Nuno has invested in the shepherding process! I would also like to thank all people who have provided feedback to this paper and the patterns. Many helpful suggestions from the Writer's Workshop at EuroPLoP 2009 have helped to shape this paper as well. Thanks to all participants.

<sup>1</sup> The trainings are in German.

# Online Training

Alias: Webcast, Tool Demonstration

## **Context**

Using software applications is a fundamental activity in everyday work life. This applies to teachers, students and employees in the same way. To acquire the desired level of skills, trainings have shown to be an effective way to impart usage scenarios, process steps, tips and tricks. Trainings are suitable to introduce new software products or versions promptly.

## **Problem**

The number of participants is usually limited for classroom trainings. There is a fair amount of costs for travelling to the training site, a loss of working time, and rooms need to be allocated. The calendars of the trainer and the several participants have to be synchronized which is not always an easy thing to do. If potential participants are located at different sites, finding a time suitable for everyone becomes even harder.

## **Forces**

**Time** For many software applications, there is a need for sophisticated trainings. However, the time resources of trainers are limited. This is true particularly for part time trainers who have other job activities on their agenda as well.

**Availability** Training services should be available on-demand and just-in-time, for example if new tasks should be coped with, or a new colleague needs to be introduced to the software.

**Overview** Sometimes people just want to get an idea of a software product without committing to a time intensive training session.

**Costs** The investment of time and money is a barrier in particular for trainings that are important but not mandatory (or at least judged to be not mandatory).

**Special interests** For special interest trainings (e.g. a special field of application for standard software) there are usually not enough attendees at a local site while there might be enough interested people on national or international level.

**Feedback** Training videos offer no way to ask questions, or to signal difficulties in understanding the demonstrated functions.

## **Solution**

Online trainings allow the participation from anywhere independent of the location. If the online training is recorded, the recorded material can be accessed at anytime.

## **Details**

Today's web conference systems allow broadcasting computer screens and actions to many participants as a web cast. A trainer uses a headset and comments his action on the screen. Participants can use a chat window to give feedback and ask questions. To make life easier for the trainer, and to engage in communication with participants, a CO-HOST is a reliable

support. In this case, the training is not led by a single trainer but supported by an assistant who takes care of questions and discussion in the chat.

To check how demonstrated work steps are broadcasted to the participants, a second monitor can work as a CONTROL MONITOR. This monitor shows the broadcasted screen from the perspective of the participants.

Integrate more PAUSES FOR QUESTIONS (“Did you understand everything?”, “Any questions?”) as you would do in your classroom education and use PREPARED EXAMPLES. Plan your trainings to take no longer than 45 minutes (60 minutes being the limit!). People get tired in longer lasting on-line sessions. Shorter sessions, say 15 or 30 minutes, will also do well. While you are hosting an online training you should provide a personal and friendly atmosphere. Don’t forget to smile occasionally even if you cannot see your participants and their reactions.

### **Obstacles**

**Preparation Gap** There is a huge difference between planning and running trainings. Only if you run and implement the training, the real challenges in operating the software and applying meaningful actions will reveal. Whether the instructional outline works can only be tested if you run the training. Therefore, do a REHEARSAL before starting the training with real participants.

**Technical Problems** According to Murphy’s law, if anything can go wrong, it will! If anything goes wrong during the live event this could cause inconvenient delays and may let the participants quit even before the event started. Furthermore, technical errors look very unprofessional – even if the trainer or host cannot be accounted for the error. Therefore, always check the technology in advance, fix any broken components, and analyze sources of failure.

### **Benefits**

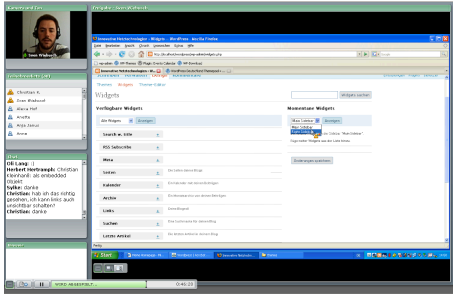
- Participants are not required to travel to a trainings site. This saves time and money. Training sessions can be integrated into the calendar of a regular working day.
- If a participant doesn’t like the training – or realizes that the content does not fit to his learning goals – one can always quit the training without causing any trouble.
- Chat windows enable participants to ask questions during the presentation (an advantage over pure training videos).
- Records allow to watch the training later on and to replay specific sections for a better understanding (an advantage over classroom training).
- No physical rooms need to be allocated or rented.

### **Liabilities**

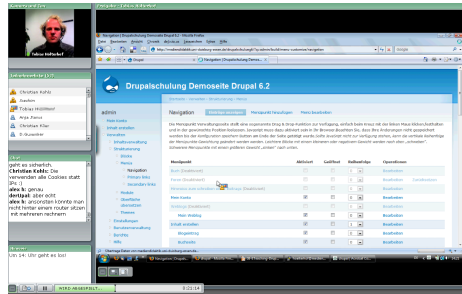
- Cognitive load for the trainer due to the lack of relaxing pauses.
- No hands-on-practice during the training.
- A lack of non-verbal feedback to see whether participants grasp everything.
- More efforts to organize the technical infrastructure for training.
- A fast internet connection is required. Still, some interruptions of audio or video may occur.
- Delayed broadcast of screen operations may cause an asynchrony of audio comments and screen video.

**Examples:**

The following examples show several online trainings hosted by [www.e-teaching.org](http://www.e-teaching.org), each introducing a different software application. Each example was hosted using the same conference system, Adobe Connect. However, there are other tools (see below) which enable to run the same type of event. The trained software applications vary in their screen dynamics. The content management systems Drupal and Wordpress are examples for web based applications where the complete screen content only changes when a new page is loaded.

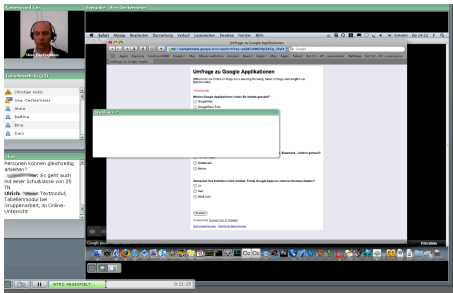


**Wordpress training**  
<http://connect.iwm-kmrc.de/p35287287/>

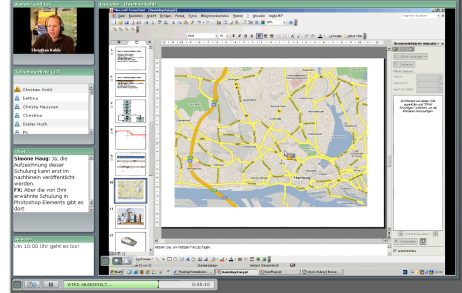


**Drupal training**  
<http://connect.iwm-kmrc.de/p68096301/>

Google Apps is an example where direct manipulation causes more frequent screen changes. In this training the participants were asked to fill out a questionnaire and their answers were evaluated live with Google Spreadsheet. The PowerPoint training is an example for frequent screen changes that occur when a slide is edited or an animation starts. While the editing was broadcasted well, the animation was not smooth because some frames were left out.

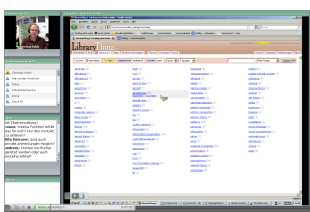


**Introduction to Google Apps**  
<http://connect.iwm-kmrc.de/p27372695/>

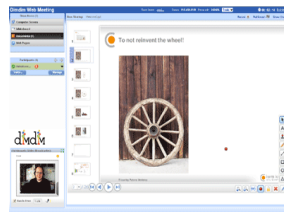


**Instructional Animations in PowerPoint**  
<http://connect.stine.uni-hamburg.de/p65354949/>

**Tools**



*Adobe Connect*



*DimDim*



*OnSync*

Adobe Connect, OnSync, and DimDim are examples for web based meeting systems that only require a flash plug-in. Hence, most web users can attend trainings without installing anything on their computer. Adobe Connect is very simple to handle and supports screen recording on Mac computers. DimDim is based on open source components and offers free hosting for up to 20 participants. OnSync has very good compressions but is less intuitive than Adobe Connect.

# Control Monitor

Alias: Second Screen

## Context

In an online training the trainer will demonstrate the use of software on a screen that is broadcasted to participants who are located at another site. In opposition to classroom settings he will not see exactly the same image on the screen. Participants will see the screen in their browser window with a delay of time, lower screen resolution and a lower refresh rate. Furthermore, the broadcasted screen is usually integrated into a conference system which brings additional panels on the screen (e.g. video-image of the trainer, chat window).



## Problem

The comments of the trainer should refer to what the participants see on their screens – and not what he sees on his screen. If the audio comments do not fit to the screen video shown, there is a danger of inducing false mental models for the participants.

## Forces

**Asynchronous streams** When speaking his comments, the trainer refers to the process steps he currently applies. However, participants receive the broadcasted screen with a time delay (whereas audio might be delivered without delay).

**True-to-detail** High frequent changes of what is shown on the screen (e.g. animation, videos or rapid input of text and drawings) will not be broadcasted true-to-detail due to the compression and low bandwidths. For examples, some frames may be left out or the viewer only shows the input of a whole word rather than the singular letters in a text field.

**Resolution** Since the captured screen is scaled down and broadcasted with lower image quality, details may get lost. In an extreme example, the trainer might see things clearly whereas participants see things only blurred.

**Lag** Frequent switches between windows frames or scrolling the screen will delay the broadcast since every pixel on the screen changes. However, the trainer will not recognise this because his own screen just acts normally.

## Solution

Put a second computer and monitor besides the computer that is used to capture the training. On this second computer one can login as a normal participant. Hence, the trainer can see the screen from the perspective of the participants.

## **Details**

The second monitor does not only show the broadcasted screen but all other panels of the conferencing system as well (e.g. his own video image recorded by the webcam). On his first monitor, the environment of the conferencing system is hidden because the trainer operates the software that is trained.

If the trainer sees a delay on the second monitor, he can mute for a few seconds to reduce the generated amount of data. That will release server capacities and bandwidth.

## **Obstacles**

***Missing Eye Contact*** While looking at the second monitor, the trainer no longer looks into the webcam. He no longer has “eye contact” with his audience because he is looking at some corner of the screen. Therefore, the control monitor should be located close to the primary monitor to avoid moving the whole body of the trainer out of the recorded video image. Not looking into the camera can affect people to think that the trainer is inattentively or unconfidently.

***Interference*** However, if the control monitor is put too close to the primary monitor, the perception of two monitors with almost (but not quite) the same content can irritate and distract the trainer.

***Echo Effects*** The speakers of the second computer have to be turned off. Otherwise, the microphone of trainer might record its own audio output recursively and annoying echo effects occur.

## **Benefits**

- By looking at the second monitor, one changes his audio comments. Rather than saying “Now you see,” one says “In a moment you will see”. Hence, the audio comments are more appropriate to what actually happens on the screen.
- A second monitor enables the trainer to observe chat messages and read them in planned PAUSES FOR QUESTIONS.

## **Liabilities**

- Looking too many times to the control monitor interferes with the work flow, for the operations will still be applied on the primary computer and not on the second one.
- Additional hardware is required.
- Time consuming setup of the trainer’s desktop.
- The trainer frequently looks to the side instead of into the camera which can be irritating.



## Examples



If the hosting computer uses an unusual screen format or a very high resolution, a second monitor can help to check what the participants see.

Small laptops or Tablet-PCs are perfect as a second screen because they are easy to setup on your desk and don't take too much space. Also, such small computers usually have a "low" resolution (e.g. 1024x768 pixels) and you can see what happens under "worst" conditions.

Using a second monitor is particularly helpful if there are frequent changes on the screen. In trainings for PowerPoint, Flash and Photoshop, the trainer was frequently checking whether the manipulations were fully captured by the conference system.

In another example, a trainer was not using a control monitor. When he was scrolling quickly up and down on a website (to just show what is at the bottom of the page), he did not realize that no participant ever saw the bottom of the website because it did not last long enough at that position to be captured by the conference system.

# Pauses for Questions

Alias: Any questions?

## **Context**

Planning and running an online training to let participants acquire software skills.

## **Problem**

In online settings there is no non-verbal feedback that could indicate whether the demonstrated process steps are understood, whether the pace is appropriate, or whether interest arouses. Without this feedback the trainer cannot adapt the training to the situation and it gets harder to optimize learning effects.

## **Forces**

**Visual Feedback** Stereotypical signals such as frowning, staring bored, widened eyes, confirming smiles etc. are giving conscious or unconscious feedback to the trainer, indicating whether his style of training is adequate or needs to be adapted. However, in an online training the trainer does not see the participants.

**Signaling** It should be allowed to ask questions at any time. However, how can a participant indicate need for more information if he cannot simply raise a hand?

**Pace** Continuous talking of the trainer is stressing and participants need some relaxing time every now and then to process the new information. However, idling moments have different effects in online trainings and seem to be awkward.

## **Solution**

Plan to invite participants to ask questions (“You understood everything?“, “Any questions so far?“) explicitly and more frequently than in classroom trainings. Announce at the beginning of the training that you will include several pauses in which questions can be answered.

## **Details**

Since there is no implicit feedback from non-verbal communication, we have to ask explicitly for feedback more frequently. At the end of the training, too, the trainer should ask for feedback: “Were the amount of information and duration appropriate?“, “Did you miss any information?“, “What would you like to hear the next time?“, “Was the quality of audio and video sufficient?“ Free answers can be typed into the chat window. Some conference systems allow preparing questionnaires in advance. To not forget pauses for questions, it is a good idea to include them explicitly into the informal TRAINING CONCEPT.

To give some relaxing time to the trainer and to involve the participants, one can ask them to contribute: “Do you have an idea for his?“, “Do you know any other example?“, “Does anybody know what I should enter into this field?“ Answers can be typed into a chat window.

Most conference systems offer chat windows and trainers are advised to make use of it.

## Obstacles

**Lose out questions.** While the trainer is demonstrating the software tool, participants can ask questions. Since the trainer concentrates on demonstrating the tool, it is easy to miss some of the asked questions. For this reason, the trainer should announce that he will not answer each question immediately but has planned several halting points to discuss questions.

**Switching focus** To read the questions, the trainer has to switch from the demonstrated software tool to the conferencing system. To avoid this switch, a second CONTROL MONITOR can help. An INVISIBLE CO-HOST could organize and filter the questions.

## Benefits

- Involvement of participants.
- Questions and obscurities can be addressed.
- Causes social awareness.
- Pauses are relaxing the trainer.

## Liabilities

- Questions cannot be answered immediately when they occur.
- Not immediately scanning the questions in the chat window makes it hard to understand afterwards which question refers to which section of the training.
- Pauses can be awkward if no participant asks any questions or gets involved.

## Examples

*Google Apps training:*

User: "Are all users required to have an account at Google?"

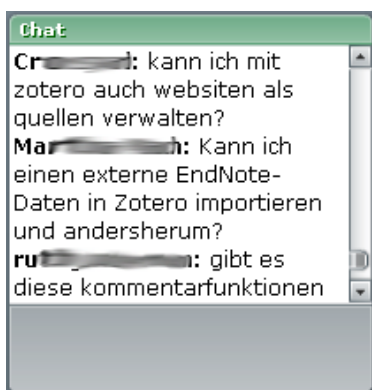
User: "How many people can work simultaneously?"

*Flash training:*

User: „What kind of braces are required for the command?“

User: "Why is the marker not directly set in the object's layer? Does that not work?"

*LibraryThing training:*



User: „Can I save website sources in Zotero?“

User: "Can I import external EndNote-files in Zotero and the other way around?"

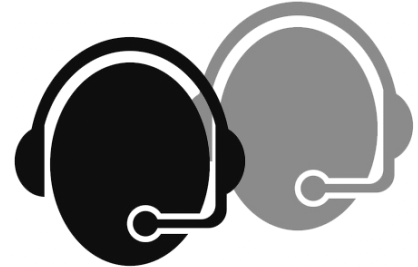
User: "Is this commenting feature available for other document types as well?"

# Invisible Co-Host

Alias: Assistant

## Context

You are running an ONLINE TRAINING with several participants and you can use a chat window for synchronous communication. A chat window allows participants to ask questions and to give feedback whether they understand everything.



## Problem

During the training, a trainer concentrates on his screen actions and his audio comments. His attention cannot be simultaneously and permanently on chat news. Also, chat news can be distracting.

## Forces

**Awareness** Since trainers can not permanently pay attention to the chat window some questions might get lost.

**Distraction** Participants can chat and communicate with each other. This extends the knowledge exchange and sharing of experiences, however, the trainer is distracted by frequent messages that scroll over the chat window.

**Filtering** Trainers should address the most important questions at an appropriate time but filtering the most important questions from the chat protocol causes inconvenient delays and binds cognitive resources of the trainer.

**Comprehension** When checking questions at a later time, for example in a PAUSE FOR QUESTIONS, it is hard to relate general questions to specific training sections, i.e. which demonstrated steps or functions are addressed by the questions.

## Solution

Run a training session in a team. Co-located with the trainer is a co-host who acts as an invisible assistant taking care of questions and messages in the chat window. The webcam only captures the trainer, while the co-host works in the back to answer questions, to sort messages and forward the important ones to the trainer.

## Details

The training is hosted by a trainer who is an experienced user of a software tool. The assistant does not need to have the same level of expertise as the trainer has. He can answer simple questions directly in the chat window and provide information about organisational issues (e.g. he can answer questions about the duration of the training, whether the training is recorded, how to control the audio volume). His most important task is to filter questions and forward the important ones to the trainer.

Since the co-host always monitors the chat window, he can relate questions in the chat to specific sections of the training (i.e. “this question is about function XYZ”).

The co-host sits next to the trainer without being captured by the camera. This way he can act in the background but can directly give signals to the trainer (e.g. wave a note) or write down questions in large letters.

### **Obstacles**

**Intrusion** The signals of the co-host can distract and irritate the trainer. For mute communication, hand signals can be agreed upon. Thumb up means: “I have seen there is a question and I will address it in a minute.” Waving the hand means: “OK, there is a questions, but at the moment I cannot address it because I want to finish the section first.”

**Briefing** The assistant should know the TRAINING OUTLINE. If somebody asks a particular question he can hint that the topic will be demonstrated later.

### **Benefits**

- Relief for the trainer allowing him to concentrate on the training.
- By permanently observing the chat window, questions can be related to specific sections of the training.
- Additional information can be provided in the chat window.
- Concurrent chat makes the training more alive.

### **Liabilities**

- More personal resources are needed.
- Questions and gestures of the co-host can irritate the trainer.
- The co-host needs to have some minimal knowledge about the topic or has to be introduced to it by the trainer.

### **Examples**



Co-Host showing a question to the trainer

User: “How long will the training last?”

Co-Host: “Approx. 45 minutes”

User: “Is the training recorded? Where can I get the link?”

Co-Host: “Yes, the training is recorded. We will publish the link in our events section at e-teaching.org”

User: “Every now and then the sound is gone”

Co-Host: “You can change the bandwidth to Modem connection. Less quality but continuously audio stream.”

User: “What is the difference between comments and pings?”

Co-Host (using his knowledge): “Comments: Visitors can write text comments. Pings: Other blogs can automatically notify the entry that they are referring it.”

User: “Can I add images and videos?”

Co-Host (using the TRAINING OUTLINE): “That’s coming in a minute...”

# Do not disturb!

Alias: On-Air

## Context

To run an online training there is no need for a special training room and trainers can host the session from their office.

## Problem

Extraneous disturbances such as ringing phones, chatting colleagues, students or any operating noises distract the trainer, interfere the quality of audio broadcast and irritate the training flow.

## Forces

**Usual Fuss** Colleagues, guests, or students cannot know that an online training is broadcasted live from your office and that entering the office or knocking at the door will disturb. In particular if you usually have a policy of open doors and people are used to just entering other people's offices, you have to communicate that there is a "special situation".

**Interruption** To turn off annoying operating noises or to get rid of someone during the training costs time, destabilizes the Zen of a training session and can cause embarrassing situations in the virtual meeting room.

**Absence** To leave the desktop temporarily is not an option because a virtual training room without a trainer is very irritating.

## Solution

Place a sign at your office door to signal colleagues, guests, visitors or students that you do not want to be disturbed at the moment. Explain the reason.

## Details

Block the access to your office by locking the door or place something (e.g. a chair) in front of the door. Tell all your close colleagues that you are running an online training. Turn off phones, mobiles, noisy hard drives or air conditioners.

## Obstacles

**Explanation** To not upset your colleagues, do communicate that your request for silence is not arbitrary but for a good reason. Explain in friendly words why people around you have to be quiet for the next 45 minutes.

**Defence** If an unexpected source of distractions occurs, a CO-HOST can help to turn off a noisy machine, ask people in neighbouring offices to be quiet and get rid of unwelcome visitors. By any means, the trainer should not find himself in a situation where he has to leave his desktop during the training.



## Benefits

- Simple and effective, easy to implement
- Everybody knows what is going on
- Running a training without disturbance

## Liabilities

- People cannot contact you in an emergency
- Explicit “Do not disturb!” hints can provoke and even invite trouble makers
- People might think you are taking things too seriously

## Examples



Radio and TV stations use an „On Air“ signal.



A simple “Do not disturb” print-out decorates the office door when an e-teaching.org event is hosted.

The sign also informs visitors what is going on (e-teaching.org Live-Webcast).

# Rehearsal

Alias: Test Run, Dry run

## Context

You are planning an online training with several participants and you are currently preparing the structure. The training content and TRAINING OUTLINE are fixed in principle.

## Problem

There is a huge difference between planning and running trainings. Only if you run and implement the training, real challenges in operating the software and applying meaningful actions will reveal. Whether the instructional outline works can only be tested if you run the training.

## Forces

**Complexity** You have to try out your PREPARED EXAMPLES to realize that they do not work as planned or are too time consuming and complex.

**Elocution** The spoken audio comments for each of the process steps have to be clear and to the point. Practice can help. If you are looking for the right words and examples during the training you are risking shakiness in your style of presentation.

**Pausing** Pauses in online training are much more intensive and seem to be longer; hence, trainers have less time to gather themselves or think about the next steps. Moments of silence (e.g. walking in the training room or looking into the audience) are harder to achieve in virtual environments and often appear to be strange.

**Coherence** The sequential order of a training requires that information is build on each other and that the structure is coherent. If you only plan theoretically it is easy to lose sight of the big picture and which details have to be presented at which time. As an expert you have everything in your head but that does not mean that you can already communicate it in an instructional way.

## Solution

Do at least one test run in advance of the actual training. Use the test run to analyze at which points the training concept can be improved.

## Details

It is crucial to speak all comments out aloud in spite of being on your own. By doing so, you get a feeling for the language, how to describe and comment actions. Also, you get a better feeling of how much time you actually need for the training and can adapt the materials accordingly.

To be a skilled user of software is not enough to be a good trainer; you also need to have a good presentation style and skills of communication.





## **Obstacles**

**Knowing the traps** Known problems of the software to be trained should not surprise the instructor. Rather, he should know the problems and offer solutions. Tackling such obstacles cannot be planned without practically try out the features. The more often you try the same flow of operations, the more likely you will find any potential source of errors.

**Passionless** It's the same as with everything: the more practice the better you get. But be aware of running too many test runs. You can easily get bored by the content or start oversimplifying concepts. This could effect your motivation when you are running the real training. You might present the content mechanically. Therefore, the final test run should not be immediately before the real training (e.g. just an hour in advance). It's better to test run one day in advance.

## **Benefits**

- The trainer gets more confident in demonstrating and explaining.
- Potential sources of errors are recognized and can be addressed.
- The TRAINING OUTLINE is optimized step-by-step.
- If you have a new idea during the test run, you can just write it down and still change the outline.
- You can interrupt a test run at any time.

## **Liabilities**

- Investment of time
- Having too many pre-phrased sentences makes the training monotonously
- Be aware of false safety and be prepared that still a lot of things can go wrong
- Excitement and stage fright will still be with you on the real training

## **Examples**

- Each online training at e-teaching.org is at least tested one time without audience
- Trainings in classroom settings are very often tested with no or just a small group of participants
- Rehearsals are quite common for theatre performances, concerts or public talks.

# Training Outline

Alias: storyline, training script

## **Context**

When planning an online training, surely everybody has a mental picture or a concept in his head of what skills should be acquired. Then comes the moment of the real training and you are excited and more stressed.

## **Problem**

While taking the stress – even stage fright – during the live training into account, one easily forgets important information or jumps over some important sections. Thus, some of the knowledge gaps of the participants may remain.

## **Forces**

**Skipping** It is tempting to leave out some content sections just to get sooner to the end of the training session.

**Sequencing** Functions and models should build on each other. The sequence in which software features are presented is crucial for its understanding.

**Flippancy** Flippancy lets one forget to mention some information, leaving behind knowledge gaps of the participants. Delivering information afterwards is laborious and not all participants might receive it.

**Biasing** Writing a full trainings storyboard costs a lot of time and limits your flexibility as it builds mental barriers to adapt to individual educational situations.

## **Solution**

Make a list of the most important features and process steps you want to communicate. Write them down as an ordered item list setting the structure of the training.

## **Details**

The items will remind you which information to provide in which order. They are the base structure.

Each item is only an anchor for your presentation – and not a pre-phrased manuscript for your training. You should avoid reading information or descriptions from a manuscript.

The item list could include special sections such as defining the learning goals, PAUSES FOR QUESTIONS, or at which time you will show a PREPARED EXAMPLE.

If you run the training with a CO-HOST, you should give him a copy of the training script. If he knows what will be tackled in the training he can better provide information in the chat.

## Obstacles

**Tracking** For your own orientation, highlight the most important information and the start of new sections using a bold font. For clarity, check all items you have talked about with a pen.

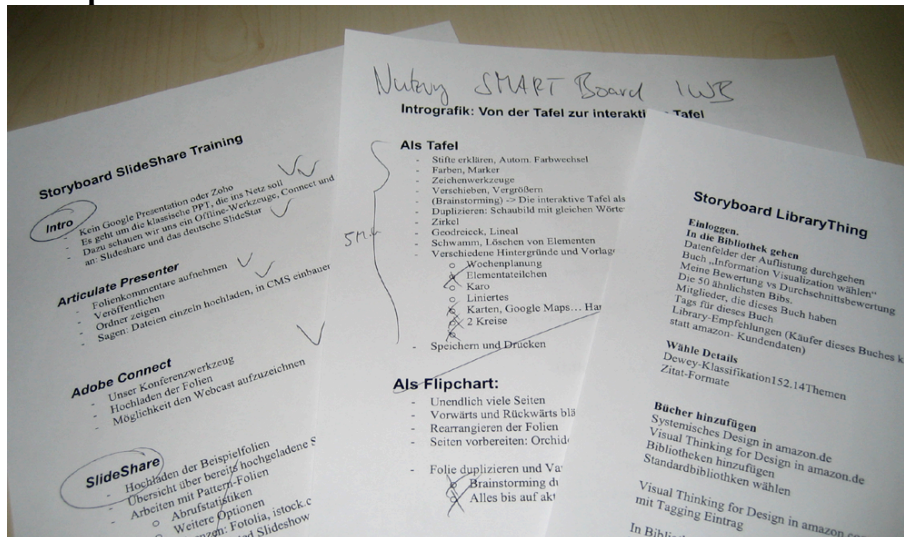
## Benefits

- Trainings get better structured
- There is less danger to forget relevant information
- PAUSES FOR QUESTIONS can be integrated explicitly
- You always keep on track
- You resist the temptation to jump over some sections

## Liabilities

- Blindly following the script makes the training inflexible
- Having too many details and an inner force to necessarily tackle everything can draw out the training

## Examples



The outlines of online trainings for SlideShare, SMART Board interactive whiteboards, and LibraryThing.

# Prepared Example

Alias: Tested Example, Working Example

## Context

You plan and run a classroom or online training. In order to show typical scenarios or actions, you will enter example data for the sake of demonstrating functions or process steps.

## Problem

The search for meaningful examples costs time which is not available during the time of training. If you use meaningless (random) data or materials, you can show the functions of a program but you communicate the purpose ineffectively. Careless chosen example data can cause undesired effects or atypical results.



## Forces

**Limitation** You cannot use a huge number of examples in a single training, hence you have to use good examples (in terms of instructional benefit).

**Meaning** Examples have an important instructional impact and are not just a means to click through program functions. Meaningful examples not only demonstrate the functions but also show their typical uses and benefits.

**Time** An example can be meaningful but time-consuming, long-winded, and take up too much time of a training session.

## Solution

Prepare your examples with care. Use meaningful input data and actions that apply to the typical needs of the trained audience. If needed prepare materials (images, spreadsheet values) in advance, in order to avoid editing times that show no key functions of the software.

## Details

The most simple example that does not skip any relevant information might be the best. Try several examples and decide which one fits best.

Never enter meaningless data such as “abc“, “xyz“ or “slkjssw swkjd“ into an input field! Use examples that are appropriate to the semantic of the input field. This is important to associate the right type of input and to demonstrate the purpose and meaning of the input data.

If you show graphic user interfaces that allow direct manipulation, you should not start wildly but create new objects purposefully. Do not just sketch on a graphic panel or create meaningless objects. This will only raise questions such as “Well, fine! But why do I need this?” If the function is meaningful then you will always find a meaningful example!

## Obstacles

**Stickiness** Due to all the efforts for finding good examples one might be tempted to use them at any price. But one should always adapt to the individual training situation. If one example takes longer than planned (e.g. participants asked some questions) just skip another example. If you use a chat window you can ask participants whether there is need for more examples.

**Renewal** Do not use examples only because you have prepared them! It's hard to discard an example that has cost you some time to create. But if you find a better one just go for it!

## Benefits

- Meaningful examples increase the learning effect
- The trainer is more relaxed during the training if he has not to search for examples
- It helps to avoid unexpected results when using the software
- Good examples can be re-used
- A toolkit of good examples lets you act flexible and offers the opportunity to adapt the training according to the interests of the participants

## Liabilities

- It takes a lot of time for preparation
- You may have to adapt examples for each specific group to provide meaningful example data

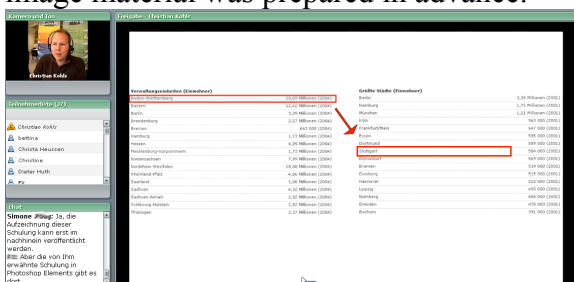
## Examples

**LibraryThing training** (<http://connect.iwm-kmrc.de/p79077399/>)

LibraryThing is an online service to manage your library. To demonstrate the service, an account was created and all books of the trainer were added online (in advance) to show a realistic setting. To demonstrate how a book was added, the trainer chose books from which he knew that meta-data were available in other online libraries. In the actual training he only added books to the list which he had successfully added before to avoid any bad surprises. LibraryThing also shows correlations to other users with similar interests (having similar books or tag clouds). To make sure to use interesting examples, the “live” exploration of correlated tags and users was tested several times in advance. The queries that created the best results were used in the actual training.

**Animation in PowerPoint training** (<http://connect.stine.uni-hamburg.de/p65354949/>)

PowerPoint offers many ways to animate objects. Usually such animations are distracting but cleverly used they can offer instructional support. This was demonstrated in an online training. Since the training was meant to show how to edit more complex PowerPoint animations, each image and each slide layout was prepared in advance. The purpose of the training was not to show how to create or find interesting images. Therefore, the example image material was prepared in advance.



The screenshot shows a PowerPoint slide with a table. The table has two columns: 'Herausgeber (Herausgeber)' and 'Gedruckte Werke (Druckwerke)'. The first row is highlighted in red. A red arrow points to the 'Herausgeber' column header. The second row is also highlighted in red. The table contains the following data:

Herausgeber (Herausgeber)	Gedruckte Werke (Druckwerke)
Verlag	1,234 567 890 (1234)
Hamburg	2,345 678 901 (2345)
Altona	3,456 789 012 (3456)
Harburg	4,567 890 123 (4567)
Verlag	5,678 901 234 (5678)
Hamburg	6,789 012 345 (6789)
Altona	7,890 123 456 (7890)
Harburg	8,901 234 567 (8901)
Verlag	9,012 345 678 (9012)
Hamburg	0,123 456 789 (0123)
Altona	1,234 567 890 (1234)
Harburg	2,345 678 901 (2345)
Verlag	3,456 789 012 (3456)
Hamburg	4,567 890 123 (4567)
Altona	5,678 901 234 (5678)
Harburg	6,789 012 345 (6789)
Verlag	7,890 123 456 (7890)
Hamburg	8,901 234 567 (8901)
Altona	9,012 345 678 (9012)
Harburg	0,123 456 789 (0123)

Animated highlighting of table rows can be used to direct the attention of the audience. In the training it was explained how to create such animated highlighting. The table itself was prepared in advance because the topic was “how to create animations” and not “how to create tables”.

# Hearing the Student's Voice - Patterns for Handling Students' Feedback

Axel W. Schmolitzky

University of Hamburg, Germany  
Vogt-Kölln-Str. 30  
D-22527 Hamburg  
+49.40.42883 2302

[schmolitzky@acm.org](mailto:schmolitzky@acm.org)

Till Schümmer

FernUniversität Hagen, Germany  
Universitätsstr. 1  
D-58084 Hagen  
+49.2331 987 4371

[till.schuemmer@fernuni-hagen.de](mailto:till.schuemmer@fernuni-hagen.de)

**Abstract:** Feedback is an important value in agile methodologies. It is also essential for any context where people are learning. Typically the focus is on *giving* learners feedback on their learning progress, either from an educator's point of view or between the learners via peer feedback. This paper focuses on *gaining* feedback from learners. We discuss methods and good practices for encouraging learners to switch their perspective from a recipient of facts to a critical observer of material produced by peer students and educators. This is essential both for educators (as a way to learn more about their own teaching) and for students (to be able to express opinions and give constructive feedback).

## 1. Introduction

The following patterns describe good practices for gaining and handling feedback *given by students*. The patterns focus on educators and facilitators at higher level education institutions who aim at raising the level of critical thinking of their students.

Critical thinking is relevant especially in areas that do not offer simple “right” or “wrong” answers, but where solutions to problems have to be chosen from solution spaces with several dimensions of freedom. The resulting solutions thus have to be produced through intensive interaction between several stakeholders; these should be able to (a) identify their interests and (b) express them in a way others can build upon.

The intensive interaction between the various stakeholders such as teachers, experts, and students heavily relies on constructive feedback. Since the students have the freedom to construct their solution from a large design space, they require feedback in order to understand how their solution is perceived by others. As long as this feedback is expressed as constructive feedback, the student will be forced to reflect on her individual solution and potentially revise the design decisions based on the advice given by others.

The ability to give constructive feedback and the strategies to digest feedback given by others are thus important soft skills that become especially important in teamwork that is omnipresent in modern work places. Since more and more

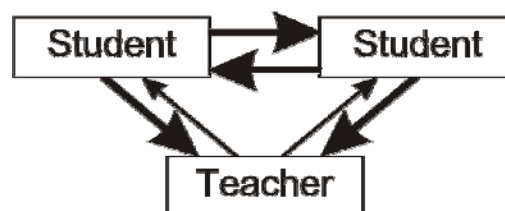
Copyright retain by author(s). Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.

work places involve distributed interaction (and since distributed interaction becomes more important in educational settings as well), we will also discuss how feedback can be gained and given in distributed computer-mediated settings. While the mechanics for distributed computer-mediated interaction have been described before [Schümmer & Lukosch, 2007], we focus on the way how standard collaboration support such as Wikis, forums, or e-mail can be used to support feedback.

The general questions that have to be answered by the following patterns are:

- **Who** is giving feedback? Typically it will be students, but other interest groups are possible.
- **On what** is feedback given? On lectures, presentations, design proposals, written text by teachers and students.
- **How** is the feedback expressed? A feedback culture needs to be established, an etiquette especially for constructive feedback.
- **To whom** is the feedback given? How large is the audience? In which setting is the feedback given (e.g., face-to-face discussions vs. anonymous interaction in a web-based forum)? How comfortable students feel to speak up?
- **What for** is the feedback given? It should be taken seriously and digested thoroughly. In the ideal case, it should improve the competencies of the feedback taker.

We can also distinguish between informal and formalized feedback. Informal feedback is highly individualized and takes place during the course: students in the same course share their views on the learning experience or teachers give the students feedback on their progress. Formal feedback is typically part of a university's QA program. In German universities, these programs model a formal process that was accredited when the master or bachelor program of the university was established. Since the formal processes are typically not under control of one individual teacher, we do not consider them in our pattern collection.



**Figure 1: Directions of feedback (fat arrows are focused in this paper)**

Considering the interaction in a course, we distinguish three different feedback channels: From teachers to students, from students to teachers, and between peer students. The feedback from teacher to student has been covered in other pattern languages before. We suggest looking at the feedback patterns provided by Eckstein et al. (2002) if your desired direction for feedback is from teachers to students.

The evaluation patterns of Derntl (2004) describe how the students' progress can be measured and how students can get feedback on their performance.

In this paper, we focus on the other two paths for feedback in which the student is giving the feedback. Especially the feedback among students does to our experience contribute to the critical thinking which we would like to increase for the students.

A general question for feedback patterns is whether feedback can be given anonymously or not. Anonymous feedback is especially important when critical feedback could influence the grades of a student. In such cases, students could fear that a critique of the teaching would result in bad marks for them. Anonymous feedback – on the other hand – often tends to be too harsh, especially when the feedback giver is not used to receive or give feedback.

In this paper we are not interested in grading, neither teachers nor students. We think that the application of agile values should result in a more balanced relation between teacher and students.

## 2. The Pattern Collection

This paper contains the following patterns:

**YOU ARE HEARD:** Ensure that students see that their feedback is considered as important and that the feedback can have an effect.

**FEETIQUETTE:** Give students guidelines on how they should express their feedback.

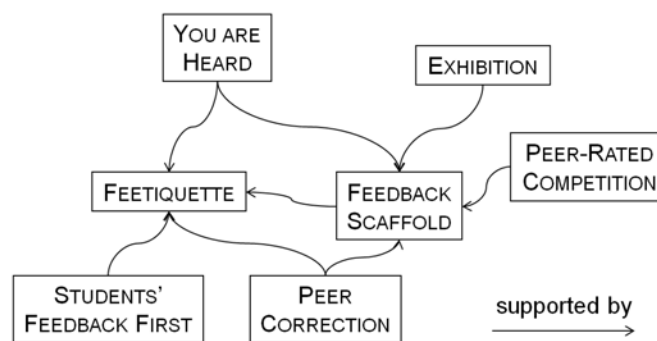
**FEEDBACK SCAFFOLD:** Help students to structure their feedback by giving them an outline of the feedback they typically give.

**PEER-RATED COMPETITION:** Let students rate other students' work and give a special reward to those students who have received the highest ratings from their peers.

**STUDENTS' FEEDBACK FIRST:** Give students room to express their feedback before you as a teacher give feedback from your perspective.

**PEER CORRECTION:** Let peer students correct assignments and make clear that the ability to rate a solution is another important way of understanding the assignment's subject.

**EXHIBITION:** Let students present their work to the whole class or even to future and past students and honor outstanding exhibits.



**Figure 2: The patterns of this paper and their dependencies**



## 2.1 YOU ARE HEARD

**Context:** You are teaching a course with a large audience (more than 100 participants), either in classroom or in distance education. Because feedback from the audience during lecture time is difficult due to time constraints (or for technical reasons in distance education, or because students are too shy to speak up in large audiences in the lecture room), you have decided to give the students of your course a *feedback channel* by means of an electronic forum, a newsgroup, or at least a contact e-mail.

**Problem:** It is hard to find the balance between 'no feedback' over this channel and 'too much feedback'. Sometimes students don't use the channel at all. Sometimes they are eager to express all their opinions and report on their experiences during the course, so that the teacher can not find the time to comment on all messages, especially in large courses.

**Forces:**

- Students are reluctant to provide written negative feedback as it persists (sometimes longer than the urge to write it down).
- Students assume that the teacher will be equally interested in giving them personal feedback as they are in giving it.
- RECIPROCITY (Schümmer & Lukosch, 2007) argues that any computer-mediated interaction should create a balance between efforts and benefits for all stakeholders. However, the distribution of efforts is not reciprocal in large courses since the students only have to contribute their personal view and the teacher would have to answer all these messages.
- Feedback should have an effect. In the case of feedback given by students, they expect that the course improves through their feedback.
- You want to increase the students' skills and competencies like sociability, reflection on various viewpoints and communication skills.
- Students can comment on their peer's feedback but since they don't have the final authority (e.g., for giving the grades), their feedback is often not considered as binding.

**Solution:** In at least one lecture, explicitly encourage the students to use the feedback channel. Define the kind of feedback that will be helpful. Make clear how fast you will reply to the student's feedback and how extensive the replies can be. Show the students how their feedback changed the course and how it helped them and others to learn.

You may decide to create a compiled response referring to multiple feedbacks of the students, especially when different students provide feedback on related topics. In such a case, you should ensure that the references to the feedback become visible so that the students can see that their feedback was heard.

**Technology Support:** Most online platforms (such as Moodle, CommSy) or even a simple net news server provide a component for THREADED DISCUSSIONS (Schümmer & Lukosch, 2007). These can be used to collect feedback and reply to it. Some systems also allow to FLAG (Schümmer & Lukosch, 2007) or tag a message. In such a case, you can flag messages that you considered as

correct, which would be the most lightweight form of telling the students that their messages were heard.

As long as you personally welcome positive and negative feedback, you should share the feedback with the whole class instead of having it sent to you by personal e-mail. This allows other students to see their peer students' feedback.

**Discussion:** If feedback is seen as an important part of learning in higher education, then it should be addressed explicitly. The fact that you appreciate feedback and encourage students to improve the course can be another incentive for the students to take a closer look at the course's subject. By saying that you need the students' feedback, you also emphasize on the fact that you can also make mistakes. This may invite students to critically evaluate the content you teach them.

**Examples:** In our introductory programming course at the University of Hamburg, we have been using the online platform CommSy for several years, mainly to distribute the digital material for the exercises and lectures. Because the platform offers a discussion forum as well, a feedback channel is provided implicitly. But this channel was not used by the students. Only after we explicitly encouraged the students during one lecture to make use of this channel, students started to comment on the problematic aspects of the course.

#### **Related Patterns:**

- **STUDENT'S FEEDBACK FIRST:** You may decide to delay your responses to the feedback given by the students and encourage other students to reply first. However, you should still ensure that the students understand that YOU hear their feedback.
- **PEER CORRECTION** can be used if there are still too many students so that you cannot reply to them individually. In that case, other students act as your delegates. But you should still stay aware of the feedback given by the students.
- **TIME FOR REFLECTION** (Manns & Rising, 2005) takes a broader perspective on the importance of feedback. The authors argue that any long-term activity should be interrupted by phases of reflection. The student feedback can serve as a trigger for such reflection episodes. Once students express feedback, they trigger teachers (and other students) to reflect on their current practice as well.

## **2.2 FEETIQUETTE**

**Context:** Any teaching context in higher education, independent of the group size, course format or distribution of the participants.

**Problem: Feedback can be harmful if done wrong.**

#### **Forces:**

- If students are very unhappy with a teaching situation they can tend to act very emotional.
- Unfair feedback can lower or even destroy motivation, both on the side of the teacher and of the student.

- Students can be forced to attend the course because it is mandatory in their particular program.
- Especially with weak contributions students can be nasty.
- Different People deal differently with personal critique. Some simply ignore any kind of feedback; others misunderstand even positive and constructive feedback as de-motivating.
- In distance education, feedback is typically expressed as text. If given without intensive reflected, it quickly becomes harmful and the lack of non-verbal clues makes it difficult for the receiver to understand how the feedback was intended.
- Potential anonymity or pseudonymity of computer-mediated feedback can lead to situations where the feedback giver becomes offensive since the feedback giver does not expect any negative consequences (note that this is a general problem in computer-mediated communication).
- Textual computer-mediated feedback can be persistent, which makes harmful statements in the feedback even more harmful (the feedback can still be read a long time after it was given and may thus still hurt).

**Solution: Explicitly introduce an etiquette for feedback (a 'feetiquette').** Give an appropriate explanation of how you understand the term feedback in your context, as part of a presentation and/or in written form. Point out the importance of feedback, but at the same time make clear that feedback is often subjective, i.e. highly influenced by the individual situation of the feedback giver. Explain when feedback can be given and what forms of feedback you consider as helpful in your course. Clearly state that feedback should always be constructive to be helpful.

**Technology Support:** If a technical feedback channel is identifiable as such, the infrastructure could ask for a confirmation, asking if the feedback typed in should really be sent and pointing out the possible consequences. The feedback etiquette could be shown in this dialog as well. This gives the student the explicit chance to reconsider the consequences of the feedback.

**Discussion:** By having a feetiquette, the students agree on a feedback culture that shares the values expressed in the feetiquette. Trying to adhere to the feetiquette makes them reflect on the way how they provide and receive feedback so that they finally improve the way how they provide feedback. Moderators (i.e., teachers) can give meta-feedback on feedback using the feetiquette as a FEEDBACK SCAFFOLD for the meta-feedback.

**Examples:** At the University of Hamburg, we apply this pattern in a regular course on design patterns, where students present use cases of design patterns to each other in a special form called 'teachlet' (Schmolitzky, 2005). We give a short presentation on the term feedback and hand out a one page description of our feedback etiquette for the course.

An example, which is not in the educational domain, is the feedback etiquette for e-Bay (online at [http://reviews.ebay.com/FEEDBACK-ETIQUETTE-netiquette-POSITIVE-neutral-NEGATIVE\\_W0QQugidZ10000000000079805](http://reviews.ebay.com/FEEDBACK-ETIQUETTE-netiquette-POSITIVE-neutral-NEGATIVE_W0QQugidZ10000000000079805)). It explains dos and don'ts for giving feedback on a transaction.

**Related Patterns:**

- NETIQUETTE (Schümmer & Lukosch, 2007) focuses on rules and guidelines for interaction in computer-mediated contexts. Good Netiquettes can become part of the FEETIQUETTE in distance education contexts.
- FAQ (Schümmer & Lukosch, 2007) describes how information about a community's culture and norms can be given based on questions asked by members. The FEETIQUETTE can fulfil the role of a FAQ for procedures on giving feedback.
- FEEDBACK SANDWICH (Eckstein et al., 2002) argues to start and end with positive feedback. It is one typical rule that can be part of a FEETIQUETTE.

**2.3 FEEDBACK SCAFFOLD**

**Context:** A teaching situation with a lot of interaction between students (presentations, discussions). Feedback can be given on several levels of abstraction, from presentation style to presented content.

**Problem: Students do not know how they should structure their feedback. They fail to distinguish important aspects that require feedback from aspects that do not need further comments.**

**Forces:**

- If feedback can be given on several levels of abstraction it can be difficult to be systematical.
- Students often feel that there was something wrong with a presentation, but cannot put a finger on it.
- Feedback is sometimes given in a destructive manner.

**Solution: Provide an Outline of a good feedback.** Pass this outline together with an explanation how to flesh out the different parts of the scaffold with feedback.

**Technology Support:** Technology support can help by providing online forms for feedback. It should give the teachers means for adding their questions to a template (e.g., as Wiki templates proposed by Haake et al. (2005)) from which the students then create their feedback documents. The technology should however be open enough to allow free-form feedback (at least one free text field).

**Discussion:** The scaffold gives the students a structure by which they can develop their feedback, but an outline can also block creativity. If students feel that they need to follow a certain scheme they can be tempted to not think outside of it. You should thus explain the students that they are free to extend the structure proposed in the scaffold.

**Examples:** At the University of Hamburg, we apply this pattern in a regular course on design patterns, where students present use cases of design patterns to each other in a special form called 'teachlet' (Schmolitzky, 2005). Because teachlets consist of several parts that are clearly distinguishable feedback can be very focused on these parts. It is then helpful to have some

standard questions at hand that have proven to be helpful in finding good feedback comments. We provide such questions in a one page document (different from the feedback etiquette).

At the University of Hagen, we provide FEEDBACK SCAFFOLDS for PEER GRADING (Eckstein et al., 2002). The scaffold guides the students through the review process and shifts their attention to different aspects of the feedback process.

Most review processes for scientific publications provide a review template. Reviewers use this template to provide constructive feedback on different aspects of the reviewed paper.

All evaluation questionnaires are examples for the FEEDBACK SCAFFOLD. The FEEDBACK SCAFFOLD pattern, if applied correctly in this area, would help the designers of the questionnaires to provide space for free feedback (e.g., an open-ended text field).

#### **Related Patterns:**

- SOFT SCAFFOLDING (Pachler et al., 2009) highlights the importance of providing scaffolding structures for e-learning settings.

## **2.4 PEER RATED COMPETITION**

**Context:** During your course, you have students working in small groups on a larger design problem. All groups work on the same problem.

**Problem:** Students tend to focus on their own solution and to ignore other, potentially better ways of solving the problem.

#### **Forces:**

- Students have different levels of proficiency upfront.
- Some courses are not graded. If reaching any solution is the main criteria for passing the course, students focus on reaching one solution, ignoring the solution's quality.
- Some students love competitions, especially ambitious students.
- Students should be aware of the possibility of different quality criteria.

**Solution:** Let the students select the best solution through several rounds of selection and offer a reward for the best solution. Let them first present their solutions to two or three other groups and make them elect the best solution out of these. Let them decide based on quality criteria that are explicitly set up by the students. Make all the winners present again in front of the whole course. Let the solutions be rated at the end of the course and offer REWARDS (Schümmer & Lukosch, 2007) for the best solution. You can define more than one category where teams can win.

**Technology Support:** You may decide to have one or more rewards where the class is the jury (extending the idea of a (RATED) EXHIBITION). In such cases, you can support the students in the election of the winners by using a VOTING (Schümmer & Lukosch, 2007) system.

#### **Examples:**

In 2004, we taught a beginners course on Java programming at the FernUni Hagen. Since the students had very diverse previous knowledge, we decided to require rather basic programming skills to pass the course while at the same time creating a competition that should encourage good students to create their best possible result in the course.

The task was to create an event location planner that helped organizers of an event to place the chairs in a way that the audience could have a good view of the stage. Students worked on this task individually and presented their solutions in micro exhibitions with 8 students in each exhibition. From each exhibition, the 8 students selected the most interesting application with respect to the design of the graphical user interface and the performance reached by the system. The winners were then presented in a course-wide exhibition and all students were allowed to vote for the best solutions.

At the end, 4 students received prizes and their solutions were placed on the course's web site (<http://web.archive.org/web/20050302111927/kalu.fernuni-hagen.de/1580+82+84/propra2004.html> ). Both, novices and advanced students reported that they were amazed by the quality and creativity that was present in the winning solutions and perceived new motivation for their next courses.

At the University of Hamburg, we conduct a similar competition in the second semester course on programming, called *Software Development 2*, since 2007. All students have to implement parts of a film schedule planning software for a real cinema, working in groups of up to six. Towards the end of the semester, the teams have to present their results to each other and elect the best contribution. The prizes are tickets for the cinema in Hamburg.

### **Related Patterns:**

- EXHIBITION focuses more on the presentation and exchange of the students' solutions in diverse areas.
- LETTER OF RECOMMENDATION (Schümmer & Lukosch, 2007) shares the focus on rating other people's contributions. The goal of the LETTER OF RECOMMENDATION is to identify experts for a specific subject or people who have a specific experience. In contrast to this, the ratings in the PEER-RATED COMPETITION use the ratings as a motivational means.
- GOLD STAR (Eckstein et al., 2002) is a more general pattern for praising good students.
- ROUND AND DEEP (Pachel et al., 2009) highlights that the experiences of peer students can augment the content taught in a course. Students should learn from other students' views by understanding the different approaches towards a specific challenge.

## 2.5 STUDENTS' FEEDBACK FIRST

**Context:** You are discussing students' solutions.

**Problem:** Discussions of solutions are often dominated by the teacher. Teachers are often pressed into the role of experts and their feedback is sought first. Students fear to criticize other students' work since they do not always know if their critique is correct.

**Forces:**

- Students are less confident in the subject than educators.
- Since there are more students than educators, there will be more opinions and different (creative) ideas and impressions present in the class than the educator expects.

**Solution:** Let students comment on peer students' solutions before providing feedback yourself. Appreciate the feedback and strengthen the confidence of the students in the solution by an explicit summary or confirmation at the end of the discussion.

**Technology Support:**

In newsgroups, you should give students the time to answer a subject on their own first. Make sure that students know that you will not respond immediately.

**Discussion:**

Danger spot: Students may get the impression that the teacher has nothing to say about the solution.

**Examples:** Newsgroup discussion at the FernUniversität in Hagen: Courses at the FernUniversität in Hagen typically have an attached newsgroup where students can ask questions and discuss solutions. Instructors intentionally leave questions unanswered for one or two days to give students the opportunity of providing an answer. Later in the discussion thread, the instructors step in to approve the solution.

Subject	Author	Timestamp
Diskussion Lösungsvorschläge		09.11.2008 14:37
Re: Diskussion Lösungsvorschläge	Frank Heide	10.11.2008 17:30
Re: Diskussion Lösungsvorschläge	Christian Heide	10.11.2008 18:38
Re: Diskussion Lösungsvorschläge	Frank Heide	10.11.2008 20:12
Re: Diskussion Lösungsvorschläge	Christian Heide	11.11.2008 19:06
<b>Re: Diskussion Lösungsvorschläge</b>	<b>Christian Heide</b>	<b>12.11.2008 11:49</b>
Re: Diskussion =?ISO 8859-1?Q?L=F6sungsvorsc...	Frank Heide	13.11.2008 13:19
Re: Diskussion =?ISO 8859-1?Q?L=F6sungsvo...	Christian Heide	13.11.2008 23:12
Re: Diskussion =?ISO 8859-1?Q?L=F6sung...	Frank Heide	14.11.2008 19:34
<b>Re: Diskussion Lösungsvorschläge</b>	<b>Christian Heide</b>	<b>12.11.2008 11:42</b>
Re: Diskussion Lösungsvorschläge	Frank Heide	10.11.2008 20:15
Re: Diskussion Lösungsvorschläge	Frank Heide	10.11.2008 20:30
Re: Diskussion Lösungsvorschläge	Christian Heide	11.11.2008 19:13
Re: Diskussion Lösungsvorschläge	Frank Heide	12.11.2008 07:35
Re: Diskussion Lösungsvorschläge	Christian Heide	12.11.2008 21:29
Re: Diskussion Lösungsvorschläge	Frank Heide	14.11.2008 19:35
<b>Re: Diskussion Lösungsvorschläge</b>	<b>Christian Heide</b>	<b>12.11.2008 11:27</b>
Re: Diskussion Lösungsvorschläge	Frank Heide	14.11.2008 19:40
Re: Diskussion Lösungsvorschläge	Frank Heide	14.11.2008 15:56
Re: Diskussion Lösungsvorschläge	Frank Heide	14.11.2008 19:59
Re: Diskussion Lösungsvorschläge	Christian Heide	16.11.2008 15:36
Re: Diskussion Lösungsvorschläge	Christian Heide	16.11.2008 15:39
Re: Diskussion Lösungsvorschläge	Frank Heide	16.11.2008 20:22
Re: Diskussion Lösungsvorschläge	Frank Heide	16.11.2008 20:27
EA3 - Aufgabe 3	Frank Heide	06.11.2008 12:35
Re: EA3 - Aufgabe 3	Frank Heide	06.11.2008 13:10
EA3-A1	Michael Heide	16.11.2008 15:22
Re: EA3-A1	Frank Heide	16.11.2008 18:50

Figure 3 : Newsgroup threads in the course on distributed systems.

Figure 3 shows a brief excerpt of a newsgroup discussion that took place in the course on distributed systems. One student initially presented his solution with the goal of getting feedback (on 9<sup>th</sup> Nov). Instead of providing an instant answer, the instructor waited three days before she commented the solution (post from 12<sup>th</sup> Nov, 11:27). She also commented the comments made by other students whenever this was required.

**Related Patterns:**

- PEER CORRECTION also argues that you as a teacher should step back in order to encourage students to make up their mind regarding other students' solutions.

**2.6 PEER CORRECTION**

**Context:** You are teaching a large course where students have to solve assignments. The solution process involves creativity; the final result is often a written text that needs to be interpreted by the reader in order to grade it.

**Problem: Students expect to receive feedback on their solutions. But you as a teacher have insufficient time to provide detailed feedback on all solutions.**

**Forces:**

- You have no resources for correcting all assignments.
- Students request that their individual solution is corrected by a knowledgeable person.
- Students have gained competencies by creating a solution.
- You want to empower students to critically comment content related to your course's subject.

**Solution: Let students give feedback on other students' solutions.** Create a prototypical solution and give that solution to the correcting students. Correcting students can use this solution as a guideline for their feedback. If appropriate, you can give the students additional guidelines such as a list of important keywords that should be discussed in the solution. Each student selects a peer student (or is assigned by you to a peer student) and provides feedback according to your solution guidelines. Note that the different roles of the correcting student and the corrected student can also be filled by groups of students. This would mean that a small group of students collaboratively creates a solution and that another group of students then does the inspection.

**Technology Support:** Technology support can help during the coordination of the correction process. Special attention should lie on support for GROUPS, i.e., the corrected student should be aware of the correcting student and have means for interacting with the correcting student.

An example for a concrete implementation provides the student with functionality for submitting his solution to a pool of correcting students. The correcting student may select this solution for correction or the teacher may assign the solution to the correcting student. From then on, the correcting student and the author of the solution form a COLLABORATIVE SESSION that



allows them to interact. You may decide to equip the students with a collaboration space, e.g., a ROOM, in which they will find the approved solution and advice for performing the grading.

The correcting students need means for adding comments to the solution. This can be handled by SHARED ANNOTATIONS. The final result of the correcting student's review can be structured with a grading template. This especially eases the teacher's task of reviewing the review.

**Discussion:** The correcting student takes over parts of your responsibility as a teacher. He can do this because he received additional input (the solution guidelines) or because he/she has gone through the assignments in a previous iteration of the course. In both cases, the student benefits from correcting other students' solutions because he/she has to analyze, understand, and critique the solutions produced by peer students. Students also get an impression of other students' approaches for the assignment.

A potential danger spot of the pattern is that the correcting student misinterprets or simply does not understand your solution guidelines. Make sure that a corrected student can contact you if he/she has any doubts regarding the correctness of the feedback.

**Examples:** At the FernUniversität in Hagen, we applied the CORRECTING STUDENTS pattern in a course on operating systems in the years 2006 to 2009.

**Related Patterns:**

- GURU REVIEW (Manns & Rising, 2005) proposes to invite well-known experts to review new ideas in an organization.
- FEEDBACK (Eckstein et al., 2002): Feedback is given by the teacher to ensure that the students understand where they are at fault.
- SELF TEST (Eckstein et al., 2002): Students should assess themselves using questionnaires and prepared answers.
- PEER EVALUATION (Derntl, 2004): The PEER-EVALUATION pattern can be considered as a predecessor of our pattern. PEER CORRECTION extends the PEER EVALUATION pattern by proposing a concrete interaction process and highlighting the need for FEEDBACK SCAFFOLDS.

## 2.7 (RATED) EXHIBITION

**Context:** You have students working on larger design problems. They work alone or at most in pairs. Each person or pair is working on a different problem. The typical example is several students that are working on their final thesis.

**Problem: Students that work very focused on their own problem tend to loose interest in other problems or solutions.**

**Forces:**

- Students construct creative solutions.
- Students can be shy to show the results of their work.

**Solution: Let students prepare presentations of their finished work, e.g. in form of a poster. Organize a gathering where the students can (but do not have to!) present their results. Identify and honor outstanding presentations.**

This encourages other (younger) students to reflect on the quality of good solutions and thereby better understand what is required by a good solution. Besides presenting the result, you may also ask the students to present their process that led to the result.

**Technology Support:** In co-located settings, you may create an exhibition where students present their results on flip-charts, posters, etc. In distributed settings, you may ask the students to upload their solutions to a Wiki. If the solution includes a presentation, you may also ask the students to create electronic versions of their presentation using presentation systems like slidecast. Encourage the students to comment other students' solutions.

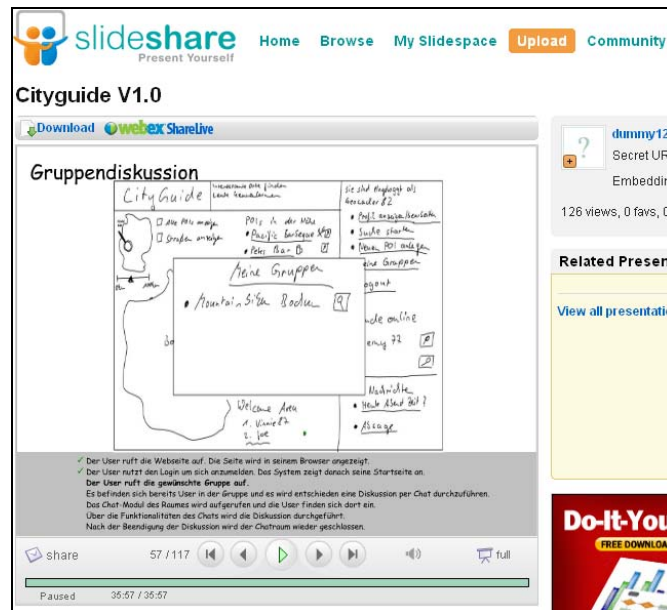
Since the discussion should have time to reflect, you may use a forum attached to each solution where the solution can be discussed.

**Discussion:** You need to decide when you make the solutions available. If you teach the same class in subsequent years, you may use the solutions of previous years' students and make them available before the solution is due. However, this may hinder the development of new / different solutions.

Ensure that you explain the added value of creating a presentation of their work for a wider audience.

**Examples:**

In the course Designing Cooperative Systems taught at the Universities of Applied Science Dortmund and Cologne/Gummersbach, the students were asked to successively build a prototype for a cooperative system. While the lecture evolved, students were frequently asked to show their intermediate solution to others. This bound the lecture to the cases created by the students and helped the students to better understand the subject. At the end of the course, the students created a SlideCast presentation of their prototype and all prototypes were finally visited together on slidshare (<http://www.slidshare.com/>) during a virtual lecture session where students provided their comments both in written form using the course's mailing list and orally within a telephone conference (Figure 4).



**Figure 4: Using a SlideCast as one way to exhibit the students' results.**

In the Department of Informatics at the University of Hamburg, once every year students are asked to present the results of their thesis projects at the EXPO, a half day event that is open to the public and actively visited by the members of the Department (staff and students). Typically students prepare posters, but sometimes they also prepare demos. The best three presentations win a prize (money).

#### Related Patterns:

- PEER RATED COMPETITION has a special focus on the peer rating. While rating can also be an aspect of the EXHIBITION, the main focus of an EXHIBITION should rather be the exchange of experiences.
- HALL OF FAME (Schümmer & Lukosch, 2007) displays successful users of a community. In an educational context, we prefer to rate the students' solutions.
- HOMETOWN STORY (Manns & Rising, 2005) discusses how new ideas can be exchanged in organizations. The authors propose to organize an informal highly interactive session in which new ideas are presented and shared among practitioners.
- STUDENT ONLINE PORTFOLIOS (Eckstein et al., 2002): Same direction but focus on providing a place for publishing results. We assume that there are additional requirements, especially that the exhibited artifacts should invite peer feedback.
- Classroom display (Pachler et al., 2009) also argues for an event at which student solutions are made accessible to peer students. The pattern concentrates on the design of an e-learning environment for supporting the exchange of the solutions. Rating is not discussed in the Classroom Display pattern.

### 3. Conclusion

This paper is intended as a systematic collection of best practices for gaining and handling feedback by students. We consider it as a first step towards a larger collection of patterns that will encourage students to become active and critical partners in the next generation of educational settings. In these settings, teachers become facilitators of learning activities. We experienced that feedback plays an important role in these settings and hope that the patterns of this paper can make teachers more aware of the importance of feedback.

**Acknowledgements:** We thank our shepherd Symeon Retalis and the participants of the writers' workshop at EuroPLOP 2009 for their feedback.

### 4. References

- Bergin, J., Eckstein, J., Manns, M.L. and Sharp, H.: *Patterns for Active Learning*. Proc. PLoP '02, Monticello, Illinois, 2002.
- Bergin, J.: *Active Learning and Feedback Patterns*. Proc. PLoP '06, Portland, Oregon, 2006.
- Derntl, M.: *The Person-Centered e-Learning pattern repository: Design for reuse and extensibility*. Proc. World Conference on Educational Multimedia, Hypermedia and Telecommunications, Lugano, Switzerland (pp. 3856–3861), 2004.
- Derntl, M.: *Patterns for Person-Centered E-Learning*. IOS Press, Amsterdam, 2005.
- Eckstein, J.: *Workshop Report on The Pedagogical Patterns Project: Successes in Teaching Object Technology*. Proc. OOPSLA '99 Educators' Symposium, Denver, CO, 1999.
- Eckstein, J., Bergin, J. and Sharp, H.: *Feedback Patterns*. Proc. EuroPLOP '02, Irsee, Germany, 2002.
- Haake, A., Lukosch, S., and Schümmer, T.: *Wiki-templates: adding structure support to wikis on demand*. Proceedings of the 2005 international Symposium on Wikis (San Diego, California, October 16 - 18, 2005). WikiSym '05. ACM, New York, NY, 41-51. DOI= <http://doi.acm.org/10.1145/1104973.1104978>, 2005.
- Manns, M.L., Rising, L.: *Fearless Change*. Pearson Education, Boston, MA, 2005.
- Martin, D., Rodden, T., Rouncefield, M., Sommerville, I., Viller, S.: *Finding Patterns in the Fieldwork*. Proc. of ECSCW'01. Bonn, Germany, Kluwer, 2001.
- Pachler, N., Mellar, H., Daly, C., Mor, Y., Wiliam, D., Laurillard, D.: *Scoping a vision for formative e-assessment*. JISC project report, <http://telearn.noekaleidoscope.org/open-archive/browse?resource=1875> (2009)
- Schmolitzky, A.: *A Laboratory for Teaching Object-Oriented Language and Design Concepts with Teachlets*. Proc. OOPSLA Educators' Symposium, San Diego, CA, 2005.
- Schmolitzky, A., Schümmer, T.: *Patterns for Supervising Thesis Projects*. Proc. EuroPLOP 2008, Irsee, Germany, 2008.
- Schümmer, T., Lukosch, S.: *Patterns for Computer-Mediated Interaction*. John Wiley and Sons, Chichester, UK, 2007.
- Schuler, D.: *Liberating Voices, A Pattern Language for Communication Revolution*. MIT Press, 2008.

# Pattern for Graduate Student Company

*Petko Ruskov<sup>1</sup>, Milena Stoycheva<sup>2</sup>, Yanka Todorova<sup>3</sup>*

<sup>1</sup>*Sofia University, FMI, 125 Tzarigradsko shosse, Sofia, Bulgaria, [petkor@fmi.uni-sofia.bg](mailto:petkor@fmi.uni-sofia.bg)*

<sup>2</sup>*JA Worldwide and JA Bulgaria, Sofia, Bulgaria, [milena@jabulgaria.org](mailto:milena@jabulgaria.org)*

<sup>3</sup>*Sofia University, FMI, 125 Tzarigradsko shosse, Sofia, Bulgaria, [todorova.yana@gmail.com](mailto:todorova.yana@gmail.com)*

## **Abstract**

Graduate Student Company (GSC) is an innovative educational model to teach entrepreneurship competencies based on the learning-by-doing methodology. It represents the formation of a student-operated mini-company which functions like a limited company, and the members buy shares in the company through personal contributions. During the process of learning, the students understand the opportunities of how to start and run a company and go through the whole company life cycle from establishment to closure.

Our approach involves review and observation of European and global practices in the field of entrepreneurship education; practical implementation of the Junior Achievement –Young Enterprise (JA-YE) methodology for the Graduate Student Company. The main findings represent the experience and best practice of the Junior Achievement -Sofia University partnership and propose a model for future implementation in both engineering and scientific faculties. Learning-by-doing methodology is a key success factor for teaching entrepreneurship. Patterns are seen as a good repeatable practice to encourage involvement and mentorship on part of the business community and improve the learning environment and entrepreneurship ecosystem. The target audience are developers of curricula, teachers and business angels. The GSC pattern is expanded and implemented in 10 more universities in Bulgaria during 2008-2009 academic years.

## **Keywords**

*Graduate Student Company, life cycle, learning patterns, learning-by-doing*

## **1. Introduction**

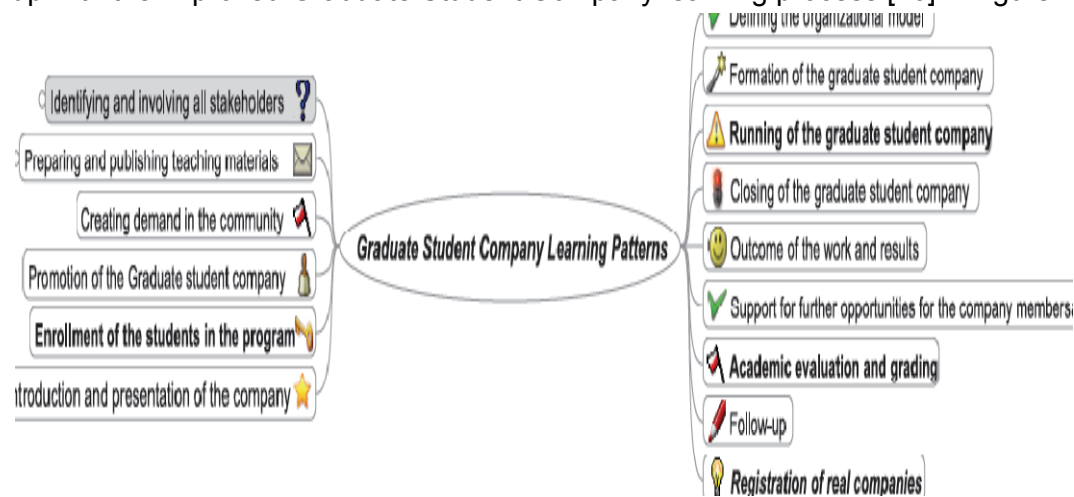
The educational processes like the processes in the service industries share two comparatively common features: intangibility and interactivity. Graduate Student Company life cycle process – which is presented in this paper as a pattern - typically has progressed further in terms of intangibility and interactivity than most other sectors being an innovative representative of the educational sector. Like the other service industries it has the following main characteristics according to Miles [13]: co-terminality, low portability, information intensity, fundamental processes, knowledge intensity, and market relations. The education of students from scientific and technology faculties in technology entrepreneurship in this complex and dynamic environment requires the evaluation and response of all of the above factors and their impact on the education. Therefore, we made a conscious decision to utilize the pattern approach which later on will help us to share and repeat the GSC practice.

## **2. Graduate Student Company Organizing Principles**

The organizing principles we have chosen represent a structure which is a set of syllabus, patterns, and academic and business services [12, 14, and 15].

- **Syllabus** is an organizing framework that groups together separate learning lessons and themes into one specific university course.
- **Patterns** are organizing concepts that facilitate dynamic mapping from stakeholders requirements to learning-by-doing educational designs.
- **Services** are a comprehensive set of activities that ensure the smooth and successful implementation of the academic processes and the related administrative and business procedures.

In the context of all of the above and having set our strategy and goals, we reviewed our processes and analysed the best practices in the field [4, 5, 6, and 18]. As a result, we came up with the improved Graduate Student Company learning process [16] – Figure 1:



**Figure 1.** Graduate Student Company process

The above described process of the GSC life cycle represents a pattern which the authors recognize consists of several sub-patterns. It is our intentions in future works to dive deeper and uncovers and describes them.

## Patterns

In principle, a pattern describes a particular problem and its solution context [9, 10, 11]. Specifically in these books, a pattern describes a (set of) problematic situation(s) for the development team that can be fixed by applying an agile practice [1, 3]. Patterns are to be trusted because each one has been used several times with real development teams and projects — they are not one-off solutions or ‘good ideas’ that might or might not work [2]. Patterns are "discovered" and not "created."

The pattern format used in [1] by Elssamadisy is as follows and has been modified in some sections of the current pattern:

- **Name:**
- **Description:** a brief overview of the practice or cluster. Not included in this pattern.
- **{Dependency Diagram:}** A diagram showing inter-practice dependencies (for practices) and grouping (for clusters). Not included in this pattern.
- **Business value:** A sorted description of the business values this practice or cluster improves.
- **Sketch:** A fictional story that describes this pattern being used on a software development project in a given context.
- **Context:** The preconditions and environment where this pattern is useful. The context is a collection of invariants: issues that do not change by applying the pattern.

- **Forces:** Used to elaborate context and give specific issues that are problems (partially) resolved by this pattern. In fact, correct application of the pattern should remove many of the forces.
- **Therefore:** The pattern description.
- **Adoption:** Steps, ordering, guides to adopting this pattern.
- **But:** Negative consequences that can occur from applying this pattern.
- **{Variations:}** Different ways this pattern has been implemented successfully other than that described in the Therefore section.
- **{References:}** Where one can read more.

### 3. Student Company Patterns

Based on the 4 year experience with the student company model, a pattern for its implementation has been emerged - GRADUATE STUDENT COMPANY pattern. It is a higher level of abstraction of the GSC lifecycle and therefore can be further developed in many sub patterns – such as curricula, enrolment of students, company operations, closing the company, etc.

**Name:** Graduate Student Company Pattern

**Description:** Most sciences and technology universities today focus on theory that does not reflect the competences and capabilities needed on the market. The GSC set of patterns describes a learning-by-doing paradigm. Students are educated and learn through a process of creation, registration and running of a student company. The GSC functions as a real company (but is registered with the local Non-Government Organization entity office/ or government department) in real market conditions. The participants in the company receive advice and mentoring by experts and practitioners from real business organizations but run the company themselves. After the GSC educational process, successful companies proceed with a legal registration and establish as new start-ups. One of the major problems in educational system is the existence of a big gap between academia and business. It is also a recognised phenomenon that needs to be addressed on a pan-European and global level. The traditional university education is rather static and lagging behind the current dynamic economic environment. The classical knowledge life cycle from the moment of creation of knowledge and delivery to the students through books and professors is much slower than the direct contact with the knowledge creators. In the side of the continuum, for students to “plunge” directly in the real business without going through the practical learning curve will take this activity out of the educational life cycle. However, the implementation of the GSC model guarantees interaction and contact of the students with practitioners and consultants from the real business sector. Furthermore, it helps to eradicate the problem of lack of practical experience and work in real business conditions.

**Problem:** How does one motivate and educate science and technical student to have entrepreneurial competences and attitude.

**Business value:** The GSC added value to all stakeholders in the chain – students perform in a real market environment and build experience; universities cooperate with the existing respective industries and transfer competences; employers participate in the educational

process and update syllabuses; local and central governments are recipients of the benefits the active citizens of their community provide. The GSC delivers the real practice value to the customers: students, potential employers and community.

**Sketch:** Students from the Masters Degree Program “e-Business and e-Government” of the Faculty of Mathematics Information Science of Sofia University have the elective course Entrepreneurship – Graduate Student Company in their curriculum. Prior to the beginning of the academic year, JA Worldwide, the oldest and the fastest growing organization teaching economics, business and entrepreneurship in the world, organizes and sponsors training of teams of professors and students from technical and scientific faculties and universities in the country, delivered with the support and participation of business consultants and practitioners from the business sector. The latter conduct a promotion campaign among the students at large from these faculties and universities, and established 2 Graduate Student Companies which followed the real business educational process of Junior Achievement and created firms, formed departments, appointed staff, issued shares, designed and developed a product and produced it as well. At the closing stage of the process, the members of the company participated in a national, regional and global competitions for a business plan and a best company and the winners represented the country in the European and global wide competition for a best Graduate Student Company and business plans.

**Context:** Preliminary conditions for conducting this program (course and training) through this strategic pattern include securing that the course is a part of the elective offerings in the curricula; training of faculty selected students in real company setting by the business sponsors of the program.. It is necessary also that teaching materials, guides for consultants, teachers, and students are secured for the participants and provided to them. These conditions are enforced and true for the respective academic year and do not change while implementing the pattern. A GSC is not expensive to run and there is no risks for the stakeholders.

**Forces:**

- **Delivering value to the customer:** The education offered in the majority of the universities today is rather academic and theoretical and does not meet stakeholders’ expectation in reflecting the dynamics of today’s global market economy. But, involving business consultants in the process delivers the real practice value to the customers: students, potential employers and community.
- **Need of Multidisciplinary competences:** Traditionally, the teaching approach lacks integration of multidisciplinary competences, i.e. technology, business, human behavior, etc. However, in the carried out GSC format, students will practice and learn many lessons in creativity and innovation and apply skills and competences from different disciplines.
- **Measuring students’ performance:** The classical method for measuring educational results is rather subjective and dependent on a limited number of evaluators. However, the GSC model allows the students to have a self-evaluation through public presentations and filming of their performance, as well as feedback from participation on the real market or in national and international contests and competitions.

**Therefore/Solutions: Establish a Graduate Student Company (GSC).**

A GSC provides post-secondary students the opportunity to experience running their own company, giving them an insight into how their talents could be used to set up in business for themselves. GSC students acquire real experience of the world of business: creating and researching a business plan, taking responsibility and being accountable to their shareholders for the running of the company. Through this program students develop attitudes and skills necessary for personal success, lifelong learning and employability, plus

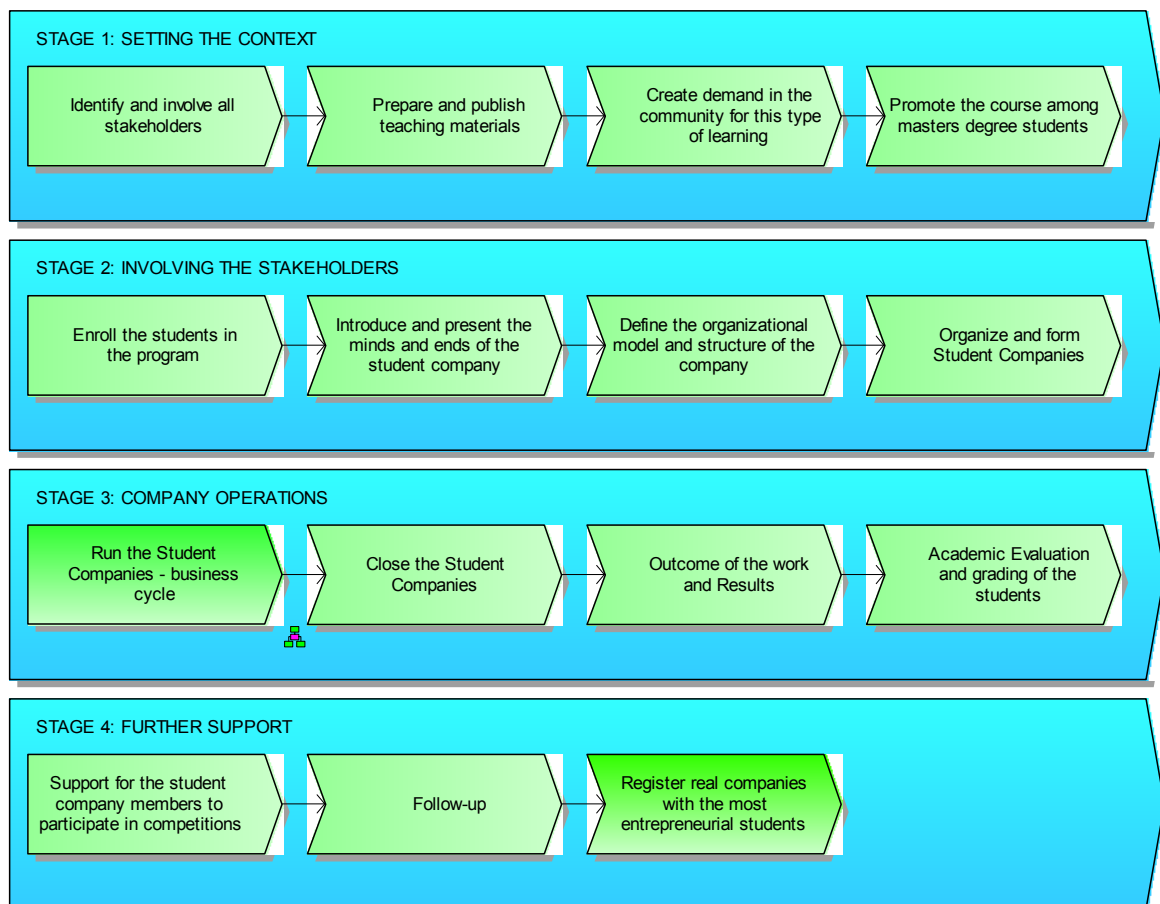


an understanding of how business works; gain an insight into self-employment, business creation, risk taking and coping with adversity, with advice and support of business consultants available.

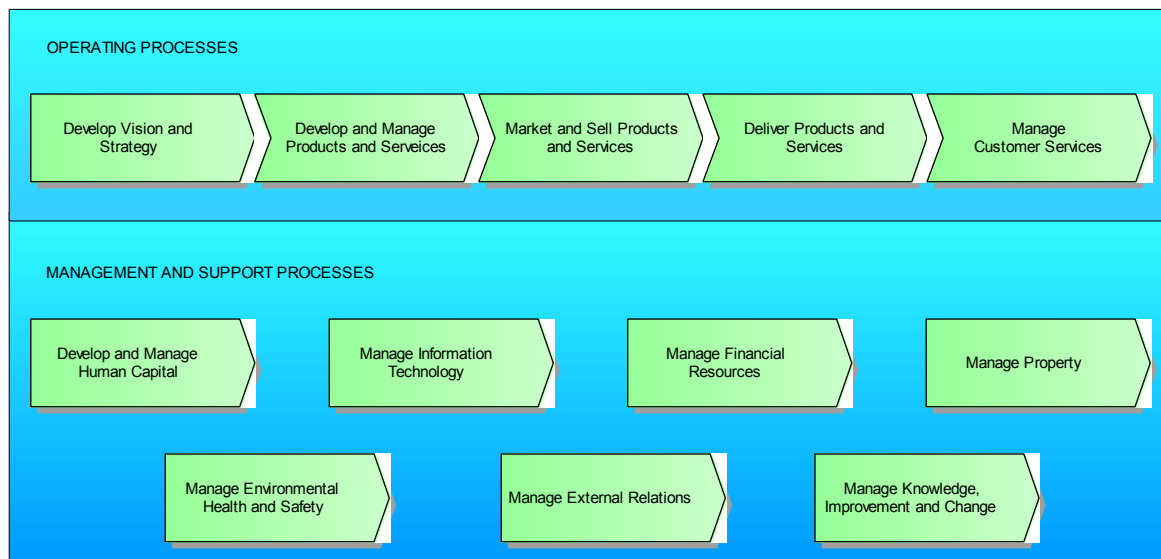
During the one and/or two semester educational process (depending on the specific local environment) the students form their own student company, manage it efficiently and achieve a real financial result at the end of the course. The Graduate Student Company functions like a limited company, and the members buy shares in the company through personal contributions. The number of participants is 12-20 people, and they operate in departments with Vice presidents as heads and a President as the CEO of the company. Two real life examples describing the processes and GSC case studies are presented in the Section 4 of the current paper.

**Adoption:** Most of the experience of the authors in the delivery of this program has been with students from high schools and some college students. The current program has been upgraded, adapted and innovated for Masters' degree students from technical and scientific faculties. The authors are convinced that such a course adds great educational value for the students with such background as it exposes them and develops their entrepreneurial mindset and business competency.

Our experience so far, has provided enough evidence that the student company life cycle pattern can be one of the steps to address this issue and apply an innovative and hands-on approach to it. The stages in the process of the GSC are many and follow the logic of a real company. To indicate where there are more patterns, the authors use capitals or terms of (potential) sub patterns. The Graduate Student Company Learning processes and their Stage Titles (Figure 2) as well as the Student Company Business Cycle based on APQC's [20] Process Classification Framework (Figure 3) are visualized and outlined below. Figure 3 shows a different view of the value added chain.



**Figure 2.** Graduate Student Company Learning processes



**Figure 3.** Student Company Business Cycle based on APQC's Process Classification Framework

**But:** Negative consequences: Specific factors and conditions which can create problems for the delivery of the pattern are related to the threat of too much theoreticing of the course. Negative consequences may be observed when implementing the pattern if the students are not properly consulted and they are allowed to divert from the educational process or they spend too much time on the company and leave their other obligations unattended. Additionally, the GSC course is an empowering course for students in terms of “going-for-real” through the process of running a business. However, the fact that there are limitations in terms of registration and cap on the capital creates a challenge for the students as they are functioning in real market conditions and yet have to comply with the requirements for less capital. One of the ways to overcome such negative consequences is to have closer interaction with the business consultants so that the students can feel and see the “value-added” that they are also creating for the business. Ultimately, the best solution to make the experience and model really “valid” is to secure a support structure for student start-ups that is supported institutionally by the government.

**Variations: The GSC allows the students to experience the whole idea of going through the life cycle of a real company.** The program can be delivered in the following variations:

- This program is usually delivered for students from high schools and college. However, these age groups lack some of the more sophisticated business experience (like with the master’s degree students), critical problem thinking, ability to see alternative solutions and decision-making.
- The authors are specializing in technology entrepreneurship and therefore find the pattern most appropriate for students from scientific and engineering backgrounds. However, there is also positive experience in applying it in economic and business faculties as it adds the hands-on experience and entrepreneurial approach in education.
- One or two semesters. If offered two semesters this will allow for more in-depth development of the product and its activity and better strategy for its development. It is a good option to use with unemployed specialists for the development of their entrepreneurial skills and competences [7].
- The best configuration, based on the authors experience is a team of approximately 15 students, more or less. This allows for a good-sized company organization which has all

of the main departments manned, and at the same time provides a format for effective communication in smaller groups.

- The course is not limited to running a Student Company where students buy shares – there are options to attract a sponsor and run the company with the funding/money of an investor as well. However, again based on the experience of the authors, the closer the delivery of the course to real market conditions and the more concrete the requirements are, the clearer and better the outcome is in terms of achieving specific results for developing entrepreneurial mindset and business competency in the students.
- It is possible to run a company in a virtual environment, especially if participants are from different countries – almost entirely. We believe though that for a real business to occur, having trust in ones' business partners and building relationships is critical. Therefore the model should provide for at least one face-to-face meeting.

**Literature:** You may read more on this pattern in:

Stoycheva M., Ruskov P., Entrepreneurship Education in Engineering and Scientific Faculties, Proceedings of the International Conference for Entrepreneurship, Innovation and Regional Development ICEIRD 2008, Skopje&Ohrid, Macedonia 8-12 may 2008, pp. 605-610, ISBN 978-9989-2636-4-4.

Ruskov P., Stoycheva M., Graduate Student Company Learning Patterns, Proceedings of the International Conference for Entrepreneurship, Innovation and Regional Development, ICEIRD 2009, 24-25 April 2009, Thessaloniki, Greece, pp.231-239, ISBN: 978 -960-89629-9-6.

**GSC known uses/applications:** After a 4 year-long experience in running the GSC, the authors developed the Student Company Lifecycle Processes and in applying them are presenting two case studies of successful GSCs.

- ***Case studies for graduate student companies***

### **YPM – Start Point**

Sector: Products

Employees: 15

Location: Bulgaria

#### **The big idea**

To be the most popular on-line and publishing media focused on issues for and targeting the Bulgarian students. To create community that enables students and business across Europe to share ideas and fill the gap between them.

#### **What they do**

The product "Start Point" Magazine is a monthly publication distributed for free. The second part of the product is a Start Point web site, where the magazine content can be found. Articles from the site are translated in English and accessible all over Europe.

#### **The challenge**

The participants in the YPM-Start Point Company started with a brain-storming session and analyzed the current environment in the country. They concluded that there is a gap between education and business in Bulgaria. They identified the challenge to create a bridge among them. Another problem is the need of popularization of Entrepreneurship among young people and its integration in the education programs.

#### **The solution**

**The GSC life cycle pattern provided the right format for the participants to address the problems and challenges that they identified. "Start Point" magazine is entirely intended**

for young people, who make their first steps in business or career. It provides information for current interesting events related to business and education. The purpose is to provide different perspectives for starting a new business and career growth.

On-line Magazine - All articles can be found on the magazine's web site. Each section has a forum, allowing readers to share their experiences and ideas between each other, related to the magazine staff.

"Start Point" is positioned on the market of free and online media, aimed at students and issued monthly. It is the first magazine of this kind, offering information on current and important topics, related to business and education. The articles are written in accessible language and the information is presented in an interesting way. Through its website, the magazine reaches much larger audience.

### **Issues arising**

To facilitate and create common interests between business and education YPM has to develop and promote a web site and a forum. In this way they allow effective communication between business and students. More over with organizing, assisting and participating in various initiatives, seminars and forums, they contribute to the building of better relationship between business and education.

### **The outcome**

Even with the first issue the company realized a positive outcome. It was achieved because of the strength and potential for development of an innovative idea of free student magazine, linking businesses and students. Proofs of this are the companies that supported the idea in the beginning.

### **What next?**

To achieve progress and growth YPM have to:

- Increase the volume, by getting more funds attracted by advertising or other sources.
- Reach a maximum number of readers.
- Increase the distribution area.
- Increase the revenue from advertising, through aggressive marketing strategy and maintaining a high quality of information provided by us.
- Increase the number of pages, by attracting new advertisers.
- Build a good team of authors through continually attract new young people.
- Build a partnership network of organizations and associations with similar to their goals.

## **Art & Innovation Group**

Sector: Products

Employees: 8

Location: Bulgaria

**The big idea:** To help reducing environmental pollution.

### **What they do:**

Alternative eco-product for the polyester bags - bags made from recycled paper.

### **The challenge:**

According to the Ministry of Environment and Water three billion and two hundred million plastic bags are bought in Bulgaria every year. If we tie together those bags (and assume that a bag is long on average 30 centimeters), we may reach the moon, then back to Earth and back to the half of the distance. A plastic bag is produced for 5 minutes. It is used for average 30 minutes, and for its degradation is needed between 100 and 400 years. The participants identified the problem of a lack of concrete solutions to deal with the environmental pollution.

### **The solution**

**The GSC Life cycle pattern showed the participants a way of creative thinking and approach to the problem. The team which was formed during the company life cycle generated an idea and found a way to implement it.**

Art & Innovation Group develop a strategy to establish the use of alternative eco-product - bags made from recycled paper.

Main advantages of paper bags:

- Recycled paper is cheaper than natural paper;
- They are designed to withstand load up to 10 kilograms;
- Provide weak elasticity: retain its shape during use;
- Advertisement inscriptions are visible for a long time;
- Provide excellent print quality on all types of printing machines;
- They are biodegradable, can be processed and turned into paper.

### **Issues arising**

To establish the use of paper products that meets the European ecological requirements. In this way they will avoid the use of packaging materials from polyester materials. Art & Innovation Group uses economic stimulus to establish the bags. As the advertisers have access to a wide segment of the market and via their advertisement they show socially responsible policy.

### **The outcome**

The company contacted most of the big retail chains and achieved signing contracts for partnership. The eco-product will be offered along with the polyester bags. They plan to sell 200 000 paper bags in the first year.

### **What next?**

Art & Innovation Group's goals are:

- To convince the representatives of large retail chains in the profitability and benefit from the use of such eco-product;
- To "force" the maximum number of advertisers to place their logo on the bags, and thus accelerate the realization of the idea;
- To change public attitudes through extensive marketing campaigns;
- To establish their brand and name of a socially responsible company.

## **5. Conclusions**

The paper presents the experience of the authors with a course -- the "Graduate Student Company", the process of introduction and implementation of the program in a university setting, the lessons learned, and the importance and use of this pattern as a good practice in the teaching and development of entrepreneurship education in the engineering and scientific faculties.

The described pattern facilitated the process of expansion and multiplication of the Graduate Student Company Life Cycle processes from one faculty of Sofia University to 10 faculties and universities and faculties in Bulgaria. The authors anticipate that it will receive a positive feedback and will be disseminated as an innovative approach of learning-by-doing in several European universities and can be promoted and successfully implemented in other universities in the region as well. The participation in the GSC and passing the patterns is an excellent opportunity for students and teachers to meet with executives, innovators, investors, industry suppliers and leading brands of the GSC stakeholders.

The described above cases represent the successful implementation and practices in 2 Graduate Student Companies and show the positive results from this process in Bulgaria.

This, as well as the achievements from the European Best Graduate Student Company

Competition, demonstrate evidence and proof for the value of the current GRADUATE STUDENT COMPANY LIFE CYCLE pattern.

### **\*Acknowledgement\***

We are most grateful to Christian Kohls and Andreas Fießler who shepherded this paper for EuroPLoP 2009.

The work on this paper has been sponsored by:

- Junior Achievement Bulgaria; Junior Achievement-Young Enterprise Europe
- UC 7FP Project SISTER: "Strengthening the IST Research Capacity of Sofia University", Grant agreement no.: 205030

### **References:**

1. Elssamadisy A. 2007, Patterns of Agile Practice Adoption, Crafting an Agile Adoption Strategy, InfoQ, 2007.
2. Engel J., Charron D.,(2006), Technology Entrepreneurship Education, Theory to Practice, Lester Center, Berkeley 2006.
3. Galic M., and all, Academic Edition: Applying Patterns Approaches Patterns for e-business Series, IBM Redbooks publication, 2007, SG24-7466-00.
4. JA Bulgaria Graduate Student Company, [http://www.linkedin.com/groups?gid=1221827&trk=hb\\_side\\_g](http://www.linkedin.com/groups?gid=1221827&trk=hb_side_g)
5. JA-YE Europe Strategy Brochure - "Vision 2010 and Beyond", <http://www.ja-ye.org/Download/STRATEGY%20BROCHURE%202007.pdf>,
6. Junior Achievement, Student Company, Entrepreneurial Spirit, Colorado Springs, Colorado, USA, (Translated and Adapted in Bulgarian by Junior Achievement Bulgaria, 2003), 2000.
7. Junior Achievement, Student Company, Entrepreneurial Spirit, Documentation, Colorado Springs, Colorado, USA, (Translated and Adapted in Bulgarian by Junior Achievement Bulgaria, 2003), 2000.
8. Junior Achievement, [www.ja-ye.org/graduatestudentcompany](http://www.ja-ye.org/graduatestudentcompany)
9. Kelly A. 2005a, Business Strategy Design Patterns, The Porter Patterns, <http://www.allankelly.net>
10. Kelly A.2005c, Business Strategy Patterns for the Innovative Company, The Porter Patterns, <http://www.allankelly.net>
11. Kelly A.2008, Business Patterns for Product Development, (EuroPLoP 2008), <http://www.allankelly.net>
12. Masters Science Degree Program of Sofia University [http://www.fmi.uni-sofia.bg/education/magisters/informatika-07-08/elektr\\_biz\\_el\\_uprav.pdf](http://www.fmi.uni-sofia.bg/education/magisters/informatika-07-08/elektr_biz_el_uprav.pdf)
13. Miles I., Patterns of innovation in service industries, IBM Systems journal, vol.47, No 1, 2008, pp.115-128
14. Robertson B., Sribar A., Enriching the Value Chain, Infrastructure Strategies Beyond the Enterprise, Intel Press IT Best Practices Series, 2002.
15. Ruskov P., Harris M., Todorova Y., Strategic Model for Master of Science program "Innovation and Technology Entrepreneurship", 3rd Balkan Conference in Informatics (BCI'2007), 27-29 September 2007, Sofia, Bulgaria, ISBN:978-954-9526-41-7, vol.1, pp. 501-512.
16. Ruskov P., Stoycheva M., Graduate Student Company Learning Patterns, Proceedings of the International Conference for Entrepreneurship, Innovation and Regional Development ICEIRD 2009, Skopje&Ohrid, Macedonia 8-12 may 2008, pp. (under review)
17. JA-YE Europe Surveys, 2009 <http://www.ja-ye.org/Main/Default.aspx?Template=TProjects.ascx&phContent=ProjectsList.ascx&CatID=213&ArtID=0&LngID=0>
18. Christensen C., Horn M., Johnson C., Disrupting Class, How Disruptive Innovation Will Change the Way the World Learns Mc Grow Hill, 2008
19. Ruskov P., Stoycheva M., Graduate Student Company Learning Patterns, Proceedings of the International Conference for Entrepreneurship, Innovation and Regional

- Development, ICEIRD 2009, 24-25 April 2009, Thessaloniki, Greece, pp.231-239, ISBN: 978 -960-89629-9-6.
20. APQC, American Productivity and Quality Center Process Classification Framework, <http://www.apqc.org/portal/apqc/site/?path=/research/pcf/index.html>.

## **Appendix1. Graduate Student Company Processes**

The stages (in bold italic) and sub processes in the educational process of the GSC are as follows: Identifying ***and involving all stakeholders***. The main stakeholders that have to be contacted and negotiate the future collaboration with them are:

Faculties from leading and diverse fields of science  
Consultants from the business community,  
Business partners,  
Government  
High schools and colleges  
Clients/employers

***Preparation and Publication teaching materials.*** Before the GSC life cycles start the next teaching materials must be ready:

Teachers Guide  
Students Guide  
Business Consultants Reference Guides

### ***Creating demand in the community for this type of learning***

Promotion among the corporate sector  
Promotion with the relevant ministries  
Promotion in the local communities and municipalities  
Promotion with local SMEs and NGOs

***Promotion of the course among university masters degree students.*** The GSC course is elective, according to the curriculum and next activities helps to attract more students to participate:

Joint business academia workshops  
Conferences and competitions  
Media publications

### ***Enrolment of the students in the program***

Evaluation of the capacity and competences of the future teams  
Company Team building  
Short student workshops

### ***Introduction and presentation of the minds and ends of the student company***

Introduction of the business models  
Introduction of the business plan concept

### ***Defining the organizational model and structure of the company***

Developing mission, vision and strategy  
Setting goals and roles of the departments

**ORGANIZATION AND FORMATION OF THE STUDENT COMPANY.** This process is very important and it can be sub pattern of the GRADUATE STUDENT COMPANY LIFE CYCLE pattern.

Choice of an organizational model

Registration of the company

Collecting of initial capital (e.g. through issuing of shares, sponsors, VC, etc.)

Establishment of departments

Selection of product or service

**RUNNING THE STUDENT COMPANY CYCLE.** The process is vital, train students on the subject of practical competences and it can be sub pattern of the GRADUATE STUDENT COMPANY LIFE CYCLE pattern.

Departments work

Department meetings and progress reporting

Company meetings and periodic reviews

Company processes improvement

Exit Strategy

**Closing of the Student Company.**

Departments final reports

Financial report

Liquidation

**Outcome of the work and Results**

Presentation

Evaluation

**Support for the student company members to participate in competitions**

Presentation of previous experiences and learned lessons

Research for different opportunities and competitions

Counselling, coaching, mentoring, and facilitation

**Academic Evaluation and grading of the students**

Test

Public presentation

**Follow-up.** Activities in this stage summarise the experience and go over learning lessons during the GSC life cycle:

Publishing and dissemination of results

Managing the graduate student company social network

**Registration of real companies by the most entrepreneurial students.** The last stage is the most exiting activity of the GSC and finalizes the efforts of the students and they become real entrepreneurs.



# Patterns for Product Line Engineering

Christa Schwanninger, Michael Kircher, Siemens AG

## Introduction

Software Product Line Engineering (PLE) has become an important way of building and reusing software. In PLE, the focus is shifted from building isolated products to building families of related products, while reuse is discussed not at an individual object level, e.g. libraries or components, but as a whole: organizational, process-wise, and also life cycle wise end-to-end from requirements to actually deployed variability at the customer. We consider only those platforms PLE efforts that are planned and prescribed to be reused in a dedicated product portfolio or to build solutions in a specific market segment, hence typical general purpose technology platforms, such as Eclipse, JBoss, or .NET, do not fall into this category.

While a lot of literature on PLE exists, there is still no consistent collection of easy to use and practical patterns. The patterns described in this paper build on the existing literature, like the book of Clements and Northrop [3] or Klaus Schmid's PhD Thesis [8]. With this initial collection of patterns we hope to motivate more patterns on PLE. The patterns describe how to centralize and objectify decision making, how to partition the domain into several sub-domains, how to set up your releases, and how to avoid dangerous complexity.

This paper is intended for any leadership staff concerned with the product portfolio content, organization, process, technology, architecture, or actual project execution of a PLE initiative. We assume some background in PLE. Refer to [1] or [2] for a more detailed introduction.

PLE has to be adapted to the specific organization at hand to be successful. Many factors influence how PLE is executed in a specific organization. The ones with the largest impact are: size of the organization, kind of business, number of product lines, maturity of the domain, and process discipline. While those factors influence PLE, they will in return get influenced by introducing PLE.

The patterns in this paper apply largely to mid- and large-scale organizations, as they typically have several product lines with several involved sub-organizations with varying business goals, which makes decision making a multi-dimensional optimization problem. Organizations with only a dozen involved staff do not have such issues; instead most of the challenge becomes a technical issue of how to deal with variability – which is not the focus of this paper.

The patterns deal with setting up the organization, the process and methodological best practices. This paper does not provide any analysis advice when to use a PLE effort or not, instead it assumes that a return on invest (ROI) analysis has been done in advance and that a PLE approach proved superior over classical one-off development.

For readers who are not familiar with PLE terminology, please read the glossary at the end of the pattern collection.

Copyright retain by authors. Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.

## Scoping Patterns

### Central Decision Making

#### Context

An organization is adopting product line engineering. Up to now it developed products or solutions in parallel, with copy&paste reuse, technological platform usage, or using a standardized infrastructure. Every product development group was responsible for deciding on the scope and release plan of its product, a task called product management. The organization wants to increase the benefit from reuse among products/solutions. The development of a set of reusable core assets is started. It has to be decided what these core assets should be: the scope of the reuse infrastructure. The reason for starting the product line is to maximize the return on investment (ROI) for building reusable assets for the whole organization. Figure 1 shows the desired state, one domain engineering activity that produces reusable assets and several application engineering activities that reuse these assets.

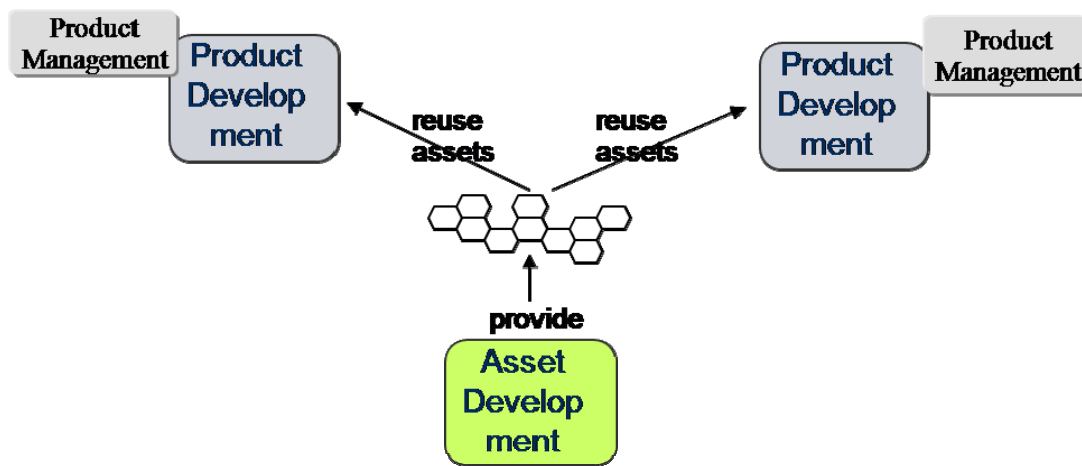


Figure 1: Relationship between domain and application engineering

#### Problem

Who decides what assets should be developed as reusable core assets and what assets should be kept product specific?

- Every product manager typically pushes his own products. Reuse is only interesting if it saves effort or otherwise benefits his products. Supporting his own products is what he is trained to do.
- Without moderation either the strongest in a group of product managers dictates what will be implemented as reusable core asset or the product managers agree on some compromise, e.g. everybody gets their most important features in. This selection is likely not maximizing the profit of the whole organization.
- Core asset development is confronted with all product managers and their corresponding priorities. For deciding about a product portfolio strategy deep market knowledge is required, which typically is not available in development. Development is inclined either to

give in on the pressure of the strongest or decide by themselves what has the highest priority, often according to purely technical criteria.

### Solution

Centralize decision making regarding scope and priorities. The best way to do this is introduce a new role in the organization, the product line product manager. His goal is to optimize the product portfolio, the solution scope, which shall be supported by the shared core assets. For bigger organizations the product line product management can also be staffed with several individuals.

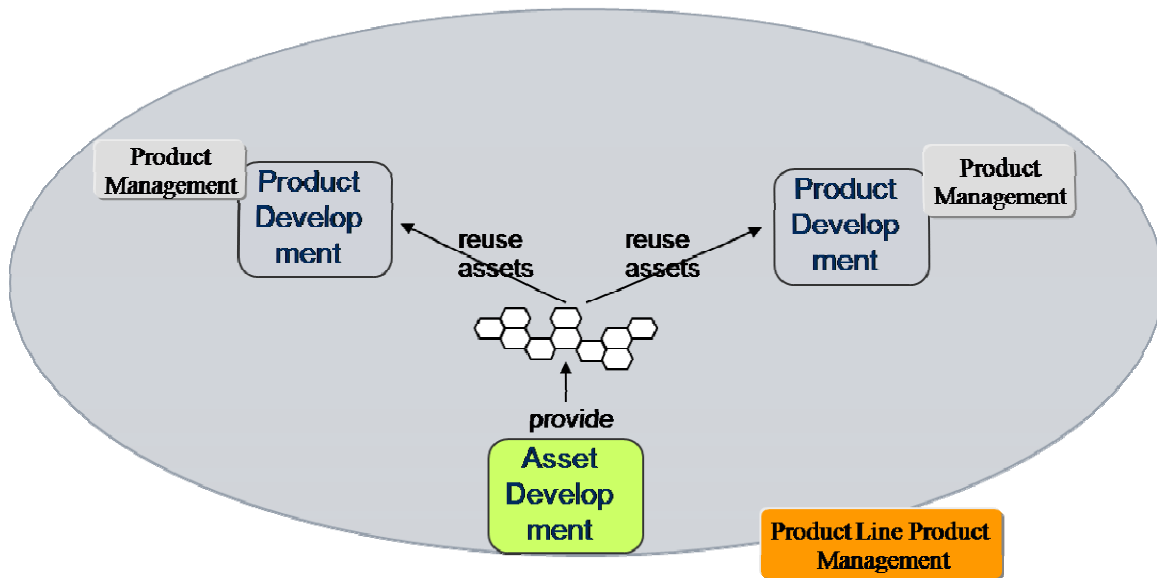


Figure 2: The Product Line Product Manager is responsible for the whole product line.

Core asset development should not decide about feature priorities. However, core asset development estimates the effort for features and analyzes dependencies among features, which is essential input for deciding on what should be implemented as core asset. Be aware that product management alone is not sufficient to find the overall optimized scope for a business, because constraints like quality, time, and cost might not be represented sufficiently; see the patterns *Balance Constraints* and *Collaboration between Problem and Solution Experts*.

The Product Line Product Manager is responsible for the overall product portfolio, therefore is neither part of any product development team, nor of the core asset development team. There might be conditions that render a dedicated, independent role not possible, e.g. if the organizations are separate profit centers with their own business targets and no operational responsibilities are at a common management level. The Product Line Product Manager would in this case be part of the core asset developing organization, which makes his position in the overall organization weaker. It requires additional process support to deal with this situation. This sub-pattern language is yet to be mined.

### Rationale

For effective core asset development a single responsibility for prioritization and scoping is required. In order to prioritize features and scope the product line to maximize the economic benefit of the

whole organization, knowledge about the market(s), competition, and customers is important for that role. Development typically lacks the full understanding of customer and business value.

This pattern works best if the core assets actually implement domain specific – customer visible – core assets. Such assets typically heavily influence the actual product portfolio. In contrast, if the core assets are mainly technical infrastructure assets, the product portfolio is only marginally influenced by the core asset scope. In such cases the product line architect role instead of the product manager, may perform the scoping, because he can adequately judge the scope of the mainly infrastructure core assets. The Product Line Architect then fulfills the role of the Product Line Product Manager as well.

The Product Line Product Manager role resolves the conflict amongst product managers and between product management and core asset development. We saw organizations that suffered from this conflict. In the end, the product lines broke apart and the whole effort of doing PLE was seen as a failure.

The Role Product Line Product Manager has to know how to manage the conflicting interests between his role and the product managers. This conflict can be resolved with *Comprehensible Decision Making*.

### References and Known Uses

This pattern is applied at a Siemens Healthcare group developing an imaging platform. The platform customers are different imaging product lines, which were heavily involved in defining the scope of the new platform from the beginning. However, there was no dedicated product manager for the platform. Hence, far too many features were assigned to the platform. There was no clear prioritization, and despite working hard, the platform customers were never satisfied with the scope of the platform. Two years later the organization installed a platform product management. The platform product manager is now responsible to define a common vocabulary and to drive scoping of core assets in collaboration with the product managers of the different products. He moderates scoping sessions and makes the scope and the rationale for the scope transparent to product developments. The organization managed to regain trust from the product groups. Installing a product line product manager improved the situation considerably for both, domain and application engineering.

A separate group at Siemens postal automation started a first platform about 10 years ago to benefit from reuse among their customer projects. There was no platform product manager, even no explicit scoping process. The platform users were not happy with the platform, quickly started to clone and own parts of the platform and the platform team was turned into a project team for one specific customer after three years. Today, the organization has a very strong product management that knows about the value of reusable assets and actively supports scoping and marketing of reusable assets. All bidding invitations go through the product line management group, that is a team of product line product managers and product line architects. The scoping is done in several steps: deciding which existing core assets will be offered to the customer, which core assets should be adapted and which should be built anew in case the customer project actually is won. Based on this the offer is made to the customer. When the customer accepts, the product line management team includes the effort for building the assets into the overall product line development plan.

## Comprehensible Decision Criteria

### Context

The pattern *Product Line Product Manager* was applied. A conflict potential is remaining between individual product managers and the Product Line Product Manager responsible for the whole product line portfolio.

### Problem

The product managers responsible for single product/solution are a group of peers who have all conflicting goals, which is promoting the specific products they are responsible for. Agreeing on a scope for the reusable base assets means resolving conflicts that arise from these different goals.

- Every product manager typically pushes only his own products. Reuse is only interesting if it saves effort or otherwise benefits his products. Supporting his own products is what he is trained to do.
- Core assets often get special funding. The cost is shared between all products/solutions. Often product managers want to unburden their product/solution specific budget and put as much development effort as possible to the core asset development. This way more budget is available for making own products better, cheaper or earlier on the market..
- When the selection of what shall be a core asset and what shall be product-specific seems not credible, development will not respect the scoping decisions. Development will decide on its own priorities.

### Solution

Define and communicate clear criteria for evaluating the value of features. The criteria should be derived from the organization's mission, vision, and strategy, because only then the decisions are in line with the business targets. Evaluate the cost and benefit for every feature and for every product according to these common criteria. Decision making boils down to calculating these values and deciding according to objective numbers.

The best way to do this is to define criteria for the benefit and the cost of each feature. Benefit criteria could be how much does the market request this feature or how well does it support the specific portfolio strategy. The cost side need not be an estimation of the development cost for the feature, since this may be hard to be calculated early in the process, but a mixture of estimated complexity, novelty, and effect on the current product line architecture.

Start with simple benefit/cost evaluation formulas and improve them over time. The product managers and architects that deliver the data for the formula typically differ in their accuracy and reliability of the data they provide. Install metrics comparing estimations with the later facts, e.g. estimated sales figures and actual sales figures for a product. The results are used as correction factors.

## **Rationale**

Most of the potential conflict can be mitigated when every stakeholder shares their business value data and the result is merely calculated. The product line product manager drives the process and controls how fairly features are rated and improves the evaluation criteria and formulas.

## **References and Known Uses**

The platform organization for imaging systems within Siemens Healthcare developed a formula for prioritizing features and calculating the cost of every feature. It is filled with information from product management and development to determine benefit and cost of every feature. Measures for a feature's benefit are for example whether it is innovative or just a "me too" feature and how often it would be reused, measures for its cost are for example the complexity and how crosscutting it is in the solution space. The results are taken to decide whether a feature should be implemented in the platform at all and what priority it should have.

Klaus Schmid suggests a similar procedure in [2]. In this context it was tested with two product lines of MARKET MAKER Software AG and with one product line in Bosch.

## Separate Sub-domains

### Context

The organization has to decide what will be developed as core asset and what will remain product specific – a typical scoping decision. Actually, the organization will also have to decide on a product portfolio that allows maximizing the benefit from reuse.

### Problem

Budget and time for setting up a core asset base is limited. If finding out which assets are worth to be developed as core asset takes too many resources, the product line might die before it even starts.

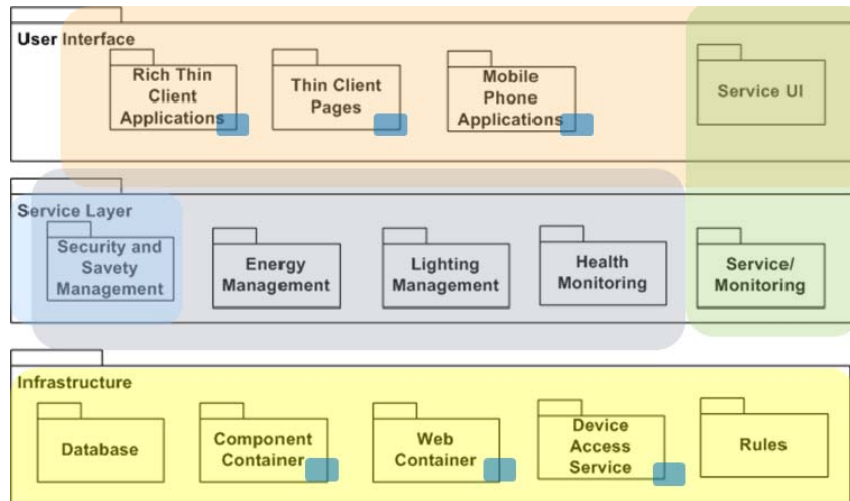
- Not all areas in a domain are equally stable and therefore suitable for reuse.
- Scoping is a pure planning activity. It is not productive and accounts as negative item in the return on investment calculation for a product line.
- Often a market evaluation is an important pre-condition for scoping. The scoping activity has to be performed in a time frame that together with implementing the scope is short enough to catch the market window that was looked at in the evaluation.
- The features in the core asset base have to be selected in a way such that it is possible to form a consistent reference architecture. They cannot result in completely independent pieces of functionality in solution space. It must be possible to architect and implement a coherent chunk of functionality.

### Solution

Divide the application domain in sub-domains that can be handled separately. The easiest way to identify sub-domains is to look at existing systems/products. Their division in subsystems is a good first subdivision of the overall domain. First estimate the reuse potential for every sub-domain based on past experience. Then continue closer investigations with one or two promising sub-domains. Find out what can be reused for these sub-domains and implement these assets.

The sub-domains might be application domain specific like telephony, short message service and data services in a mobile phone, or technical, like memory or power management.





**Figure 3: Consistent sub-domains may be vertical, horizontal, or scattered across the architecture.**

### Rationale

Implementing only core assets for one or two sub-domains gives early feedback to all stakeholders whether the organization and process changes for PLE were the right ones without bearing too high risk for the whole organization. With such potential quick wins the organization is motivated to extend the product line also to other sub-domains.

### References and Known Uses

Car manufacturers typically divide their cars in sub-domains like chassis, engine, electrical system, or safety systems. This division is typically also mirrored in the organizational setup; similarly the configuration options for the customers are grouped in these sub-domains.

The features of the imaging platform in Siemens Healthcare are grouped into about 10 sub-domains that structure the problem as well as the solution space.

Klaus Schmid introduces a new approach for evaluating sub-domains, the domain potential assessment in [2].

## Start with Product Feature Map

### Context

An organization wants to benefit from reuse. The organization already implemented systems in the domain for a while. Existing products or solutions are available.

### Problem

To benefit from reuse in current and/or future product portfolio the organization needs to find out which artifacts should be implemented for reuse.

- Taking already available assets and turn them into reusable assets by increasing the quality and adding variation points is a low risk option to get a set of core assets. However it is unclear which variants should be supported.
- Development knows what it takes to build certain functionality and what should be reusable. But still this is at most a projection from the past product portfolio.
- Domain analysis looks at sub-domains that presumably have reuse potential, identifies the commonalities and variants in each sub-domain and gives a feeling for reuse potential. However, if all variants possible in a sub-domain are considered, this will not support the business of the organization optimally.

### Solution

Before the scoping of core assets starts, the products or solutions that shall be supported by these core assets have to be clear. Thus, the first step is to identify the products and market segments that shall be addressed by the product line. Next, the characteristics or features that are included in each of the products are listed to see the commonalities and variability between the products or market segments. This is the starting position for further commonality/variability analysis. The envisioned product portfolio restricts the list of possible features in each sub-domain. At the same time products get better shaped by deciding for each additional feature if it should or should not be in the product.

### Rationale

If the product portfolio or market segments of interest are not the starting point for core asset scoping, the assets will typically be more generic than necessary. The advantage of product line engineering over simple reuse as propagated in the 1990s is that only the variability elementary necessary to support a specific business is built into the core assets and not more – see *Favor Platform Simplicity*. Core assets have to be build considering variability that is to be expected for future portfolios, but the variants should be clearly identifiable and evidently motivated.

### References and Known Uses

Product/Feature maps are quite common in portfolio development. Car manufacturers like Audi and BMW present their portfolio in such lists.

Similarly, Siemens Industry Drive Technologies scope their product portfolios with the help of product/feature maps und use the customer visible part to present their product portfolio to customers, Figure 4 shows an example from <http://www.automation.siemens.com/>.

Explosion-protected Motors – Technical Data

Motors	Increased safety – "e"	Increased safety – "d"	Increased safety – "n"	Dust explosion protection	NEMA EX
<b>Voltage and power ranges</b>	0.12 - 165 kW	0.25 - 950 kW	0.09 - 1000 kW	0.06 - 1000 kW	1-300HP
	All commonly used voltages				230, 460 & 575 V with 60 Hz
<b>Frame sizes and designs</b>	63M - 315L	71M - 450	63M - 450	56M - 450	140 - 440
	All common construction types				
<b>Rated speed and rated torque</b>	750 - 3600 min <sup>-1</sup>	750 - 3600 min <sup>-1</sup>	750 - 3600 min <sup>-1</sup>	750 - 3600 min <sup>-1</sup>	900-3600 min <sup>-1</sup>
	0.3 - 10300 Nm	0.3 - 10300 Nm	0.3 - 10300 Nm	0.3 - 10300 Nm	1.5-1772 lb-ft
<b>Application area</b>	Ex-Zone 1 II 2G Ex e II T1-T3	Ex-Zone 1 II 2G Ex de IIC T1-T6	Ex-Zone 2 II 3G Ex nA II T3	Ex-Zone 21/22 Zone 21: II 2D Ex tD A21 IP65 T120°C Zone 22: II 3D Ex tD A22 IP55 T120°C	Class I, Group D, Class II, Groups F&G Division 1 hazardous areas

**Figure 4: Example of Siemens Automation and Drives Portfolio**

Klaus Schmid suggests this procedure in [2]. In this context it was tested with two product lines of MARKET MAKER Software AG and with one product line in Bosch.

In [3] Paul Clements and Linda Northrop suggest “Developing an attribute/product matrix”, which is one pattern in their collection of PLE patterns. The patterns in [3] were mined from experience with consulting companies that introduced or improved a PLE approach.

## Collaboration between Problem and Solution Experts

### Context

An organization decides to build a core asset base. It tries to optimize the return on investment.

### Problem

Neither product management, nor development alone is supposed to know all details around commonality and variability.

- Product management knows all facts around market, marketable features, value of features, and products. Further, it knows which features are common between products on a high level of abstraction. However, with more detailed analysis variability quickly increases, especially if non-functional requirements vary. Such variability can not be spotted on a high level of abstraction. Hence, the complexity and cost for features cannot be determined to the full extent by product management.
- Development knows the cost for developing assets, both for product-specific and core assets. Development has the 'solution space' knowledge for determining the actual variability and dependencies between variation points. However, development lacks the knowledge about the value of features for customers.

### Solution

Let scoping be lead by product management, but make it an iterative and collaborative effort between product management and development. Product management initially sets up the product map with feature assignments to products. Development analyses the feature in depth by drafting designs and identifying technical dependencies between variants. It might for example be necessary to include certain features into the core asset base that would not have been selected, if it were only based on their business value for the customer, but because they are necessary to develop higher prioritized features efficiently. The additional dependencies and cost are fed back to product management that re-estimates the cost/benefit ratio for features and products. Both parties iterate over the product portfolio commonly and alternately.

### Rationale

While product management is responsible for setting up a product portfolio that optimizes the business of an organization, development is responsible for minimizing the cost of a product portfolio, potentially optimizing reuse and to decide whether a feature set can be implemented in a consistent platform architecture at all. Neither can product management estimate efforts reliably, be it for product-specific or core assets, nor can development decide on priorities of products or features. The feedback from development often influences the product portfolio directly. Product management is made aware of existing knowledge or assets that can be marketed by changing the portfolio.

### References and Known Uses

For deciding which features go into the imaging platform at Siemens Healthcare, data from product management is used to calculate the value of a feature and from architects to get the cost of features.

The Siemens postal automation group and part of the hot rolling mill automation have product line teams installed that consist of product management (or sales) and architects. Together they decide whether a feature is developed in the platform or only for one specific solution.

In [3] the risks of scoping include “scope includes the wrong products”. Further, the authors describe that the reason typically is the missing collaboration between product management and development.

## General Patterns

### Regular Platform Releases

#### Context

Only a limited number of knowledgeable people exist per core asset that can change or extend a specific core asset. Core assets face high demand of change and extension by the products that are reusing them. How do you mitigate the pressure?

#### Problem

Accepting the high pressure, specifically regarding time, can result in several shortsighted solutions.

- Several branches of the same code parts for different products in parallel. The consequence is high merge efforts between the branches and overhead of switching between the branches for individual developers.
- High flexibility of the code parts to accept any variant, e.g. overly 'strategized' design or creating a #ifdef hell.
- In urgent cases application developers are inclined to fall back to copy & paste reuse, or even re-write the functionality product specific.

#### Solution

Therefore, prioritize your total feature backlog considering when reusable components have to be available for which products and provide high quality releases regularly, e.g. every 3 months. If a platform user urgently needs any change or extension, the feature backlog can be reprioritized accordingly.

#### Rationale

Regular releases alleviate the high pressure that typically leads to multiple branches; hence branches can typically be avoided or at least minimized in their number. In order to be accepted by the platform users the quality of the regular releases must be sufficiently high. Fully automated regression test suites that are executed on every change, up to the extent of Continuous Integration, secure the quality while minimizing extra efforts.

However, providing regular releases challenges the organization to define a proper versioning strategy that regulates which releases are compatible, e.g. all minor releases, and which releases contain incompatible API changes, e.g. major releases.

## Favor Platform Simplicity

### Context

Source code of reusable assets gets complex very easy. Often it is the well-intended flexibility designed and implemented by developers that increases the complexity.

### Problem

Platforms are intended for longevity, but the high complexity lets them die the too-complex-and-not-maintainable-any-more dead faster than expected. How do build the simplest possible platform?

- Platform development is often defined as providing the software with enough variability mechanisms, such as heavily applying the Strategy pattern and allowing for declarative programming via configuration files.
- Hard wiring only the very current requirements might lead to monolithic design, hence badly maintainable and evolvable software, too.

### Solution

Therefore, establish the mindset of the beauty of simplicity. Favor simplicity instead of flexibility. Firstly, differentiate between the inherent complexity of a problem and the accidental complexity. At all times be aware of the inherent complexity of a problem.

For example, the demanded variability of an application domain is inherent complexity, while the solution approach, e.g. flexibility, dynamicity, or genericity are typically sources of accidental complexity. In many regards this also means that the use of the classical GoF design patterns has to be limited to the bare minimum. Basically it is “Do the simplest thing that could possibly work” [9], while not obstructing future potential extension.

Reducing the inherent complexity is only possible by reducing the required variability, which in turn means reducing the scope of the product line. Accidental complexity can be avoided by demanding that all flexibility in the platform has to be justified by variability in the chosen scope of the problem space.

### Rationale

While the solution might sound trivial, it is not. With all the intrinsic motivation and creativity of a typical software developer, only lots of discipline and experience will allow him to actually focus on the essentials. Simplicity should guide itself on the actually demanded variability, the scope, of the product line, not the general interest in technology, preparing for all potential variations or the application of the most patterns. So in some sense this pattern is meant as reminder of adhering to best practice and pragmatism. In hardware development creating flexibility is much more expensive compared to software, hence overly-flexible design in hardware is less of an issue.

The role of the software architect, often also called technical lead, is crucial in this aspect. He guides the organization by establishing the right mindset. Continuous review of design decisions by development teams allows him give input and potentially correct misguided design decisions.

## References and Known Uses

At the imaging platform development effort in Siemens Healthcare this pattern is applied to ensure the longevity of the platform as well as the development efficiency of the applications on top. The group introduced the tracking and tracing of dependencies between solution and problem space variability. Due to the regulatory requirements, which already require some extent of tracing, this step was not too difficult. The additional tracing causes some overhead in maintaining, but on the positive side forces developers to design for simplicity.



## Balance Constraints

### Context

Strategic and operational decisions need to be made. Every project follows the constraints of the project management triangle [4]. Mapped to the area of software development they are defined as:

- Scope – the functionality especially the unique selling points offered by a product
- Quality – the developmental as well as operational qualities, both determining the total-cost-of-ownership
- Time – the duration it takes to develop the product, hence the time-to-market
- Cost – the cost of personnel, mostly wages, and costs for the infrastructure, like PCs, servers, building. Typically the largest part of the cost is directly related to the development time and involved staff.

### Problem

Individual roles in a software development organization typically span not all four categories, hence decisions can easily be concentrated on scope, quality, time, or cost only. The classical separation is as follows.

- Project management typically feels most responsible for the triple constraint time, cost, and developmental quality, such as regulatory compliance and availability of all documentation artifacts.
- Product management typically feels most responsible for scope, time, and customer visible quality, like performance and usability.
- Technical leadership (architects) typically feels most responsible for scope and developmental as well as operational quality.

Letting only one or two of the roles decide on a strategy leads to potentially unbalanced decisions.

### Solution

With every decision make sure you have a balance between the four constraints through the involved roles. A typical steering committee should consist of representatives from all streams of leadership (project management, product management, technical leadership). Only this way you can make sure that sound decisions with proper risk assessment are being made.

### Rationale

The partitioning between project management, product management, and technical leadership is the classical separation. Agile methodology approaches provide a slightly different partitioning, e.g. Scrum product owners are typically broader set up and care about time, scope, and quality. The Toyota approach [10] would even go so far to integrate all constraints into a single person: the chief engineer. Of course in this case there is no need anymore to balance the constraints, besides making sure you pick the 'right' chief engineer for your business success.

### **References and Known Uses**

Many groups in Siemens, among them the postal automation group and the hot rolling mill automation group, intensively apply this pattern to their advantage. They are well aware of the different forces of a project and use the respective perspectives to the advantage of creating innovative and excellent products.

## Acknowledgements

We thank our shepherd Jason Yip for his elaborate comments and his patience. Special thanks also to our Writer's Workshop group at EuroPloP 2009: Rene Bredlau , Eduardo Fernandez, Claudius Link, Klaus Marquart, Dietmar Schütz, Markus Völter, and Alain-G. Vouffo Feudjio. In this paper we base on the foundations laid by authors like Klaus Schmid, Linda Northrop, and Paul Clements. Our learning continued out of their product line engineering experiences.

## Terminology

**Core asset** – A set of reusable artifacts, i.e. requirements, domain models, architecture, patterns, concepts, documentation, test cases, budget plans, work plans, process, tools, workflows and code assets.

**Platform** – The sum of all core assets is typically called the product line platform. However, the term platform is sometimes also used for a base of technologies on which other technologies or processes are built [1], e.g. an operating system, middleware or component container. In this paper we refer to the first interpretation.

**Mission, Vision, Strategy** – Mission = why do I exist? Vision = where would I like to be? Strategy = how would I like to get there?

**Total-cost-of-ownership (TCO)** – The sum of all costs related to developing, maintaining, supporting, installing, servicing, and owning a certain product of the providing as well as receiving party.

**Time-to-market (TTM)** – The time it takes to deliver a product from initial conception to actual availability on the market to be bought by customers. The longer the delay the higher the chances for the competition bring their products into the market and endanger the potential market share for the own product.

**Project management triangle** – The triangle describes the constraining relationship between scope, time, and cost assuming the quality to be fixed. All are interrelated; you cannot change any of them without adopting the others [4].



**Scoping** – Scoping [1][3][5] is an activity to find the boundaries of a product line that can be divided into three sub-steps [6][7]: product line scoping, scoping the domain, and scoping the reusable assets (platform). The objective of product line scoping comprises defining the set of products and the main features to be included in the product line. Domain scoping represents the process of identifying appropriate boundaries for the sub-domains of the overall product line. Scoping the assets results in the decision which of the required functionality a development organization should implement as reusable base assets and which it should consider product specific. For simplicity we can stick to the

following definition: Scoping means setting the focus of reuse on the functionality that promises an optimal return on investment [8].

**Application and Domain Engineering** – Domain Engineering (DE) is the sub-process of PLE that is responsible for building and managing reusable base assets, while Application Engineering (AE) is about building products or solutions from these base assets. Variation in organization forms – PLE organizations may be split in a Domain Engineering part that is responsible for implementing the core assets and several Application Engineering parts that build products/solutions based the core assets and adding product specific features.

Alternatively, PLE organizations may as well be divided along sub-domains that do both, development of core assets and product/solution development. This setup is common in mature PLE organizations, mostly for solution development. Such a division is very efficient since no handover between the two disciplines is necessary. How to set up the best organization for a product line would be a pattern language on its own.

**Proactive evolution** – Core assets are identified up front and developed for reuse.

**Reactive evolution** – Creation of core assets based on existing, previously product-specific assets.

## References

- [1] Pohl, Böckle, van der Linden, *Software Product Line Engineering*, Springer, 2005
- [2] Klaus Schmid, *Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines*, Dissertation, Fraunhofer IRB Verlag, 2003
- [3] Clements, P. & Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002
- [4] [http://en.wikipedia.org/wiki/Project\\_triangle](http://en.wikipedia.org/wiki/Project_triangle)
- [5] K. Czarnecki and U. W. Eisenecker, *Generative Programming. Methods, Tools, and Applications*. Amsterdam: Addison-Wesley Longman, 2000.
- [6] Jan Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*, Addison- Wesley Reading, 2000.
- [7] Klaus Schmid, Steffen Thiel, Jan Bosch, Susanne Johnsson, Michel Jaring, Bernhard Thomé, Siegfried Trosch, *Scoping*, EASPS consortium wide deliverable CWD1.2.4, 2001.
- [8] Klaus Schmid, *A Comprehensive product Line Scoping Approach and Its Validation*, In *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*, ACM Press, 2002, pp. 593-603
- [9] XP guidance “Do the simplest thing that could possibly work”
- [10] <http://www.poppendieck.com/leadership.htm>

Dietmar Schütz  
Siemens AG, Corporate Technology  
CT T DE IT1  
System Architecture & Platforms  
Otto-Hahn-Ring 6  
81739 München  
Germany  
eMail: dietmar.schuetz@siemens.com  
Phone: +49 (89) 636-57380  
Fax: +49 (89) 636-45450

## ***VARIABILITY REVERSE ENGINEERING***

Version 1.0, (Final Version for Printed Proceedings), EuroPLoP2009

---

---

### **Summary**

In the realm of Product Line Engineering (PLE), Variability Management is one of the key issues. The success of the whole product line approach relies on the correctness of the variability models. Unfortunately, before transiting to PLE, knowledge on the variability is not addressed explicitly, but embedded in many development artefacts. This pattern provides an approach to extract that hidden knowledge, and transform it into the required problem side commonality/variability model.

---

---

### **Context**

Product Line Engineering (PLE), Platform Development,  
Product Business, Solution Business

An established development organisation with several successful similar projects has identified the potential for a Product Line (PL) approach. It has defined a business strategy and market scope to be covered, and developed a coarse roadmap and internal business case for developing reusable assets.

### **Example**

Consider your company operates in solution business in the web applications domain, developing customer specific (software) applications. These applications typically share a common set of features and solutions. For every customer/application, a separate project is created, responsible to satisfy customer needs within the given budget.

In order to reduce their own development effort, the projects have applied a copy/paste approach on the code (and project) base, using the most similar project from the past as starting point for their own work.

---

Copyright retain by author(s). Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.

This ad-hoc reuse has sped up initial development of other applications, but reveals weaknesses in an increasing number of maintenance scenarios. Every bug that is found has a high probability to affect other projects too, but is difficult to be located and fixed in the different branches.

Now, your management wants to establish a “platform” (better: product line engineering) approach in order to benefit from reusing common parts during software development and the entire software lifecycle. In the context of the “Commonality-Variability-Analysis” (C/V-analysis) tasks during domain analysis, the projects from the past are revisited in order to extract a useful set of commonalities that shape the basis for planned reuse in the future. In addition, the variant parts should be identified for a proper platform scoping, defining the complete set of core assets that should be provided upfront.

## Problem

A lot of knowledge regarding commonality and variability that has been accumulated in the past, but is not explicitly documented. How can the undocumented knowledge be made accessible for future work, and contribute to a viable commonality/variability-model?

The following **forces** influence the solution:

- *Different kind of artefacts.*  
The exiting base of information from previous projects spans various types of artefacts (documents, specifications, tools, code). All of them might be a source for commonality and variability.
- *No explicit highlighting of variability.*  
From the perspective of a single project, variability does not exist, since the customer wants a *specific* system. Therefore, each project supports its own needs, but does not care about the differences to others. The copy/paste approach has helped to have a quick start, but couldn't keep the different projects to follow their own, isolated path, resulting in an overly wide code base.
- *Constraints and dependencies.*  
Typically, a variability model does not only contain the possible variation points and variants, but also the dependencies between them, such as conflicting variants.

## Solution

Extend the forward oriented variability modelling (feature based C/V-analysis) with backward oriented techniques (reverse engineering). To this end, analyse promising types of artefacts that have been created by previous development projects in order to extract candidates for problem side variability. Assess these candidates for there relevance to identify solution side variations points and variants. Map those back to (customer-visible) features in order to problem side variability model. With that model in place, assess the variability on both sides to derive technical constraints and dependencies as necessary part of the variability model.

## Structure

The intended outcome of the C/V analysis is a *commonality/variability-model*. It contains features that are common to all products, and those where the products differentiate from others, characterized by variation points (e.g. colour) and specific variants (e.g. blue, green).

In order to distinguish this C/V model from the developmental solutions, this model is typically denominated as the *problem side C/V model*. It is counterparted by the *solution side C/V model*, which relates to components, modules, or code fragments of the realization structure and implementation. Both parts of the C/V-model are connected by a *mapping structure* that links the variations points from both sides together, hence allowing deriving a concrete implementation based on a given set of features.

Input sources for the analysis are all kinds of *artefacts* that have been created and/or modified in relation with the definition and development of concrete products. Since similar kinds of artefacts exist for the different products, they build a *comparison base* for the analysis.

## Realization

Applying *VARIABILITY REVERSE ENGINEERING* incorporates at least the sequence of six steps described below.

### 1 *Decide on input sources.*

Based on the knowledge which are the key artefacts that your business and development operates on, establish a set of artefact types that probably will provide much to variability.

The starting point are definitely the artefacts available on the problem side:

- Marketing material containing product descriptions.  
These might even contain explicit variability information by means of (comparative) feature lists.
- User Manuals

On the solution side, typical promising artefact types are the “high level” specifications of the system:

- Requirements specifications
- Architecture specifications
- System test cases

Some others artefacts contain the variability information more directly, but maybe on a too fine level:

- Code  
(maybe even explicitly exposing variability, e.g. by means of conditional compilation directives)
- Configuration files (.ini-files, and similar)

Last but not least, the development environment provides structural information too,

- Configuration management structure (branches)
- Build/development (management) structures



## 2 *Compare to derive differences.*

For most kinds of artefacts, variants reveal themselves as differences between two documents.

The results are best if you do not compare two random documents to each other, but use the copied master and those derived from it, to limit the set of differences/changes to just one level. To this end, it might be useful to do some “project archaeology” and dig out these “based on” relationships. Unfortunately, they are not necessarily identical throughout the whole set of artefacts.

Comparing two files using text/line oriented tools like `diff` is nice for code, but there is also need for a semantic diff /compare. Sometimes this is provided by development tools. Otherwise, it might be helpful to export the artefacts to an xml representation and compare these files incorporating the hierarchical structure. But this approach needs careful observation, since for example the order of nodes in containers (not ordered lists) might lead to misinterpretation the results.

## 3 *Identify candidates.*

Walk through the different kinds of artefacts and the comparison findings, in order to identify common and variant parts. Different artefact types require different assessment techniques, related to their typical content and change scenarios that might have been applied during “copy/paste/modify” cycles.

Marketing documents:

Scan for keywords like “option”, “additional”, “alternative”, since they are explicitly indicating variability. The features listed there can (after verification) directly added to the problem side C/V model, and used as input for considering variability. If there are comparative specs for the different products, they already contain most of the variability information.

Requirement documents:

Due to the “contractual” character of requirement documents, often polishing of wording is necessary. In order to not mistake these as variation candidates, it is helpful to look into related solution side artefacts (architecture and design documents, hopefully linked to the requirements) to assess the variability potential of the discovered changes.

Configuration management system:

This source of information might provide explicit variability candidates, e.g. by means of branches. Another useful source is the code and its changes over time. The changes (when taken from configuration management) often indicate the reason for change by check-in comments.

## 4 *Select solution side variability.*

After having all the differences and changes available, it is necessary to filter out the irrelevant elements. While this might be obvious for smaller code changes, all bigger differences must be reflected according to their

relevance to stakeholders (customer, product manager, key developer) regarding variation scenarios. The remaining elements are arranged into a solution side variability model, typically oriented along the structure of the system and development organisation, which contains the common parts too.

5 *Map back to problem side variability.*

We need to separate customer relevant requirements and variability from corresponding design related elements. For design related issues, try to analyze the rationale behind it –it could be an undocumented requirement or a constraint

- use established RE analysis techniques to extract the reason behind
- try to reverse the refinement step by abstraction/generalization

Use high level features identified in the solution domain as a guide for further reverse engineering activities and as completeness check

Finally merge all identified requirements, variability and constraints with requirements on solution side and consolidate the models.

6 *Complete white spots.*

When looking at your problem side, you might recognize that the system you see does not match the perspective of the customer. Use scoping techniques to define the boundaries of your product line, and complete missing elements within your desired scope.

**Consequences** *VARIABILITY REVERSE ENGINEERING* provides the **benefits** depicted below:

- *Efficiency*  
By focussing on artefacts types with high potential first (explicit variability, on problem side), the process of extracting variability the generates the most important results with minimal effort early (80:20-rule).
- *The variability model reflects knowledge from past projects*  
Even if not obvious upfront, the implicit variability knowledge from the past projects is incorporated into the commonality/variability-model, making it much more realistic than a strictly top down approach.
- *Derivation support*  
The artefact types that have provided useful content to the backward analysis are also promising candidates for forward oriented tasks: they should be supported by the platform for the efficient derivation of concrete projects/products.

On the other hand, the pattern carries the following **liabilities**:

- *Reverse Engineering can be laborious and expensive*  
Especially the tasks of deriving and assessing the candidates for variability can eat up tremendous resources, due to their sheer number.
- *Risk of overdoing*  
The past does not necessarily reflect the future. Hence focusing too much on the derived information can be misleading.
- *Risk of missing dependency information*  
Possible dependencies between variants (e.g. conflicts) are typically not expressed in the development artefacts, and hence cannot be derived from them. Hence, the variability model needs to be proof-read, consistency-checked and possibly extended afterwards.

**Variants** In product business, there are even more sources: information gained from competitors can be subject to the variability reverse engineering too: products catalogues, (comparative) feature lists, reverse-engineered products. Although they do not necessarily fit to your solution side variability model (and therefore should be kept separately in the beginning), they valuably contribute to the problem side.

**Credits** Thanks to my shepherd Hans Wegener, for his patience and guidance in busy times. To my colleagues Horst Sauer, Anne Hoffmann, and Christa Schwanninger, for providing their thoughts and never getting out of discussion. And last but not least I thank the participant of the writers workshop at EuroPLoP 2009, namely Alain-Georges Vouffo-Feudjio, Christa Schwanninger, Claudius Link, Ed Fernandez, Klaus Marquardt, Markus Völter, Michael Kircher, and Rene Bredlau.

**References** [SPLE2006]  
Pohl, Böckle, van der Linden, Software Product Line Engineering, Springer, 2005

# Digital Signature with Hashing and XML Signature Patterns

Keiko Hashizume, Eduardo B. Fernandez, and Shihong Huang  
Dept. of Computer Science and Engineering,  
Florida Atlantic University  
Boca Raton, FL 33431, USA,  
ahashizu@fau.edu, ed@cse.fau.edu, shihong@cse.fau.edu

## Abstract

Data security has become one of the most important concerns for organizations. Information is a valuable asset and needs to be protected. One important countermeasure against attackers is the use of digital signatures. Digital signatures provides message authentication and may also provide message integrity. We present here two patterns: XML Signature and Digital Signature with Hashing patterns. The XML Signature pattern, a specialization of the Digital Signature with Hashing, is used to secure XML messages.

## 1. Introduction

Data security has become one of the most important concerns among us, especially for organizations that have valuable information. An important security risk is that information can be modified during its transmission by somebody trying to make us believe something to his convenience. How do we prove that a message came from a specific user? Digital signatures use public-key cryptography to provide message authentication by proving that a message was sent indeed from a specific sender [dig, Sta06]. The sender encrypts the message using his private key to sign it. In this case, the signature has at least the same length as the message. This works but it wastes bandwidth and time. Thus, we need to reduce the length to the message before signing it. This can be done producing a digest through hashing. When the receiver gets the signed message, he verifies the signature by decrypting it using the sender's public key, thus proving that the message was encrypted by the sender. Also, digital signatures can provide message integrity by verifying whether a message was modified during its transmission. Digital signatures can also protect the integrity and verify the origin of a digital document, e.g. a certificate, or of programs. Digital signatures provide also *non-repudiation*, the sender cannot deny having sent the message he signed. In several countries, including the U.S., digital signatures have legal validity.

An emerging use of web services that exchanges XML messages also can be target of attacks. Some security standards have been developed to apply mechanisms that reduce security risks, one of these is. XML Signature. This standard is a joint effort between the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF). XML Signature defines how to digitally sign an entire XML message, part of an XML message, or an external object. XML Signature also includes hashing, but the pattern name follows the name of the standard. Because XML documents can have the same contents but in different layouts, we need to convert the documents into a canonical form before we apply digital signatures. Note that XML

Signature solves the same problem as the Digital Signature with Hashing pattern but in a more specialized context.

We present here two patterns: XML Signature and Digital Signature with Hashing patterns. The XML Signature pattern, a specialization of the Digital Signature with Hashing, is used to secure XML messages. We assume the reader is a designer intending to use message authentication in her design or a user intending to sign documents; we also assume both types of users have a basic knowledge of cryptography and UML. We provide a solution with sufficient detail so as it can be used as a guideline for design of signature systems and for users of signed documents.

Section 2 presents the Digital Signature with Hashing pattern, and Section 3 presents the XML Signature pattern.

## **2. Digital Signature with Hashing**

### **2.1. Intent**

Digital Signature with Hashing allows a principal to prove that a message was originated from it. It also provides message integrity by indicating whether a message was altered during transmission.

### **2.2. Example**

Alice in the Sales department wants to send a product order to Bob in the production department. The product order does not contain sensitive data such as credit card number, so it is not important to keep it secret. However, Bob wants to be certain that the message was created by Alice so he can charge the order to her account. Also, because this order includes the quantity of items to be produced, an unauthorized modification to the order will make Bob manufacture the wrong quantity of items. Eve is a disgruntled employee who can intercept the messages and may want to do this kind of modification to hurt the company.

### **2.3. Context**

People or systems often need to exchange documents or messages through insecure networks and need to prove their origin and integrity. Stored legal documents need to be kept without modification and indicating their origin. Software sent by a vendor through the Internet is required to prove its origin.

We assume that those exchanging documents have access to a public key system where a principal possesses a key pair: a private key that is secretly kept by the principal and a public key that is in a publicly-accessible repository. We assume that there is a mechanism for the generation of these key pairs and for the distribution of public keys; that is, a public key infrastructure (PKI).

### **2.4. Problem**

In many applications we need to verify the origin of a message (message authentication). Since an impostor may assume the identity of a principal, how do we verify that a message came from a particular principal? Also, messages that travel through insecure channels can be captured and modified by attackers. How do we know that the message/document that we are receiving has not been modified?

The solution for these problems is affected by the following **forces**:

- For legal or business reasons we need to be able to verify who sent a particular message. Otherwise, we may not be sure of its origin and the sender may deny having sent it (repudiation).
- Messages may be altered during transmission, so we need to verify that the data is in its original form when it reaches its destination.
- The length of the signed message should not be significantly larger than the original message; otherwise we would waste time and bandwidth.
- Producing a signed message should not require a large computational power or take a long time.

## 2.5. Solution

Apply properties of public key cryptographic algorithms to messages in order to create a signature that will be unique for each sender [Sta06]. The message is first compressed (hashed) to a smaller size (digest), and then it is encrypted using the sender's private key. When the signed message arrives at its target, the receiver verifies the signature using the sender's public key to decrypt the message, if it produces a readable message, it could only have been sent by this sender. The receiver then generates the hashed digest of the received message and compares it to the received hashed digest: if it matches the message has not been altered.

This approach uses public key cryptography where one key is used for encryption and the other key for decryption. To produce a digital signature (SIG), we encrypt (E) the hash value of a message (H(M)) using the sender's private key (PrK):  $SIG = E_{PrK}(H(M))$

We recover the hash value of the message (H(M)) by applying decryption function D to the signature (SIG) using the sender's public key (PuK). If this produces a legible message, we can be confident that the sender created the message because she is the only one who has access to her private key. Finally, we calculate the hash value of the message as  $H(M) = D_{PuK}(SIG)$ . If this value is the same as the message digest obtained when the signature was decrypted, then we know that the message has not been modified.

It is clear that the sender and receiver should agree to use the same encryption and hashing algorithms.

### *Structure*

Figure 1 describes the class diagram for the Digital Signature Pattern.

A **Principal** may be a process, a user, or an organization that is responsible for sending or receiving messages. This **Principal** may have the roles of **Sender** or **Receiver**. A Sender may send a plain Message and/or a SignedMessage to a receiver.

The **KeyPair** entity contains two keys: public and private, that belong to a **Principal**. The public key is registered and accessed through a repository, and the private key is kept secret by its owner. In a Public Key system, one key is normally used for encryption, while the other is used for decryption. **PublicKeyRepository** is a repository that contains public keys that can be available to anyone. The PublicKeyRepository may be located in the same local network as the principal or in an external network.

The **Signer** creates the **SignedMessage** that includes the **Signature** for a specific message. On the other side, the **Verifier** checks that the **Signature** within the SignedMessage corresponds to that message. The Signer and Verifier use the **DigestAlgorithm** and **SignatureAlgorithm** to create and verify a signature respectively. The DigestAlgorithm is a hash function that condenses a message to a fixed length called a *hash value* or message digest. The SignatureAlgorithm encrypts and decrypts messages using public/private key pairs.

### ***Dynamics***

We describe the dynamic aspects of the Digital Signature Pattern using sequence diagrams for the use cases sign a message and verify a signature.

*Sign a message* (Figure 2):

Summary: A Sender wants to sign a message before sending it

Actors: A Sender

Precondition: A Sender has a public/private pair key

Description:

- a) A Sender sends the message and its private key to the signer.
- b) The Signer calculates the hash value of the message (digest) and returns it to the Signer.
- c) The Signer encrypts the hash value using the sender's private key with the Signature Algorithm. The output of this calculation is the digital signature value.
- d) The Signer creates the Signature object that contains the digital signature value.
- e) The Signer creates the SignedMessage that contains the original message and the Signature.

Postcondition: A SignedMessage object has been created.

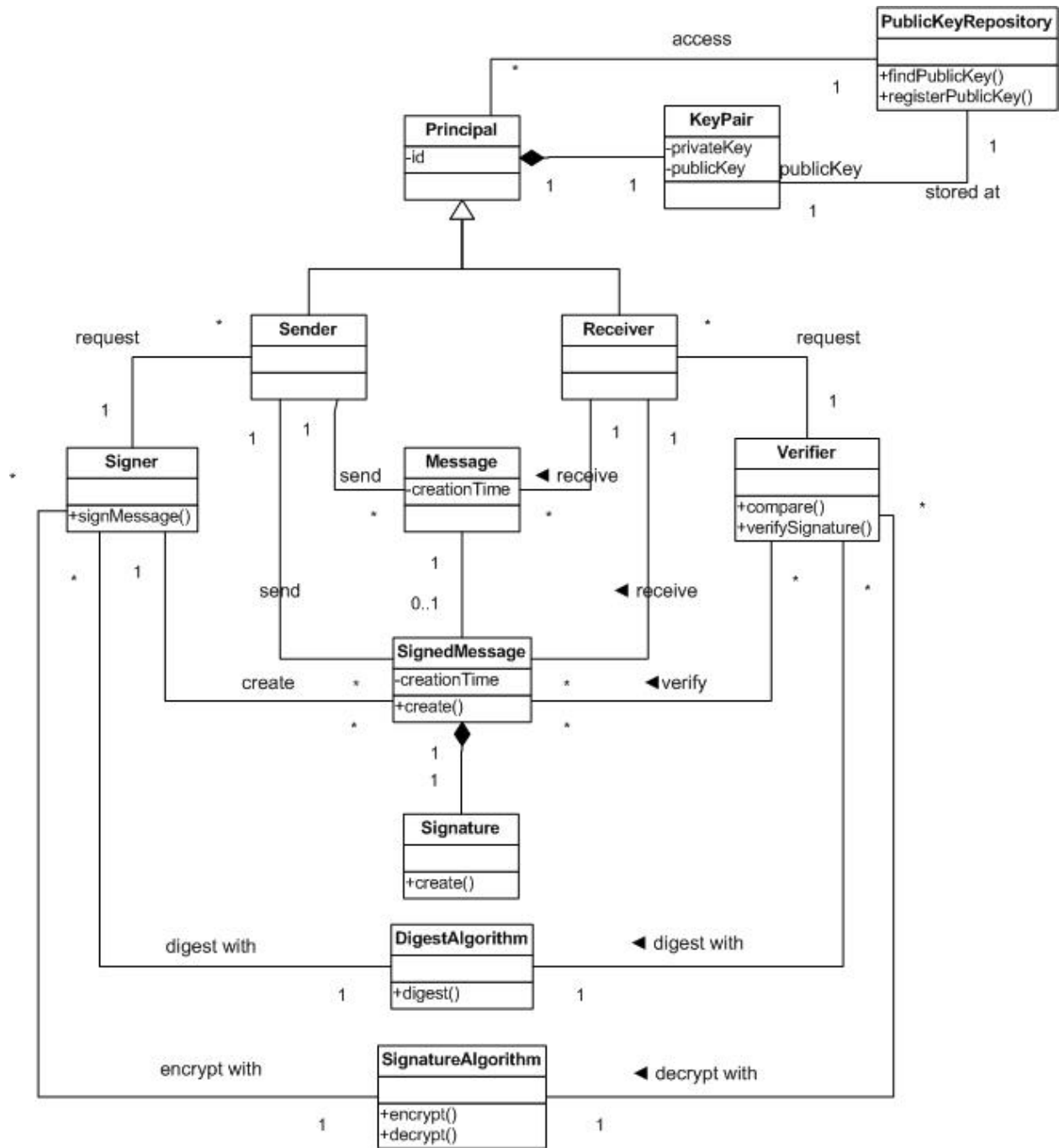


Figure 1: Class Diagram for Digital Signature Pattern



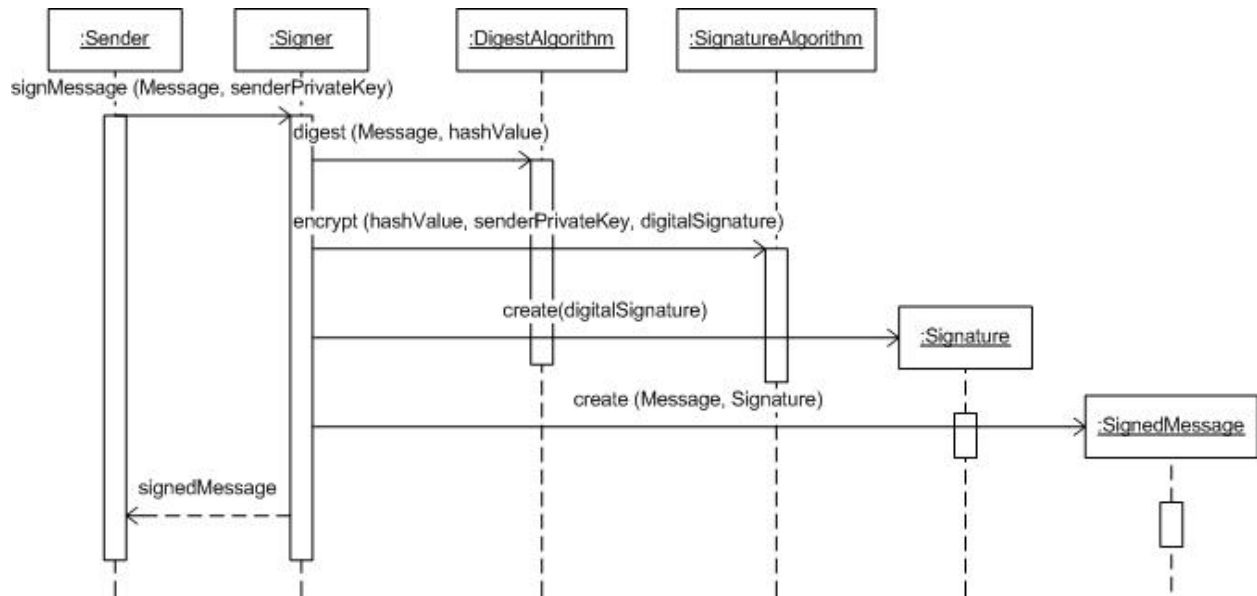


Figure 2: Sequence Diagram for signing a message

*Verify a Signature (Figure 3):*

Summary: A receiver wants to verify that the signature corresponds to the received message.

Actors: A Receiver

Precondition: None

Description:

- a) A Receiver retrieves the sender's public key from the repository.
- b) A Receiver sends the signed message and the sender's public key to the verifier.
- c) The verifier decrypts the signature using the sender's public key with the Signature Algorithm.
- d) The verifier calculates the digest value of the message.
- e) The verifier compares the outputs from step c) and d).
- f) The verifier sends an acknowledgement to the receiver that the signature is valid.

Alternate Flows:

- The outputs from step c) and d) are not the same. Then, the verifier sends an acknowledgement to the receiver that the signature failed.

Postcondition: The signature has been verified.

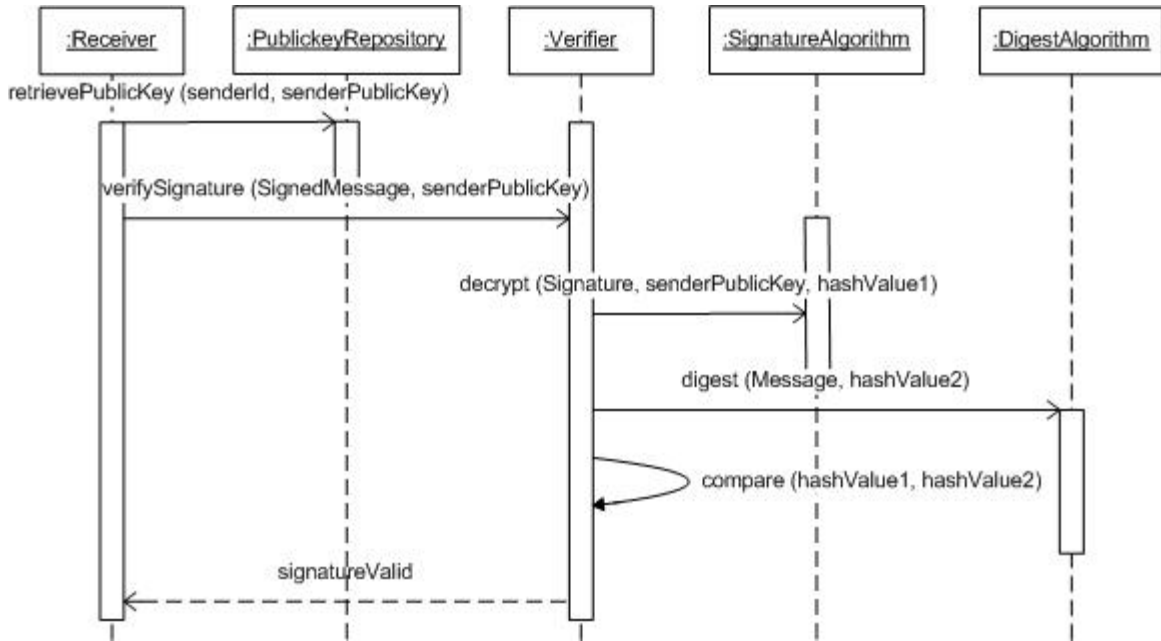


Figure 3: Sequence Diagram for verifying a signature

## 2.6. Implementation

- Use the Strategy Pattern [Gam94] to select different hashing and signature algorithms. The most widely used hashing algorithms are MD5 and SHA1. Those and others are discussed in [Sta06].
- A good hashing algorithm produces digests that are very unlikely produced by other meaningful messages, meaning that it is very hard for an attacker to create an altered message with the same hash value. The message digest should be encrypted after being signed to avoid man-in-the-middle attacks, where a person who captures a message could reconstruct its hash value.
- Two popular digital signature algorithms are RSA [RSA], and Digital Signature Algorithm (DSA) [Fed00, Sta06].
- The designer should choose strong and proven algorithms to prevent attackers from breaking them. The cryptographic protocol aspects, e.g. key generation, are as important as the algorithms used.
- The sender and receiver should have a way to agree on the hash and encryption algorithms used for a specific set of messages. XML documents indicate which algorithms they use and pre-agreements are not necessary.
- Access to the sender's public key should be available from a public directory or from certificates presented by the signer.
- Digital signatures can be implemented in different applications such as in email communication, distribution of documents over the Internet, or web services. For example, one can sign email's contents or any other document's content such as PDF. In both cases, the signature is appended to the email or document. When digital signatures are applied in web services, they are also embedded within XML messages. However,

these signatures are treated as XML elements, and they have additional features such as signing parts of a message or external resources which can be XML or any other data type.

- When certificates are used to provide the sender's public key, there must be a convenient way to verify that the certificate is still valid [SOA01].
- There should be a way to authenticate the signer software [dig]. An attacker who gains control of a user's computer could replace the signing software with his own software.

## 2.7. Known Uses

Digital Signatures have been widely used in different products.

- Adobe Reader and Acrobat [Ado05] have an extended security feature that allows users to digitally sign PDF documents.
- CoSign [Arx] digitally signs different types of documents, files, forms, and other electronic transactions.
- GnuPG [Gnu] digitally signs e-mail messages.
- The Java Cryptographic Architecture [Sun] includes APIs for digital signature.
- Microsoft .Net [Mic07] includes APIs for asymmetric cryptography such as digital signature.
- XML Signature [W3C08] is one of the foundation web services security standards that defines the structure and process of digital signatures in XML messages.

## 2.8. Consequences

This pattern presents the following advantages:

- Because a principal's private key is used to sign the message, the signature can be validated using its public key, which proves that the sender created and sent the message.
- When a signature is validated using a principal's public key, the sender cannot deny that he created and sent the message. If a message is signed using another private key that does not belong to the sender, the validity of the signature fails.
- If the proper precautions are followed (See 2.6), any change in the original message will produce a digest value that will be different (with a very high probability) from the value obtained after decrypting the signature using the sender's public key.
- A message is compressed into a fixed length string using the hash algorithm before it is signed. As a result, the process of signing is faster, and the signed message is much shorter.
- The available algorithms that can be used for digital signatures do not require very large amounts of computational power and do not take large amounts of time.

The pattern also has some (possible) liabilities:

- We need a well established Public Key Infrastructure that can provide reliable public keys. Certificates issued by some certification authority are the most common way to obtain this [Sta06].
- Both the sender and the receiver have to previously agree what signature and hashing algorithms they support. This is not necessary in XML documents because they are self-describing.
- Cryptographic algorithms create some overhead (time, memory, computational power), which can be reduced but not eliminated.
- The required storage and computational power may not be available, e.g. in mobile devices.
- Users must implement properly the signature protocol.
- There may be attacks against specific algorithms or implementations [dig]. These are difficult to use against careful implementations.
- This solution only allows one signer for the whole message. A variant or specialization, such as the XML Signature pattern, allows multiple signers.
- Digital signatures do not provide message authentication and replay attacks are possible [SOA01]. Nonces or time stamps could prevent this type of attacks.

## 2.9. Example Resolved

Alice and Bob agree on the use of a digital signature algorithm, and Bob has access to Alice's public key. Alice can then send a signed message to Bob. When the message is received by Bob, he verifies whether the signature is valid using Alice's public key and the agreed signature algorithm. If the signature is valid, Bob can be confident that the message was created by Alice. If the hash value is correct Bob also knows that Eve has not been able to modify the message.

## 2.10. Related Patterns

- Encryption/Decryption using public key cryptography [Bra00]
- Generation and Distribution of public keys [Leh02]
- Certificates [Mor06] are issued by a Certificate Authority (CA) that digitally signs them using its private key. A certificate carries a user's public key and allows anyone who has access to the CA's public key to verify that the certificate was signed by the CA.
- Strategy Pattern [Gam94], defines how to separate the implementation of related algorithms from the selection of one of them.

## 3. XML Signature

### 3.1. Intent

XML Signature allows a principal to prove that a message was originated from it. It also provides message integrity by defining whether a message was altered during transmission. The XML Signature standard [W3C08] describes the syntax and the process of generating and

validating digital signatures for authenticating XML documents. XML Signature also provides message integrity. It requires canonicalization before hashing and signing.

### **3.2. Example**

Alice in the Sales department wants to send product orders to Bob in the production department. The product orders are XML documents and do not contain sensitive data such as credit card number, so it is not important to keep them secret. Each order must be signed by Alice's supervisor Susie to indicate approval. Bob wants to be certain that the message was created by Alice so he can charge the order to her account and also needs to know that the orders are approved. Because the orders include the quantity of items to be produced, an unauthorized modification to an order will make Bob manufacture the wrong quantity of items. Eve can intercept the messages and may want to do this kind of modification.

### **3.3. Context**

Users of web services send and receive SOAP messages through insecure networks such as the Internet and need to prove their origin and integrity. During their transmission these messages can be subject to a variety of attacks.

We assume that a principal possesses a key pair: a private key that is secretly kept by the principal and a public key that is in a publicly-accessible repository. We assume that there is a mechanism for the generation of these key pairs and for the distribution of public keys.

### **3.4. Problem**

In many applications we need to verify the origin of a message (message authentication). Since an impostor may assume the identity of a principal, how do we verify that a message came from a particular principal? Also, messages that travel through insecure channels can be captured and modified by attackers. How do we know that the message/document that we are receiving has not been modified?

- For legal or business reasons we need to be able to verify who sent a particular message. Otherwise, we may not be sure of its origin and the sender may deny having sent it (repudiation). We assume the sender has signed the message to prove she is its author.
- Messages may be altered during transmission, so we need to verify that the data is in its original form when it reaches its destination.
- The length of the signed message should not be significantly larger than the original message; otherwise we would waste time and bandwidth.
- Producing a signed message should not require a large computational power or take a long time.
- We need to express a digital signature in a standardized XML format, so interoperability can be ensured between applications.
- There may be situations where we want to ensure proper origin or integrity in specific parts of a message. For example, an XML message can travel through many

intermediaries that add or subtract information, so if we sign the entire message, the signature would have no meaning. Thus, we should be able to sign portions of a message.

### 3.5. Solution

Apply cryptographic algorithms to messages in order to create a signature that will be unique for each message. First, the data to be signed may need to be transformed before applying any digest algorithm. The series of XML elements (that includes other subelements) is canonicalized before applying a signature algorithm. Canonicalization is a type of transform algorithm that converts data into a standard format, to remove differences due to layout formatting. This process is required because XML is a flexible language where a document can be represented in different ways that are semantically equal. Thus, after calculating the canonical form, both the sender and the receiver will sign and verify the same XML data respectively. After applying a canonicalization algorithm, the result value is digested and then encrypted using the sender's private key. Finally, the signature, in XML form, is embedded in the message.

In the other side, the receiver verifies the signature appended in the signed message. The verification process has two parts: reference verification and signature verification. In the reference verification, the verifier recalculates the digest value of the original data. This value is compared with the digest value included in the signature. If there is any mismatch, the verification fails. In the signature verification, the verifier calculates the canonical form of the signed XML element, and then applies the digest algorithm. This digest value is compared against the decrypted value of the signature. The decryption is done using the sender's public key.

There are three types of XML Signature: enveloped, enveloping and detached signature. In an enveloped signature, the signature is a child element of the signed data. For example, when you sign the entire XML message, the signature is embedded within the message. An enveloping signature is a signature where the signed data is a child of the signature. You can sign elements of a signature such as the Object or KeyInfo element. A detached signature is calculated over external network resources or over elements within the message. In the latter case, the signature is neither an enveloped nor an enveloping signature.

#### *Structure*

Figure 4 describes the structure of the XML Signature Pattern. Note that the upper part of this figure is almost the same as Figure 1. The main difference is that Figure 4 adds more details about the structure of the elements of the message so that signature can be applied more finely.

A **Principal** may be a process, a system, a user, or an organization that sends and receives **XMLMessages** and/or **SignedXmlMessages**. This principal may have the roles of **Sender** and **Receiver**.

Both an **XMLMessage** and a **SignedXMLMessage** are composed by XML elements, but this is only shown in the SignedXMLMessage. Each **XMLElement** may be a **SingleElement** that does not have any children or be a **Composite** element which is composed by other XML elements.

The **XMLSigner** and the **XMLVerifier** create and verify a **Signature**, respectively. They can select signature and digest algorithms. A **Signature** element is an XML element that has two required children: **SignedInfo** and **SignatureValue** and two optional children: **KeyInfo** and **Object**.

The **SignedInfo** element is the one that is actually signed. It contains one or more **Reference** elements, the canonicalization algorithm identifier, and the signature algorithm identifier. The Canonicalization algorithm is used to convert the SignedInfo element into a standard form before it is signed or verified. The Signature algorithm includes also a digest algorithm that is applied after calculating the canonical form of the Signed Info in both process creation and verification of XML signatures.

Each **Reference** element includes a Uniform Resource Identifier (URI), a hash value (DigestValue), the digest algorithm identifier (DigestMethod), and an optional list of **Transform** elements. The URI is a pointer that identifies the data to be signed. It can point to an element inside an XML message, an element inside the Signature element such as Object or KeyInfo, or resources located in the Internet. The DigestValue contains a hash value after applying the digest algorithm to the data pointed by its URI. If the Transform element exists, it includes an ordered list of transform algorithms that are applied to the data before being digested.

The **SignatureValue** element includes the value of the digital signature.

If the **KeyInfo** is present, it indicates the information about the sender's public key that will be used to verify the signature. This flexible element may contain certificates, key names, and other public keys forms. Additional information about this element can be found in [W3C08].

The optional **Object** element may contain **SignatureProperties** and/or a **Manifest**. The **SignatureProperty** identifies properties of the signature itself such as the date/time when the signature was created. The **Manifest** element includes one or more Reference elements same as the **Reference** element within the SignedInfo. They are semantically equal; however, each Reference in the SignedInfo has to be validated in order to consider a valid signature. On the other hand, the list of Reference elements within the Manifest is validated.

The sender and receiver must use the same hash, signature, and canonicalization algorithms. XML documents are self-descriptive and indicate this information so the sender only needs to find the corresponding algorithms.





## *Dynamics*

We describe the dynamic aspects of the XML Signature Pattern using sequence diagrams for the use cases sign different XML elements of an XML message and verify an XML signature with multiple references.

*Sign an XML message (Figure 5):*

Summary: A sender wants to sign specified XML elements of an XML message.

Actors: A sender

Precondition: A sender has a private/public key pair.

Description:

- g) A sender requests the signer to sign different XML elements of a message.
- h) The signer calculates the digest value over the XML element.
- i) The signer creates the <Reference> element including the digest value and using the digest algorithm.
- j) Repeat steps b) and c) for each XML element to be signed.
- k) The signer creates the <SignedInfo> that includes the Reference elements, the canonicalization algorithm identifier, and the signature algorithm identifier.
- l) The signer applies the canonicalization algorithm to the <SignedInfo> element.
- m) The signer signs the output from step f). First, it applies the digest algorithm, and then it encrypts the digest using the sender's public key. The output is the signature value.
- n) The signer creates the <SignatureValue> element that includes the signature value.
- o) The signer created the <KeyInfo> element that holds the sender's public key that will be used to verify the signature.
- p) The signer creates the <Signature> element that includes the <SignedInfo>, the <SignatureValue>, and the <KeyInfo> elements.
- q) The signer creates the SignedXMLMessage that includes the Signature and the XMLMessage.

Alternate Flows: None

Postcondition: The specified elements of the document have been signed

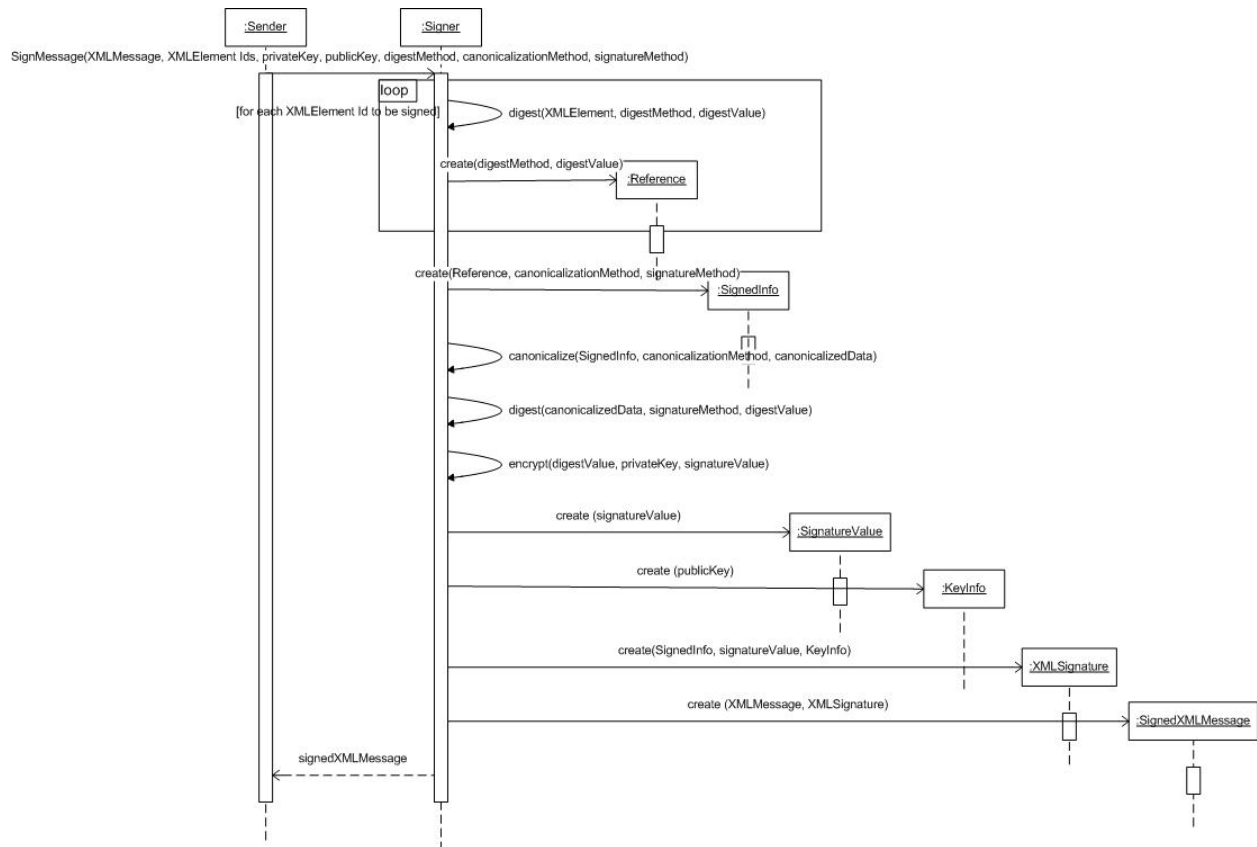


Figure 5: Sequence Diagram for signing an XML message

Verify an XML signature with multiple references (Figure 6):

**Summary:** A receiver wants to verify the signature of a received document.

**Actors:** A Receiver

**Precondition:** None

**Description:**

- A receiver requests to verify the signature that is included in the SignedXMLMessage.
- The verifier obtains the signature elements such as the <SignedInfo> which includes the <Reference> elements, the <SignatureValue>, and the <KeyInfo> elements.
- The verifier calculates the digest value over the XML element that is pointed (URI) in the <Reference> element using the digest algorithm specified in the <Reference> element as well.
- The verifier compares the output from step c) against the digest value specified in the Reference element.
- Repeat step c) and d) for each <Reference> included in the <SignedInfo> element.
- The verifier canonicalizes the <SignedInfo> element using the canonicalization method specified in the <SignedInfo>.
- The verifier digests the output from step f) using the digest algorithm specified in the Signature Algorithm.
- The verifier decrypts the signature value using the sender's public key (<KeyInfo>).
- The verifier compares the outputs from step f) and h).

j) The verifier sends an acknowledgement to the receiver that the signature is valid.

Alternate Flows:

- If the values compared in step d) are not the same, then the signature is invalid.
- If the outputs in the step i) are not the same, then the validation fails.

Postcondition: The signature is validated.

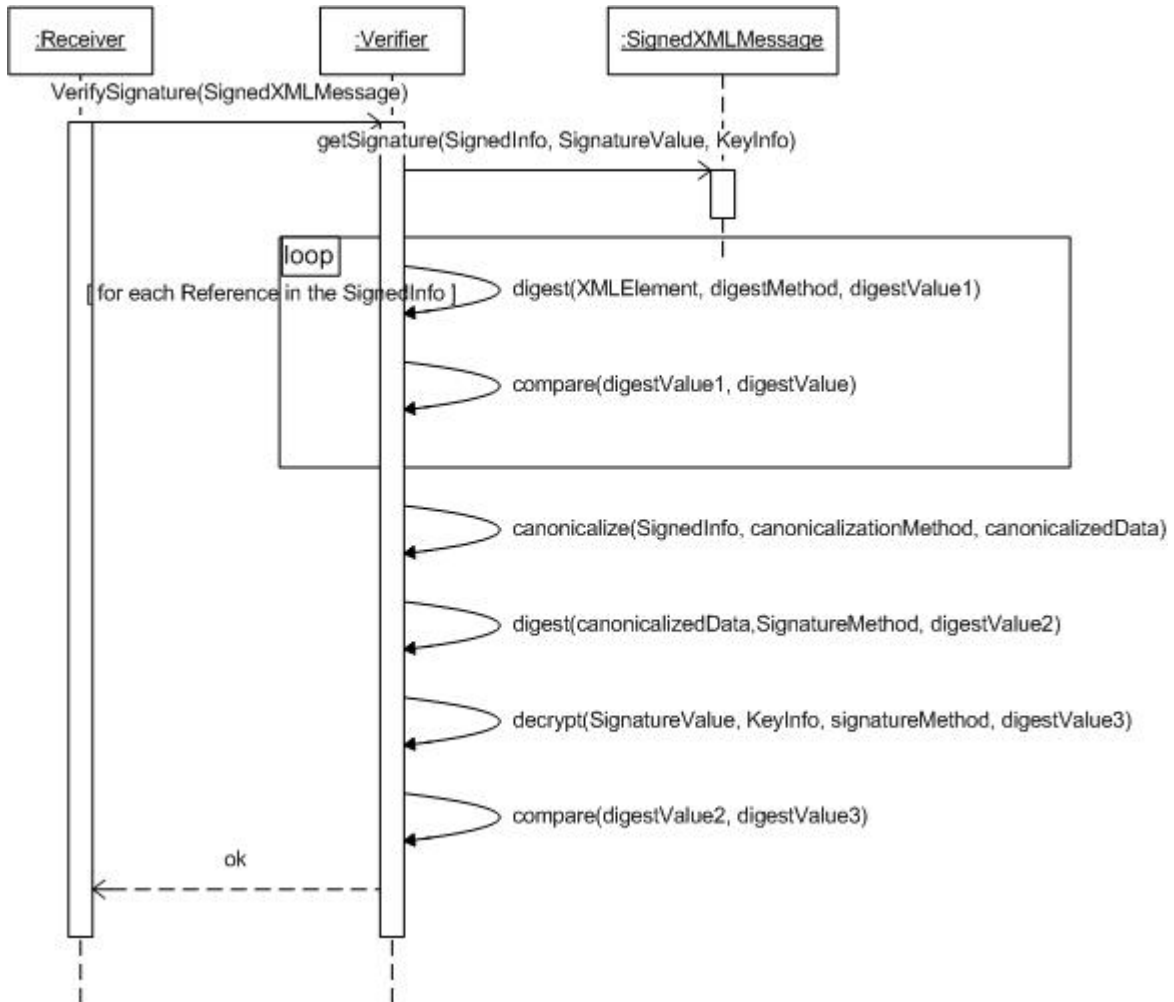


Figure 6: Sequence Diagram for verifying an XML signature

### 3.6. Implementation

- Identifiers of algorithms used to create a signature are attached along with the signature, so they also should be protected from being modified by attackers.
- XML documents may be parsed by different processors, and also XML allows some flexibility without changing the semantic of the message. Thus, we need to convert the data to be signed to a standard format.
- All the signers of a given document should have the same level of trust to avoid misleading the receivers about the trust of the whole message. Allowing untrusted signers might give them a better chance to attack the message.

- Use the Strategy Pattern [Gam94] to select different hashing and signature algorithms. The most widely used hashing algorithms are MD5 and SHA1. Two popular digital signature algorithms are RSA [RSA] and Digital Signature Algorithm (DSA) [Fed00].
- If needed the data to be signed needs to be transformed using transformation algorithms before producing a digest. For instance, if the object to be signed is an image, it needs to be converted into text.
- It is recommendable the use of certificates issued by an Certification Authority that are trusted by the sender and the receiver.

### 3.7. Known Uses

Several vendors have developed tools that support XML Signature.

- IBM - DataPower XML Security Gateway XS40 [IBM] parses, filters, validates schema, decrypts, verifies signatures, signs, and encrypts XML message flows.
- Xtradyne – Xtradyne’s WS-DBC [Xtr]. The Web Services Domain Boundary Controller is an XML firewall that provides protection against malformed messages and malicious content, XML encryption, XML signature, and authentication, authorization, and audit.
- Forum Systems - Forum Sentry SOA Gateway [For] conforms to XML Digital Signature, XML Encryption, WS-Trust, WS-Policy and other standards.
- Microsoft .NET [Mic] includes API that support the creation and verification of XML digital signatures.
- Java XML Digital Signature API [Mul07] allows to generate and validate XML signatures

### 3.8. Consequences

This pattern presents the following advantages:

- A principal’s private key is used to sign the message. The signature is validated using its public key, which proves that the sender created and sent the message.
- When a signature is validated using a principal’s public key, the sender cannot deny that he created and sent the message. If a message is signed using another private key that does not belong to the sender, the validity of the signature fails.
- Any change in the original message will produce a digest value that will be different from the value obtained after decrypting the signature using the sender’s public key.
- Before applying any signature algorithm, the data is compressed to a short fixed-length string. In XML Signature, digest algorithms are used two times; one is used to digest data to be signed indirectly, and the other digest algorithm is used to digest the canonical form of the SignedInfo element.
- Any change in the data that was indirectly signed will produce another digest that will invalidate the signature.
- The available algorithms that can be used for digital signatures do not require very large amounts of computational power and do not take large amounts of time.
- We can sign different parts of a message with different signatures. This allows a set of principals to write portions of a document and sign them.

- An XML signature is an XML element that is embedded in the message. The XML signature is composed of several XML elements that include information such as the value of the signature, the key that will be used to verify the signature, and algorithms used to compute the signature. This standard format helps XML parsers to better understand signature elements during the validation process.
- This pattern supports also message authentication code (MAC). Both signatures and MACs are syntactically identical. The difference between them is that signatures use public key cryptography while MAC uses a shared common key.
- The data being signed is pointed by its URI (Uniform Resource Identifier), so elements within XML messages and external network resources can be located using their identifiers.
- The SignedInfo is the element that is actually signed. It includes the references that point the data being signed along with their digest values, and algorithms identifiers. Thus, the XML signature also protects the algorithm identifiers from modification.
- XML Signature uses canonicalization algorithms to ensure that different representations of XML are transformed into a standard format before applying any signature algorithm.
- XML documents are self-describing and the sender and receiver don't need to agree in advance on the algorithms to be used.

The pattern also has some (possible) liabilities:

- We need a well established Public Key Infrastructure that can provide reliable public keys. Certificates issued by some certification authority are the most common way to obtain this [Sta06]. There is a public key standard for XML that should be used.
- Users must implement properly the signature protocol.
- There may be attacks against specific algorithms or implementations [dig]. These are difficult to use against careful implementations.
- Signing and verifying XML messages may create a significant overhead.
- The pattern does not describe the complete standard. For example, details of transforms and key values have been left out for simplicity [W3C08].

### 3.9 Example resolved

Alice and Susie sign each product order sent to Bob. Bob has access to Alice's and Susie's public keys. When the message is received by Bob, he verifies whether the signatures are valid using Alice's and Susie's public keys and the signature algorithm specified in the order. If the signature are valid, Bob can be confident that the message was created by Alice and approved by Susie. If the hash value is correct Bob also knows that Eve has not been able to modify the message.

### 3.10 Related Patterns

- This pattern is a specialization of the Digital Signature with Hashing Pattern.
- WS-Security Pattern [Has09] is a standard for securing XML messages using XML signature, XML Encryption, and security tokens.

The following specifications are related to XML Signature, but they have not been expressed as patterns.

- The XML Key Management Specification (XKMS) [W3C01] specifies the distribution and registration of public keys, which works together with the XML Signature.
- WS-SecurityPolicy [OAS07] standard describes how to express security policies such as what algorithms are supported by a web service or what parts of an incoming message need to be signed or encrypted.

## 4 Conclusions

We presented two patterns: Digital Signature with Hashing and XML Signature, the latter a specialization of the first one for a more specific context. Since the XML pattern solves the same problem it repeats the general aspects of the Digital Signature pattern but repeating this information allows the XML pattern to be used alone. We showed these two patterns together to make clearer the logic behind XML Signature, a rather complex pattern. Future work will include completing our development of other web services security patterns such as XML Encryption and WS-Security [Has08].

## Acknowledgements

We thank our shepherd Amir Raveh, for his careful comments that improved the quality of this paper. The workshop participants at EuroPLoP 2009 also provided valuable comments. This work was supported by a grant from DISA, administered by Pragmatics, Inc. Our security research group provided useful comments.

## References

- [Ado] Adobe System Incorporated, Digital Signatures, <http://www.adobe.com/security/digsig.html>
- [Arx] Arx, Digital Signature Solution (Standard Electronic Signatures), <http://www.arx.com/products/cosign-digital-signatures.php>
- [Bra00] A. Braga, C. Rubira, and R. Dahab, “Tropyc: A pattern language for cryptographic object-oriented software”, Chapter 16 in *Pattern Languages of Program Design 4* (N. Harrison, B. Foote, and H. Rohnert, Eds.). Also in *Procs. of PLoP’98*, [http://jerry.cs.uiuc.edu/~plop/plop98/final\\_submissions/](http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/)
- [dig] Digital signature, [http://en.wikipedia.org/wiki/Digital\\_signature](http://en.wikipedia.org/wiki/Digital_signature)
- [Fed00] Federal Information Processing Standard, “Digital Signature Standard,” 27 January 2000, <http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf>
- [For] Forum Systems, Sentry: Messaging, Identity, and Security, <http://www.forumsys.com/products/soagateway.php>

- [Gam94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994
- [Gnu] GnuPG, The GNU Privacy Guard, <http://www.gnupg.org/>
- [Has09] K. Hashizume and E. B. Fernandez, "A Pattern for WS-Security", submitted for publication.
- [IBM] IBM, WebSphere DataPower XML Security Gateway XS40, <http://www-01.ibm.com/software/integration/datapower/xs40/>
- [Leh02] S. Lehtonen and J. Parssinen. "A Pattern Language for Key Management," EuroPlop 2002. <http://www.hillside.net/patterns/EuroPlop2002/papers.html>
- [Mor06] P. Morrison and E.B.Fernandez, "The Credential pattern", Procs. of the Conference on Pattern Languages of Programs, PLoP 2006, Portland, OR, October 2006, <http://hillside.net/plop/2006/>
- [Mic07] Microsoft Corporation, .NET Framework Class Library, November 2007, <http://msdn.microsoft.com/en-us/library/ms229335.aspx>
- [Mul07] S. Mullan, Programming with the Java XML Digital Signature API, Sun Microsystems March 2007, [http://java.sun.com/developer/technicalArticles/xml/dig\\_signature\\_api/](http://java.sun.com/developer/technicalArticles/xml/dig_signature_api/)
- [OAS06] OASIS, Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), 1 February 2006, <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
- [OAS07] OASIS, W-S SecurityPolicy 1.2, 1 July 2007, <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf>
- [RSA] RSA Security, PKCS #1: RSA Cryptography Standard, <http://www.rsa.com/rsalabs/node.asp?id=2125>
- [SOA01] W3C, SOAP Security extensions: Digital Signature, W3C NOTE 06, February 2001, <http://www.w3.org/TR/SOAP-dsig/>
- [Sta06] W. Stallings, *Cryptography and network security* (4<sup>th</sup> Ed.), Pearson Prentice Hall, 2006.
- [Sun] Sun Microsystems Inc., Java SE Security, <http://java.sun.com/javase/technologies/security/>

- [W3C01] W3C, XML Key Management Specification, March 2001  
<http://www.w3.org/TR/xkms/>
- [W3C08] W3C, XML Signature Syntax and Processing (Second Edition), 10 June 2008,  
<http://www.w3.org/TR/xmlsig-core>
- [Xtr] Xtradyne, Xtradyne's WS-DBC - the XML/SOAP Firewall for Enterprises,  
<http://www.xtradyne.de/products/ws-dbc/ws-dbc.htm>



# Dealing with Complexity

Klaus Marquardt  
Dorothea-Erxleben-Straße 78  
23562 Lübeck  
Germany  
<mailto:pattern@kmarquardt.de>  
<http://www.kmarquardt.de>

Copyright © 2009 by Klaus Marquardt. Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website

All but the most trivial software systems are complex, and complex systems have a high risk of failure.

Across all industries, large projects have a higher risk of failure than small projects. Their sheer size is a major contributing factor to their internal complexity; the infrastructure and communication becomes more complex. With many factors combined and interrelated, smaller disturbing effects get out of control; the project develops unexpected behavior and refuses to be manageable.

Within the project the complexity must be tackled to increase the chances for success. Engineering disciplines manage complexity, avoid, circumvent, and reduce it. A prominent aspect again is to limit the size of the system to take care of at once – decomposition and incremental development resemble each other in this respect. Our engineering and project management wisdom well in place, we are inherently optimistic that we will be able to succeed.

At the same time, complex systems are the natural friends of highly qualified engineers. They provide intellectual and cultural challenges, and they require experts to solve it. The latest project is typically more complex than the ones we have already completed. We need to cope with complexity, we know it and we actually love it [*Marquardt2008*]. Viewed from very far away, the continued overestimation of our abilities and this inherent optimism ultimately enables the successes of homo faber [*Frisch1957*] in the first place.

Success, just as complexity, is in the eyes of the beholder. This paper is aimed to assist project managers and key stakeholders inside and outside of the project, to cope with the complexity and control its contributing factors.

## Complexity

Dealing with large and complex systems is the dominant occupation of IT professionals. Many advances of the past decades have helped here, mostly in the Lampson style [*c2lampson*]: "all problems in Computer Science can be solved by another level of indirection", allowing to neglect most levels of detail for the moment.<sup>1</sup>

---

<sup>1</sup> In praxis, this is only partly successful. The separation of abstraction levels does not imply them being independent; details often influence the more abstract levels.

Complexity has been analyzed and treated extensively, including the founding of scientific branches. However, few insights have found their way into the working knowledge of software professionals, and are applied in daily routine.

- Complexity and complicatedness are different concepts: complicated systems or problems can be subdivided and solved in independent parts, where this is by definition impossible with complex problems.

Solving complicated problems thus benefits from a divide-and-conquer approach in some form – breaking tasks down, developing features sequentially or in parallel, separating technical or domain concerns. In contrast, solving complex problems requires an understanding of the entire mechanism and its internal and external influences.

This distinction is not entirely helpful in actual projects. Complexity is already added with human interaction; adding different aspects of complicatedness will create a problem that is not distinguishable from a complex problem. If you are fluent in different languages, you might want to check out different understandings to complexity in [*Wikipedia*].

- The separation of essential (intrinsic, problem domain related) complexity from accidental complexity, introduced by Brooks [*Brooks1995*], helps to distinguish complexity that stems from the approach of problem solving, i.e. that the project is essentially creating itself.

While the awareness created by this distinction is a major step, Brooks resolution proposals, like his concept of conceptual integrity, are only valuable if you know already what to do. They are hardly accessible to and applicable by the uninitiated.

Directly attacking complexity will change the situation, but the response often is that complexity comes back at another place. The following attempts to reduce complexity contribute their own complexity, at a different location depending on their individual mechanism:

- Raising the level of abstraction requires education and personal ability, which is expensive at best. In case of “analysis paralysis” or extreme toolsmithing (XT) it can break your project at worst.
- Scope reduction is a craft that can be taught, but it always has a political dimension to it: the renegotiation with the stake holders. Applying it to actual project situations is both an art and tedious work.
- Finally, striving for conceptual integrity creates a strong coupling between projects, teams and individuals. Such an intended coupling between many system components is a factor to complexity itself, and the necessary amount of work and friction may outweigh any potential benefits.

## Chosen Complexity

Complexity thrives from contributing factors, and the factors can be classified and considered individually. When the complexity contributions are reduced in amount or severity, humans are able to deal with the remaining complexity due to

their personal experience. Indirectly, the complexity itself becomes decomposed and resolved. Early complexity factor management is more promising, but it is never too late to take control of your project's influence factors.

A distinction between chosen complexity, and imposed complexity, helps to gain consciousness about the mechanisms that add complexity to the project, and to take control of and possibly eliminate contributing factors. The chosen complexity is typically a subset to Brooks' accidental complexity – but also intrinsic complexity could partly be chosen. Most importantly, the viewpoint and focus are different. Chosen complexity comprises any complexity factor a project leader and team has actively or passively accepted into the project.

Ultimately the project leader is responsible for the project's success. This includes managing and minimizing risks; if some stakeholder initially imposed some condition that increase the overall complexity and thus increase the risk, it is the project leader's duty to bring these conditions under project control and remove obstacles when possible. This attitude is closer to Beck's "play to win" [Beck1999] than to dutiful acceptance of the requirements document. The project shall strive to be successful, including negotiation of success criteria, or to fail quickly.

The complexity factor attribute chosen versus imposed is not stable. Many factors start out as imposed by stakeholders or other forces. However, once a project leader becomes aware that the current situation imposes an obstacle or significant risk, she needs to remove the respective factors, or negotiate about their importance and how success is defined for this particular project. By silently accepting or ignoring imposed factors that are known to contribute to complexity, the project leader abandons responsibility, taking an "absent without leave".

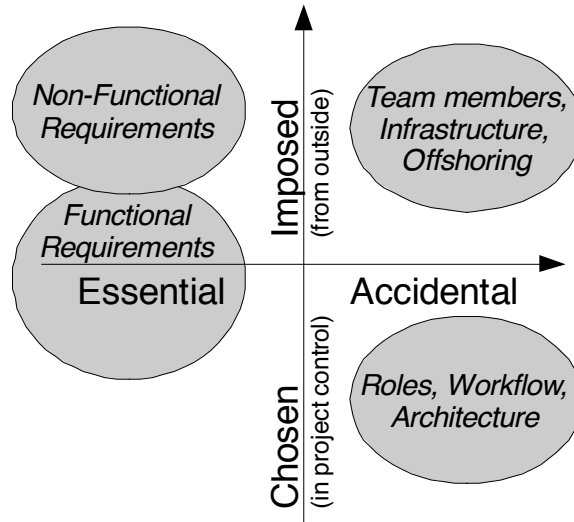
## Complexity Factor Classification

The key technique for coping with complexity is to identify and name the contributing factors, and to treat them in a way that reduces their impact on the project. This works against complexity in two ways: by the removal of the factors, and by creating consciousness about these factors and gaining insight and security.

CLASSIFY COMPLEXITY is the introductory pattern in this paper. It helps to list all the risk and size factors, and to classify them. This classification follows the two dimensions stated above, essential and accidental versus imposed and chosen. The diagram shows these dimensions, and typical complexity factors found in these coordinates:

- The team members and overall infrastructure are often established from the very beginning. They are initially imposed and accidental, i.e. they belong to the solution domain.
- Roles and workflow are typically chosen by the team and could be changed.
- Non-functional requirements are mostly implicitly stated, and typically not subject to discussion: imposed, and essential as they relate to the problem.

- Functional requirements belong to the problem domain. However, a fair amount of them is often not essential for the project’s success and should be interpreted as a part of the chosen complexity as it could be removed.



Once some factor is identified as chosen and preferably related to the solution approach, the project can control and address it.

The following patterns help to move the complexity factors into a lower quadrant and enable project control. The overview also lists the key techniques used:

- to approach and influence the project’s stakeholders;
- to shrink the size of (some aspect of) the project;
- to improve on the internal organization and communication;
- to gain competence to enhance the personal ability to cope with complexity.

Pattern	Key Mechanisms
RE-NEGOTIATE COMPLEXITY	influence stakeholders
DE-VISUALIZE STRATEGIC PROJECTS	influence stakeholders; shrink size
COMBINATORIAL BUDGET	shrink size; influence stakeholders
PIECEMEAL GROWTH	shrink size
DIVIDE AND CONQUER <sup>2</sup>	improve organization
DELEGATE COMPLEXITY	improve organization
MANAGEMENT BY TRIGGERS	improve organization
LOCAL DECISION COMMUNITIES	improve organization; gain competence
GATHER DOMAIN KNOWLEDGE	gain competence

<sup>2</sup> Divide and Conquer has many known publications and is not included in this paper.

## Development Methodology

Most development methods claim to successfully cover complexity management. While process models that closely follow CMMI address complexity by ensuring the organization's ability to minimize risk, agile approaches ensure that the organization is able to react to feedback.

With respect to complexity, both virtues are essential. Many of the mechanisms can be found in agile methods, while others resemble project management best practices. Your project likely benefits from both. Beware though: whenever some practice or approach does not help you to improve the situation, refrain from enforcing more of the same.

## Pattern Form

The pattern format can contribute to the quality of brevity.

In this paper, the pattern context is kept very broad and sketched within one line after the name. The problem statement is followed by the forces pulling in different directions and listed mostly in "...but..." sentences. The "therefore" keyword initiates the description of the solution. The solution includes general strategy as well as implementation details, and examples wherever these would not compromise the desired brevity.

## Classify Complexity

Applies to projects considered complex by key stakeholders.

It is unclear how complex the project actually is, and which measures can be taken to increase the project's success probability.

Knowing your complexity does not remove it,  
but not becoming aware of complexity will not yield effective measures.

Many factors contribute to a project's complexity,  
but reducing the number and severity of even a few factors increases the chances for success.

Some complexity factors appear imposed onto the project,  
but whether to accept them or to fight them is a choice of the project.

**Therefore**, create awareness of the project's complexity by listing all the risk and size factors that have an impact on your project's complexity. Afterwards, classify these factors along multiple dimensions, especially including a distinction whether a factor is essential to the problem or created by the solution approach, and whether it is imposed or chosen. Chosen factors are those that, according to the evaluation, could be changed by the project itself.

The complexity factors' classification is similar to a project risk assessment. It needs to be fairly complete so that you can communicate and discuss the influence factors. The classification also needs honesty, so it can give a prospect and a healthy signal to upper management: we are aware of our potential problems, and we try to gain control over them.

Depending on the company culture, the classification needs to avoid factors that are considered trivial – except when you think you need to address exactly these factors. Furthermore, each company has its taboos; just mentioning factors like “offshoring” or “lack of education” might set a political tone. If your company demands or favors some practice, you would only want to mention it if you are willing to make a strong case against it in your particular project.

Use the classification's visualization to discuss with the stakeholders and move factors into the chosen area. When the stakeholders are optimistic and would not value an increased chance for project success at the expense of allowing slips in some initial boundaries, your negotiation might be successful only after you have already missed some milestone.

## Re-Negotiate Complexity

Applies to projects whose stakeholders impose on the project's approach.

Accepting every wish that a stakeholder mentions limits the project's options to make decisions that fit its situations.

Stakeholders define the terms and conditions of the project,  
but terms that proscribe parts of the solution may limit the chances for success,  
and the project leader is responsible for the project's success.

At project initiation, all costs and effects are negotiated,  
but complexity factors that are identified late still affect you,  
and when new insights arise, a renegotiation might be needed.

**Therefore**, renegotiate all factors that contribute to the overall complexity, as soon as you become aware of them.

Stakeholders may prescribe project relevant topics like development team members or technology choices. Once you can with knowledge argue about chosen complexity, you can start making a case to change the project. Even with prominent risks, however, it might be that the negotiation just confirms the conditions you tried to overcome.

Approaching stakeholders and asking for a change in project settings is an inconvenient step. However, not raising issues turns them into your own – the project leader becomes responsible for all choices she did not challenge. You need a strategy to escalate in a way that keeps everybody's face, and you need to be successful with your first try.

For implementation, use MOTIVATIONAL QUESTIONS [*Marquardt2004*] to address all aspects that hit back when ignored. They need to answer the immediate questions for steering, address prioritization aspects, and ensure that the decision becomes secured against later opposition. An example set of questions contains these:

- What is the problem?
- What is the proposed solution?
- Who wants this?
- What does it cost?
- What happens if we don't do it?
- Does everybody agree?

## De-Visualize Strategic Projects

Applies to innovative projects with high visibility.

Projects initiated by top management and aimed at fulfilling high expectations, are suspiciously observed from all parties that might consider themselves affected. These projects likely suffer from stakeholder creep, followed by all other types of creep including complexity creep.

All potential parties desire to be involved in strategic projects,  
but a project involving all parties might never start at all.

Strategic projects will affect many different commodities and departments,  
but the final effects can merely be guessed.

Highly visible projects invite fans and critics alike, making non-political progress impossible,  
but hiding important projects will even be more counterproductive, once they go public.

Important projects are often assumed large and generously funded,  
but simple, small and properly funded projects have a higher probability to succeed.

**Therefore**, start the most visible projects as small as possible, and reduce their scope even further. Define them to answer very few questions, so that all parties that did not become sponsors or stakeholders see no need to interfere.

Small projects have fewer factors that contribute to complexity, and they have a higher potential to successfully cope with the remaining complexity since they need not spend effort due to their sheer size.

Strategic projects need to address numerous aspects of change. However, it is virtually impossible to get them addressed all at the same time, and likely some of the answers will prove incorrect in the final implementation. Furthermore, strategic projects typically have a bunch of stakeholders and subsequent projects to serve. Have one stakeholder to become the project's sponsor [*MannsRising2005*], and focus on his aspects only. Define the project to be less strategic at first, its success depending on its usefulness for the sponsor.

Follow-up efforts can take care of other aspects and another stakeholder. DE-VISUALIZE STRATEGIC PROJECTS can be applied in a PIECEMEAL GROWTH manner, growing the number of stakeholders, requirements, and amount of visibility.



## Combinatorial Budget

Applies to projects with many dimensions of variability.

Sheer size is the key risk to unmastered complexity. The combinatorial explosion of many variables defines the technical size of the project, contributing to implementation, test, installation, and maintenance.

Variability can help you to satisfy different users with the same application, but variability multiplies the effort in implementation and testing.

Variability can compensate for uncertainty and cover indecisiveness, but it contributes a complexity factor that increases the project risk significantly.

**Therefore**, budget the amount of variability and configurability in the same manner as you budget resource consumption. Allow the customer to select a small number of configurable items. Whenever the demand for further configuration arises, the necessary budget needs to be freed by removing variability in another area of the application.

Be sure not to miss the relevant variability factors of your application. These may include: number of product variants; number of options a user may order; number of releases (versions) that need to be maintained in the field; number of other applications for interaction; number of configurable items for installation or usage. While these factors do not directly multiply, their consideration easily turns one application into several 1.000 applications to develop, test and support.

The combinatorial budget needs to be defined and negotiated with product owners respectively product managers. It is mandatory to trade combinatorial factors against each other, and not try to enable a high combinatorial factor with a larger team or a prolonged development time. These would be additional factors to the overall product complexity. Also take care that the maintenance costs are included in that subsequent development projects can be scheduled less aggressively.

The COMBINATORIAL BUDGET is related to the COMPLEXITY BUDGET [Marquardt2005] that also includes metrics from organization and design. The key to dealing with complexity is to turn as many contributing factors as possible into chosen factors, and then eliminate them. Variability needs to be discussed with the product owner, while organizational changes need to be agreed with the organization owner.

## Piecemeal Growth

Applies to projects that are large and hard to comprehend.

The project team needs to react to incomprehensible situations, dealing with many issues and requirements at once.

Following a plan avoids unnecessary mistakes during the project's course,  
but it cannot describe necessary changes due to gained experiences.

Plans can be established for anticipated circumstances,  
but a project exploring new territory will experience the unplanned.

Risk management prepares project management to cope with the unexpected,  
but changes and learning experiences will leave the range of anticipated risks, and will exhibit unknown challenges and chances.

**Therefore**, introduce an attitude into the project to solve problems one at a time. Reduce the amount of things to care for at once by focusing on the next few important things, only one or two per person. Adapt an attitude that actively refuses to plan ahead for complex issues, even if that would seem smart and apparently could reduce the overall effort. The question to ask is: what could we try or show next?

PIECEMEAL GROWTH [FooteYoder1998] helps on problem domain as well as on solution domain complexity. Since “the problem with Big Design Up Front is the big, not the up front”<sup>3</sup>, it slices the problem to portions that are comprehensible. While establishing this culture, some amount of stubbornness helps. Refuse any task that would take days, is not immediately in reach, and has links with other tasks – as long as there is still some gain possible with less coupling.

This attitude can best be transported by an external mentor or coach brought into the project. Novices often apply it by themselves, but seasoned engineers can benefit from a frequent reminder to ignore some assumed facts and focus on each function individually. Without external help, it could become tough to actively ignore some of the company culture.

PIECEMEAL GROWTH is a counterpart to DIVIDE AND CONQUER, it describes the iterative and incremental nature of progress as contrasted to independent progress in different areas. It also contrasts to GATHER DOMAIN KNOWLEDGE, where the perceived complexity is reduced by increasing the personal ability of comprehension.

---

<sup>3</sup> Proverb of unknown origin, mentioned during the workshop at EuroPLoP 2009

## Delegate Complexity

Applies to complex projects with a competent team.

You cannot deal with all complexity the project offers, and you cannot control it.

Complexity cannot be controlled or planned,  
but what is considered as complex varies with personality.

Complexity scares many people away,  
but engineers and technical leaders love to handle complex topics, and are proud of their problem solving abilities.

**Therefore**, share dealing with complexity. Decide who of the team shall deal with which topics. Give the authority to deal with complexity to the team members who are willing and able.

Complexity has a strong link to cognitive psychology. Whether some endeavor is considered complex depends on the undertaker and his perception. Individuals with a strong attitude to problem solving and the ability to abstract thinking likely perceive problems as simple that would overwhelm other people.

There are common pitfalls when managing complexity:

- Risk avert managers would try to avoid complexity; this is helpful though not always possible.
- Managers with a strong technical background would engage themselves on the most difficult and exciting topics, and become a bottleneck within the project [*Coldewey1998*] while neglecting their management duties.

Develop the habit to approach team member strong in analysis or design, and discuss difficult problems with them while these are not urgent yet. Discuss informally every once in a while: your peers will have a different but helpful view on many aspects that you had viewed as hopeless. Once some complex issue becomes urgent, you know who could handle some weight and you have established communication mechanisms.

Dealing with difficult problems adds to the job satisfaction and to the reputation of engineers. It gives them positive visibility. To DELEGATE COMPLEXITY both gets managers more help, and prevents communication faults that otherwise could cause the brightest people to quit.

The caveat is that you need to know when to stop. Exposure to complexity often equals exposure to conflicting goals and company politics. Employees need to be backed that once they are overstrained, they can return to technical tasks.

## Management by Triggers

Applies to managers with large teams and complex project settings.

When a project gets out of hands, adding even further levels of control and tracking is hardly possible and rarely helpful.

Uncontrolled projects do not allow monitoring or informed decision making, but controlling and tracking a project costs effort and must only be done for a clear purpose.

Control does not answer the important questions from within the project, and may discourage initiative of the project team, but desired behavior can be triggered by inducing thoughts and mindset.

**Therefore**, set triggers to invoke desired behavior in the mid term.

When projects get out of hands, many managers react by increasing the level of control. However, detailed process instructions or tight tracking adds further complexity to the process, and minimizes the initiative of the project participants to cope with complexity.

Refrain from adding more control to a project that is fundamentally uncontrollable, it would just be costly and result in frustration on all sides. Change the fundamental assumptions from the leaders solving the problems, to each participant solving the problems. Loosen on the “how” side, and take a look at the “who” side: who is the right one to finish the job? And then, what trigger can I set to foster initiative and create a supportive project team?

Choosing good triggers is a virtue that parents learn with their children. Make others think instead of providing them with solutions. Ask questions, guide team members beyond the scope of their daily duties. Give (non-monetary) incentives to team members that evolve beyond their work assignments.

---

Richard Gabriel tells the story of two classes joining a one-week pottery course. The first class is asked to build the most beautiful vase they can, the second one to build as many vases as they can. In the end the best vases from the second class are the most beautiful. The teacher has set a trigger that led to a much higher level of experience in creation and in judgment.

In a large software team, changes to already finished code became unwanted since several clients to that code did not want to change their code in return. However, the system was still in development; restricting change would have stalled progress. Relaxing on code ownership, the team agreed that who wants to change some code also needs to cleanup the entire code base. The undesired workload prevented thoughtless changes. However, the changes that were still tackled by some engineer managed to earn acceptance by all developers [Marquardt2007].

## Local Decision Communities

Applies to projects with a large team.

Project team members' decisions need to be in synch with the overall architecture, without asking for approval individually.

Conceptual integrity is best created by asking a single mind for advice, but a central approval person is a bottleneck for project progress, and no process step could replace implementation of actual functionality.

Homogenous approaches increase the maintainability of software, but they introduce a tight coupling between engineers and tasks, and the need to establish concepts prior to implementation hinders the immediate progress.

**Therefore**, enable developers to take local decisions. Encourage a communication culture of small neighborhoods. Within these developers can develop a common spirit without the need to share this spirit with the entire project team. Establish very few rules to adhere to, the core of the overall architecture.

LOCAL DECISION COMMUNITIES are the more useful, the higher the costs of communication within the project team are. Local decisions are virtually unavoidable then, and better named and planned for. Distributed development is a typical example: interfaces and strategies can be aligned, but the implementation is subject to a local team. Any rigor, in architecture or process, requires control measures that quickly become overly hard to implement and maintain. Less rigor can actually enable new ideas, and increase the project's options for reaction.

However, several kinds of decisions should not be considered local since they have a tendency to influence other teams. Useful advice depends on the kind of system at hand; typical examples are

- the processing model, with infrastructure and communication design;
- transactions, notifications, pull-push and provider-retriever decisions;
- cross-cutting services like security, failure handling, and system startup.

The potentially harmful effects of inadequate local decisions can be measured and addressed by an INTEGRATION FIRST ARCHITECTURE [Marquardt2004]. Furthermore, local decisions also increase complexity, especially when they are not backed by adequate experience. To prevent effects only visible at system maintenance, the system architects may apply a DEFINED NEGLECTION LEVEL [Marquardt2004a] at one decision level inside the individual teams. Some mentoring and frequent contact between the teams' key developers can provide sufficient early warning signals.

## Gather Domain Knowledge

Applies to organizations that run many projects in related domains.

When complexity is a key problem to projects, and the ability to deal with complexity is largely depending on individuals, what should the team members learn and do to increase their personal ability to cope with complexity?

Each defined processes guides you through a project,  
but no process can replace knowledge about what is important to the project at hand.

Engineers need to be knowledgeable in the solution domain,  
but the project team has to fulfill expectations in the application domain.

Complexity is perceived largest where you leave your comfort zone of situations you are familiar with,  
but additional experience and expertise will expand your area of familiarity.

**Therefore**, increase your knowledge about the application domain. You will be able to apply your own judgment, and reduce the complexity associated with unknown settings and questions.

Software development process models typically place the domain expertise outside of the development team.<sup>4</sup> The development team should focus on generic planning and problem solving, keeping the project on track no matter what the domain. However, most large companies have their own development departments, or they cooperate with partners that are familiar with their business and domain for many years. The reason is a risk reduction, stemming from an increased ability to cope with complexity.

Clients look for competence that guides them and sees their problem. Similar to house building, customers expected to be guided to what they want. The choices they make cannot break the project, but make their house a more or less comfortable home. Architects and project managers never allow choices known to cause trouble. The decisions customers make are on the problem domain side and define the project's inherent complexity.

A standard development process can serve as a solid foundation, and remove risk during implementation. However, generic methods have an attitude of carelessness in domain responsibility. However, it cannot replace domain expertise. Knowing your domain and the way the project owner and software user thinks reduces the complexity dramatically.

---

<sup>4</sup> Following a technical interpretation of DIVIDE AND CONQUER, vendors occasionally declare that some tool implemented by their developers can let the domain experts express their knowledge without further communication and adaptation. However, the successful creation such a tool requires exactly that domain knowledge.

## Outroduction

It is interesting to see how coping with complexity reveals parallels between traditional and agile approaches. Pragmatism takes the best of both worlds, and leaves aside all dogma. You need courage, sometimes ignorance (and courage to ignore your best intentions), and local adaptations. Add humbleness and appreciation, and take control of your project!

## Acknowledgements

Many thanks to Markus Völter, my knowledgeable and challenging shepherd for EuroPLoP 2009. Further thanks to the workshop participants at EuroPLoP 2009 for their valuable feedback: Rene Bredlau, Eduardo B. Fernandez, Michael Kircher, Claudius Link, Dietmar Schütz, Alain-Gearges Vouffo Feudjio, and Markus Völter.

## References

- Beck1999* Kent Beck, Extreme Programming Explained.
- Brooks1995* Frederick P. Brooks Jr.: The Mythical Man Month. Anniversary Edition, Addison-Wesley 1995
- c2lampson* found at <http://c2.com/cgi/wiki?ButlerLampson>
- Coldewey1998* Jens Coldewey: Lazy Leader. Available at <http://www.coldewey.com/publikationen/Management/LazyLeader.8.html>
- FooteYoder1998* Brian Foote, Joe Yoder: Big Ball of Mud. In: Pattern Languages of Program Design, edited by Neil Harrison, Brian Foote, Hans Rohnert, Addison-Wesley 1998
- Frisch1957* Max Frisch: Homo Faber.
- MannsRising2005* Mary Lynn Manns, Linda Rising, Fearless Change. Addison-Wesley 2005
- Marquardt2004* Klaus Marquardt: Platonic Schizophrenia. In: Proceedings of EuroPLoP 2004
- Marquardt2004a* Klaus Marquardt: Ignored Architecture, Ignored Architect. In: Proceedings of EuroPLoP 2004
- Marquardt2005* Klaus Marquardt: Indecisive Generality. In: Proceedings of EuroPLoP 2005
- Marquardt2007* Klaus Marquardt: Zeus: Innovation in Life-Supporting Systems. In: Cutter IT Journal, Vol. 20, No. 5, May 2007
- Marquardt2008* Klaus Marquardt: Sisyphean Leadership. To appear in: Proceedings of EuroPLoP 2008
- Wikipedia* found at <http://en.wikipedia.org/wiki/Complexity>

# Handling Variability

Version 2.0, December 11, 2009

Markus Völter, Independent/itemis  
([voelter@acm.org](mailto:voelter@acm.org))

© 2009 Markus Voelter

*Copyright retain by author(s). Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website*

---

## Introduction

Traditionally, in software engineering, development happens for single products. This is a very inefficient approach in cases where groups of products are related. Software product line engineering [1] is about systematically developing families of related products together, as a product line. The products within a product line usually have many things in common, but also significant differences. Managing and implementing these differences can become complex because in realistic product lines, variability abounds, and it is often a cross-cutting concern. Hence, to exploit the benefits of product line engineering, it is important to systematically manage the variability between the products.

Variability denotes differences between related products in a product line. Typically one talks about variation points, where, to define a product, you need to bind each variation point. There are different ways to bind a variation point: setting a value, selecting an option or implementing a program fragment or model (we'll talk about this in the patterns below). For each variation point, you'll also have to define the binding time: at design time, load time, runtime, etc.

This paper is a collection of patterns for handling variability in software systems. It contains patterns for managing variability, introduces different kinds of variability, and illustrates realization of variability in implementation artifacts such as models or source code. The patterns are intended as a contribution to a more comprehensive pattern language on product line engineering.

The paper is intended to be read by architects who want to get a better grasp on managing and implementing variability. The paper does not address requirements and product management. I assume the requirements that drive the variability are known.



---

## Structure of the Paper

The paper is structured into three sections, *Managing Variability*, *Classes of Variability* and *Implementing Variability*.

*Managing Variability* provides two approaches on how to reduce the overall complexity that results from variability. One pattern, SEPARATE DESCRIPTION OF VARIABILITY recommends the separation of the logical description of variability from its implementation. The other one, MODEL-BASED IMPLEMENTATION describes how and why to use domain-specific models to capture variability.

The second chapter, *Classes of Variability*, contains two patterns, CONFIGURATION and CONSTRUCTION. The level of expressiveness of these two approaches is fundamentally different, and you have to make a conscious decision for one of these when thinking about how to describe variability.

Chapter three, *Implementation Strategies*, deals with lower-level mechanisms for representing variability in implementation artifacts. It consists of three patterns: REMOVAL, where you conditionally take something away from a whole, INJECTION where you conditionally add something to a minimal core, as well as PARAMETERIZATION where you define a variant by providing values for a predefined set of parameters.

---

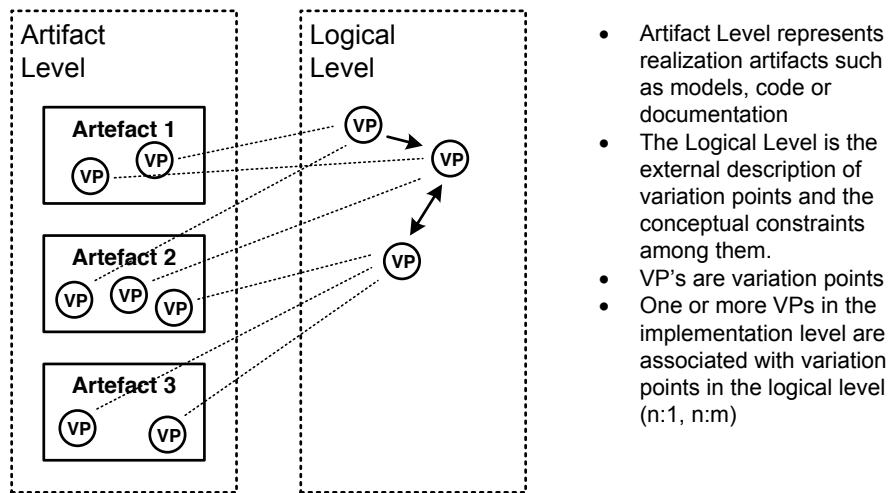
## Representing Variability

In large product lines with many products and many differences between the products, the variability inherent to implementation artifacts can easily overwhelm developers. The overhead for representing, organizing or managing the variability can become so complex that the potential benefits of product line engineering cannot be realized

**How can you represent the complexity introduced by variability in implementation artifacts?**

### Separate Description of Variability

**Make sure that the logical description of variability is separate (external) from its realization in the implementation artifacts. The logical description describes the variation points, the variants, as well as constraints between these variants. The realization of the variability in the implementation artifacts is tied to the logical variability description.**



As a consequence of the cross-cutting nature of variability in many of today's systems, the implementation of variability is scattered over many implementation artifacts. However, in many cases several variation points need to be configured in a consistent, mutually dependent way for the resulting product to work. If each has to be configured separately, the overall complexity grows quickly. By identifying logical variation points, and then tying the (potentially many) implementation variation points to these logical variation points, related implementation variations can be tied together and managed as one. With reasonable tool support, you can also select a logical variation point and navigate to all the implementation variation points, providing a level of traceability. When customizing the artifact level based on a configuration of the logical level, the mapping should be automated, but doesn't have to be.

In most cases, the logical variability is also much more closely aligned with the problem domain. The variability in the artifacts corresponds to the solution domain. Consequently, meta data (why does the variability exist, which stakeholders care about it, etc.) is associated with the logical level. The logical level is typically visible to the non-developer stakeholders.

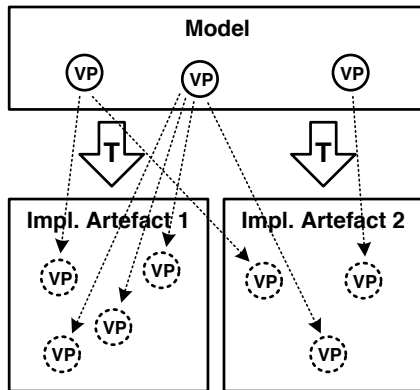


One way of separating logical variations from implementation variations is using feature models. A feature model [2] describes features and their dependencies in a hierarchical fashion. Implementation artifacts or artifact processors can refer to those features and construct the product variant accordingly.

An alternative approach is OVM, or orthogonal variability models [3]. In contrast to feature models, they are not hierarchical. Technically, they don't describe features and their relationships but rather variation points. The two representations are semantically equivalent.

# Model-Based Implementation

Describe the implementation of the system with high-level constructs, such as models based on domain specific languages, and a subsequent transformation, interpretation or code generation step. Because of the closer alignment with the actual problem domain, variability is much more localized, and the number of variation points is significantly reduced in models compared to code.



- A model describes domain abstractions in a formal and concise way
- Transformations map that model to (typically more than one) implementation artifact
- Variability is expressed with fewer VPs in the models compared to implementation artifacts

## Model-Driven Development

In Model-Driven Development, we develop domain-specific languages that are very closely aligned with the domain at hand. Consequently, when using such a DSL (Domain-Specific-Language) to describe a system in that domain, the resulting models/programs become very concise. There's much less repetition and low-level detail in the description.

In a subsequent step, code generation (and sometimes interpretation) is used to map the models to implementation code. The knowledge about how to "expand" the knowledge in the models to implementation code is encoded into the code generator.

If you can describe something with a smaller amount of "stuff" (i.e. code, configuration files, etc) on a more abstract domain specific level, and then use the transformation or generator to expand all the details, you can simply implement the variation on the more abstract level in one place. The trade-off is, of course, that you have to define this high-level domain specific language, including a way to define variants of programs written in that language. You also need to define the transformation down to the actual implementation artifacts.

The relationship of this pattern to SEPARATE DESCRIPTION OF VARIABILITY is interesting. As the name suggests, the models mentioned in this pattern play the role of the implementation/artifact level in SEPARATE DESCRIPTION OF VARIABILITY. The logical description "customizes" the models which are then further mapped down to code. In some cases the models in this pattern play the role of the logical level and are not further customized by an additional logical level.

≈≈≈

Consider the case where the attributes of an address entity need to be variable. For example, in the US version of the system the address needs to have a state attribute. In European countries this is not necessary. The state attribute needs to be taken into account in the UI, the data structures, the database table structure, the SQL code to persist the data, and maybe in several other places. Instead of implementing the variability in each of these places, you can simply put one variation into a model that describes the data structure, and then use

code generation to derive the UI, the data structure, the database table creation statements, as well as the SQL code from that model.

Another, similar example is the implementation of state-based behaviour. If it is implemented directly with a programming language, you have to use either the State pattern, a big switch/case statement, a number of arrays pointing into each other, or state tables, together with a number of constants representing the states, events and transitions. If the state-based behaviour should be variable, implementing this variability on the level of the implementation is very tedious and error prone. An alternative approach is to directly describe the behaviour as a state machine using a suitable language, together with an interpreter written in the target language. Making some of the states, events or transitions variable requires only *one* change (for each variability) in the model and no changes to the interpreter, reducing the overall complexity significantly.

---

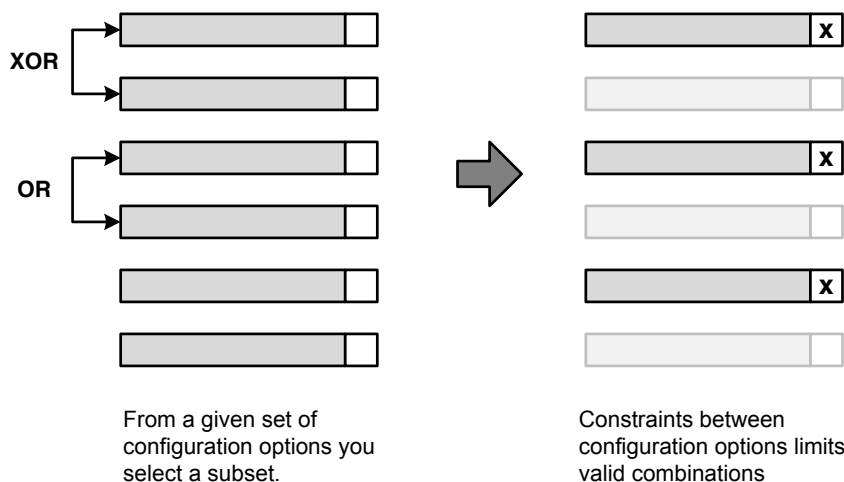
## Classes of Variability

Regarding the definition of variation points and the mechanisms to define the variants, there are several alternatives with different levels of expressiveness.

**How can the different alternatives be grouped according to their expressiveness?**

### Configuration

**A variation point allows the selection from several alternatives. Each alternative is either in the system or not. Constraints between the alternatives limit the valid combinations.**



The biggest advantage of configuration is its simplicity. People don't have to learn complex formalisms for defining a variant, they simply select from a predefined set of alternatives. Invalid selections are avoided by

restricting the valid combinations. To achieve this, constraints (*requires*, *prohibits*, *recommends*, *discourages*) are defined between the configuration options. Of course there are limits to what you can do with configuration only. For example, cardinalities, instantiation or relationships cannot be expressed very well. This can be seen as an advantage (makes the configuration process simple) and as a liability (the degrees of freedom are limited).

If you want end-users to configure your product, you should try to go as far as possible with configuration only.

≈≈≈

In the simplest case, configuration can be achieved by simply setting flags in a configuration file.

In C compilers, the ability to define symbols which are then evaluated by *ifdefs* is another way of configuration. Another alternative is using the Bridge or Strategy patterns. These support “plugging in” different implementations at a specific variation point. In contrast to preprocessors, they are bound at runtime using polymorphism in object oriented languages.

A more powerful formalism for configuration variability is feature models [2]. Feature models are hierarchical collections of flags (features) that can be selected or not. There are several default constraints between such features: mandatory (the feature must be included), optional (the feature might be included), alternative (exactly one of the set of features has to be included) and or (one or more from a collection of features has to be included). Powerful tools exist to manage even large sets of features and their relationships.

Most wizards are also a kind of configuration. You are guided through a number of selections and parameter specification. What you have selected in steps 1 through *n* possibly determines the options you can select from in options *n+1* through *k*, a form of constraints. From the resulting overall configuration some kind of artifact is generated or some functionality is executed.

## Construction

**A language is provided to define the variant. The definition of the variant is a sentence in this language.**

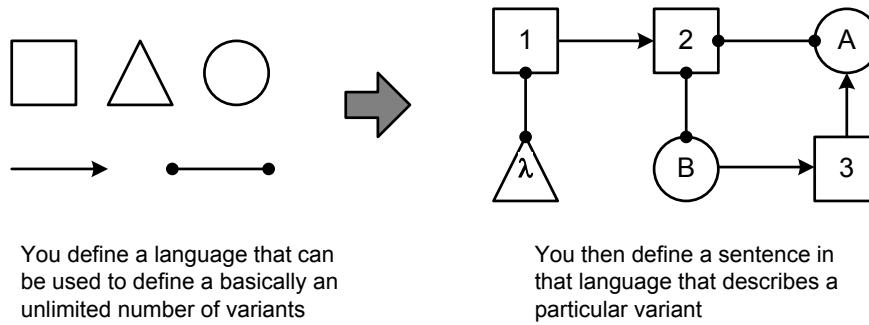
### The C Preprocessor

The C and C++ language family includes a preprocessor that can process the source code before it is submitted to the actual compiler.

One of the features of the preprocessor is to conditionally remove a region of code. To do that, you have to use `#ifdef`:

```
#ifdef aSymbol
    // here is some code
#endif
```

The code between the `#ifdef` and the `#endif` is removed (and hence, not compiled) if *aSymbol* is not defined. A symbol can be viewed as a boolean variable, and defining a symbol means to set it to *true*.



Construction is much more powerful than configuration, since it provides an unlimited variant space. The language defines a grammar (or a meta-model) and all valid instances are valid variants. Picture this in the illustration above: you can always add one more box and line. Depending on the language definition, construction can also be much more complicated to use than configuration, because of the unlimited variant space. However, it can be used to express relationships, instantiation and cardinalities.



The most well-known example for construction is simply programming languages. Frameworks define hooks into which the developer can plug in code, as long as it conforms to a certain interfaces or other framework imposed constraints. Essentially, the variability is unlimited.

Whenever domain specific languages [4] are used to configure a product, then this is also construction. The variability is more limited, i.e. domain specific, but almost all DSL grammars allow for unlimited variability.

The composition of a system from components that are then hooked up in order to communicate is also a form of construction. This hooking up can, for example, happen through a dependency injection framework or through any other means of configuration file.

## Combinations

Of course, configuration and construction can be used in conjunction.

- A complex system can be subdivided into several subsystems, where possibly one set of subsystems is configured by a configuration and another set of subsystems will be configured by construction.
- Configuration can be superimposed onto construction, where a constructively created variant is customized by configuration. This can be achieved using Removal or Injection, as explained below.
- It is also feasible to use construction to provide details to configuration. Many configuration options have parameters (see Parameterization below). The type of such a parameter can be a

construction language. Every instance of a construction language would be a valid value for the parameter.

---

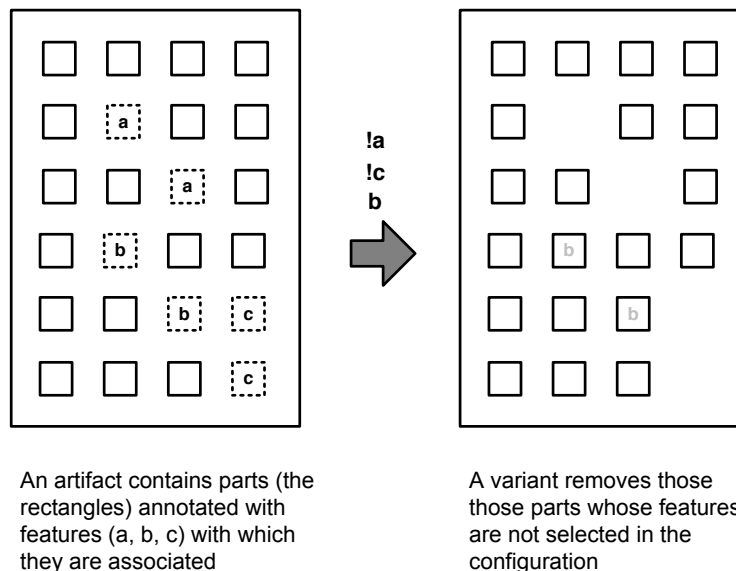
## Implementation Strategies

Now that we have defined the various classes of expressiveness we can look at the actual implementation of variability in implementation artifacts.

**How can variability be implemented in implementation artefacts?**

### Removal (aka negative variability)

**Remove parts of a comprehensive whole. This implies marking up the various optional parts of the comprehensive whole with conditions that determine when to remove the part.**



The biggest advantage of this approach is its apparent simplicity. However, the comprehensive whole has to contain the parts for all variants (maybe even parts for combinations of variants), making it potentially large and complex. Also, depending on the tool support, the comprehensive whole might not be a valid instance of the underlying language or formalism. For example, in an IDE, the respective artefact might show errors which makes this approach annoying at times. Because of its technical simplicity, the approach can be easily retrofitted to all kinds of artifacts: documentation, code, models.

≈≈≈

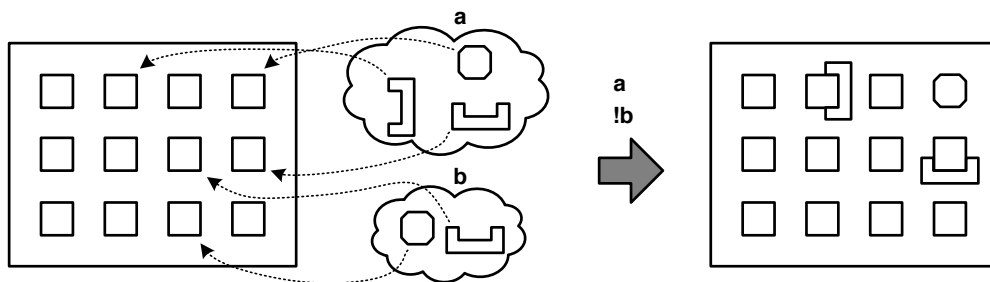
ifdefs in C and C++ are a well-known example of this strategy. A preprocessor removes all code regions, whose ifdef condition evaluates to false. When calling the compiler/preprocessor, you have to provide a number of symbols that are evaluated as part of the ifdef conditions.

Conditional compilation can also be found in other languages. Preprocessors that treat the source code simply as text are available for many languages and are part of many PLE tool suites, such as pure::variants [5] or Software Gears [6].

The Autosar [7] standard, as well as other modeling formalisms, support the annotation of model elements with conditions that serve the same purpose. The model element (and potentially all its children) are removed from the model if the condition evaluates to false.

## Injection (aka positive variability)

**Inject additions into a minimal core. The core does not know about the variability, the additions point to the place where they need to be added.**



A base artifact made of various parts (the small rectangles) exists. There is also variant specific code (the strange shapes), connected to features external to the actual artifact and pointing to the parts of the artifact to which they can be attached.

Defining a variant means that the variant specific code associated with the selected features are injected into the base artifact, to the parts they designated.

The clear advantage of this approach is that the core is typically small and contains only what is common for all products. The parts specific to a variant are kept external and added to the core only when necessary.

To be able to do this, however, there must be a way to refer to the location in the minimal core at which to add a variable part. This either requires the markup of hotspots or hooks in the minimal core or some way of pointing into the core from an external source. In the latter case, the core requires no modification and the approach can be used for implementing unexpected variability.

≈≈≈

Aspect Oriented Programming (AOP) [8] is a way to implement this strategy. Pointcuts are a way of selecting from a set of join points in the base asset. A joint point is an addressable location in the core. Instead of explicitly defining hooks, all instances of a specific language construct are automatically addressable.

Various preprocessors can be used in this way. However, they typically require the explicit markup of hooks in the minimal core.



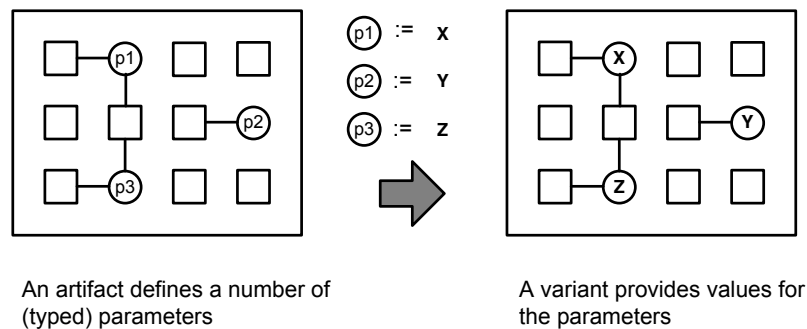
For models, injection is especially easy, since in most formalisms model elements are addressable by default. So it is possible to point to a model element, and add additional model elements to it, as long as the result is still a valid instance of the meta model.

The installation of optional packages for software systems is another example of this pattern.

An example in the architectural patterns world would be the Microkernel [9]. A microkernel-based system is one that provides a minimal set of functionality in its base functionality together with a protocol for plugging in additional pieces of functionality that makes use of the functionality in the microkernel, or other additions.

## Parameterization

**The artifact that shall be varied needs to define parameters. A variant is constructed by providing different values for those parameters. The parameters are usually typed to restrict the range of valid values.**



The artifact that shall be parameterized needs to explicitly define the parameters, as well as a way to specify values (this makes this approach different from injection where it is possible to make it work without marking up the minimal whole). Hence, the variability is limited to the locations where parameters are defined. The core has to query the values of those parameters explicitly and use them for whatever it does. The approach requires the core to be explicitly aware and define all parameters, unexpected variability cannot be handled.

In most cases, the values for the parameters are relatively simple, such as strings, integers, booleans or regular expressions. However, in principle, they can be arbitrarily complex.

≈≈≈

A configuration file that is read out by the using application is a form of parameterization. The names of the parameters are predefined by the application, and when defining a variant, a set of values is supplied.

The strategy pattern is a form of parameterization, especially in combination with a factory. A variant is created by supplying an implementation of an

interface defined by the configurable application. Once again, the application has to explicitly query the factory, and the type of the values is defined by the interface which its strategy classes implement.

All kinds of other small, simple, or domain specific languages can be used as a form of parameterization. A scripting language in an application is a form of parameterization. That type of parameter is "valid program written in language X". Also, systems where some kind of behavior can be configured using workflow languages, activity diagrams, state machines or business rules is a form of parameterization. In this case, too, the languages used to define the behavior are the type of the parameter.

The classical approach of copying resources is also a form of parameterization. Consider the place where a logo is exchanged. The application defines a parameter ("logo for the company"), the type being the file type (such as GIF, 32x32 pixels) and the parameter is any valid image that makes sense as a logo for the company.

## Combinations

Of course there are also combinations of all of these approaches. Going back to the component example introduced in the CONSTRUCTION pattern, components that are wired together often also use PARAMETRIZATION to implement another, smaller grained form of variability.

Another combination is using PARAMETRIZATION to determine which parts are REMOVED or INJECTED.

---

## Acknowledgments

I want to thank my EuroPLoP 2009 workshop for the feedback on this paper: Christa Schwanninger, Klaus Marquardt, Didi Schütz, Rene Bredlau, Claudius Link and Ed Fernandez.

Thanks to Iris Groher for providing feedback on earlier version of this paper. I also want to thank my EuroPLoP 2009 shepherd Michael Stal for his repeated useful feedback.

---

## References

- [1] <http://www.softwareproductlines.com/introduction/introduction.html>
- [2] [http://en.wikipedia.org/wiki/Feature\\_model](http://en.wikipedia.org/wiki/Feature_model)
- [3] Klaus Pohl, Günter Böckle, Frank van der Linden, Software Product Line Engineering. Foundations, Principles, and Techniques, <http://www.amazon.de/Software-Engineering-Foundations-Principles-Techniques/dp/3540243720>
- [4] [http://en.wikipedia.org/wiki/Domain-specific\\_language](http://en.wikipedia.org/wiki/Domain-specific_language)

- [5] [http://www.pure-systems.com/pure\\_variants.49.0.html](http://www.pure-systems.com/pure_variants.49.0.html)
- [6] <http://www.biglever.com/solution/product.html>
- [7] <http://www.autosar.org/>
- [8] [http://en.wikipedia.org/wiki/Aspect-oriented\\_programming](http://en.wikipedia.org/wiki/Aspect-oriented_programming)
- [9] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Pattern Oriented Software Architecture, Vol 1,  
<http://www.amazon.com/Pattern-Oriented-Software-Architecture-System-Patterns/dp/0471958697>

---

# Test Automation Design Patterns for Reactive Software Systems

A.-G. Vouffo Feudjio, I. Schieferdecker  
Fraunhofer Institute for Open Communication Systems (FOKUS)  
Alain.Vouffo, ina.schieferdecker@fokus.fraunhofer.de

## Abstract

*Patterns have been successfully applied in software development to improve the development process, by facilitating reuse, communication and documentation of sound solutions. However, the testing domain is yet to benefit from a similar approach. This although, with the growing complexity of test automation solutions, identifying and instrumenting patterns in test design to facilitate reuse appears to be a promising approach for shortening the development cycle and save costs. This paper presents a collection of patterns for designing test automation solutions for reactive software systems and reports on first experiences of applying those patterns in a case study.*

## 1 Introduction

It is now widely acknowledged that testing is no longer just an art, but an engineering discipline in its own, with test development following a similar process as generic software development. Patterns are a canonical documentation of the essential concepts underlying successful solutions to recurrent engineering and design problems. They are used to capture experiences, expertise, and facts to improve system quality and facilitate the production of new solutions. In previous publications, we proposed a similar approach for the design and implementation of tests automation systems [23]. This is particularly interesting with the growing popularity of Model-Driven Testing (MDT), which raises the level of abstraction for test design, to a degree that it allows reuse of concepts for new solutions.

In this paper, we present a selection of test patterns we have collected so far by performing pattern mining on existing test automation solutions designed using different test design and test scripting notations e.g. the Testing and Test Control Notation (TTCN-3) [12] the UML Testing Profile(UTP) [21] or JUnit [15]. Each test pattern is defined along a template we introduced in previous work [23], but which was refined to align with generic pattern methodologies. This work is organised as follows: Section 2 presents a selection of the test patterns we have identified, then Section 3 reports on a case study in which a prototype tool implementing some of those patterns was used to design a conformance test suite for the IP Multimedia Subsystem communication protocol (IMS). Section 3.2 discusses some related work in this area, before Section 5 concludes the paper and draws an outlook for further research.

*Readers familiar with the pattern concept can skip parts of section 3 and go directly to section 2.1, where the description of patterns starts.*

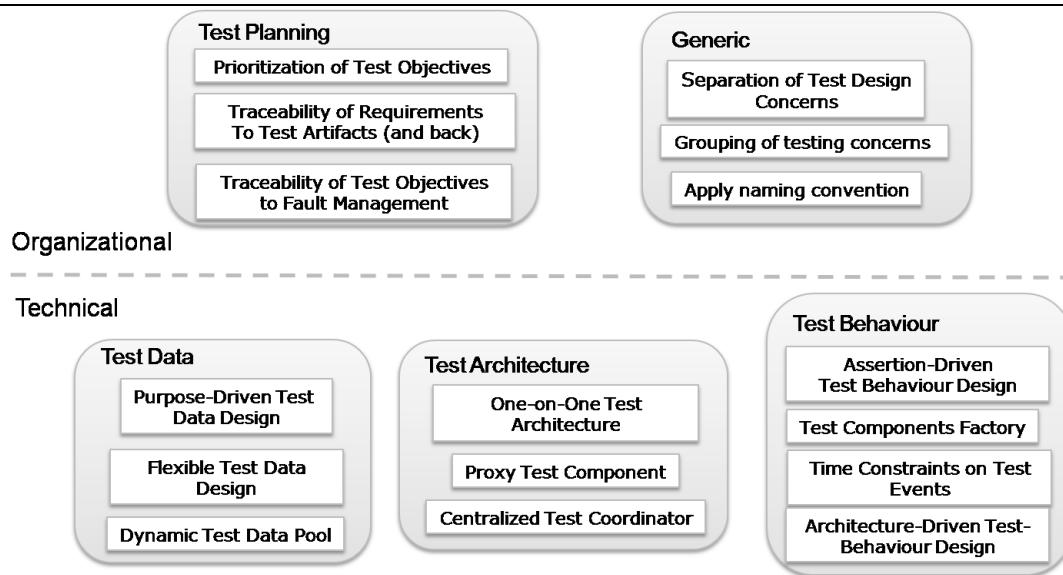


Figure 1. Overview of Identified Test Patterns

## 2 A Collection of Test Patterns for Black-box test design

We consider test engineering to be a process that starts from the definition of test objectives via abstract test models through to executable test cases. We classify test patterns along the various activities of that process into the following categories:

- *Generic test design patterns* are those applicable to all activities of the test engineering process.
- *Test objectives design patterns*: According to IEEE 829 [16], a test objective<sup>1</sup> is a brief and precise description of the special focus or objective for a test case or a series of test cases. Test objectives can be viewed as the equivalent to system requirements in product development and will certainly benefit from the application of patterns in a similar way as with requirements engineering (RE) patterns [13]. Accordingly, *test objective design patterns* are those addressing that activity in the test engineering process for a given system under test (SUT).
- *Test procedure design patterns*: A test procedure is defined as a prose description of a sequence of actions and events to follow or to observe for executing of a test case. A test procedure describes how a test objective will be assessed. Therefore, Test procedure design patterns are those that are applicable when designing the test procedures for a given SUT.
- *Test architecture design patterns* define good practices and established recommendations in selecting and designing appropriate test architectures. The test architectures describe the topology of a test system, i.e. its composition as a collection of (parallel) test components, interconnected among each other and with the SUT and communicating through Points of Control and Observation (PCOs). Depending on the overall goal of a test e.g. conformance, performance, functionality, robustness, etc., different test architectures are suitable.

<sup>1</sup>Test objectives are sometimes also referred to as *test purposes*, *test requirements* or *test directives* in the literature

- 
- *Test data design patterns* describe approaches for designing the data used in test scenarios as stimuli for the SUT or to express assertions, based on which the SUT's response will be evaluated to assess if they meet their requirements or not.
  - *Test behaviour design patterns* document approaches and principles for designing the behaviour of test systems, i.e. the patterns of interactions between entities in a test architecture.

The ultimate goal of test patterns is to increase the quality of tests. Which leads us to the issue of defining the characteristics of test quality. The ISO/IEC 9126 standard [17] defines a model for internal and external quality of software, including quality characteristics and associated metrics.

Zeiss et al [26] demonstrated the applicability of that model to tests and came up with a model combining test-specific quality characteristics such as test effectiveness and test efficiency with more generic ones such as (re-)usability, maintainability, portability etc. However, the value and applicability of those characteristics will mainly depend on the chosen test design strategy. Binder [2] classifies test design strategies in four main categories: Responsibility-based, implementation-based, hybrid and fault-based test design. Responsibility-based test design uses specified or expected responsibilities of an SUT to design tests and is synonymous to “black-box”, “specification-based”, “behavioural”, “functional” testing. Implementation-based test design relies on internal knowledge (e.g. source code, internal design) of the SUT for test design and is also labelled “structural”, “white-box”, “glass box” or “clear box” testing. Hybrid test design combine responsibility and implementation-based test design. Whereas fault-based testing purposely injects faults in the SUT to check whether those faults are discovered by a test suite.

Although some of the patterns discussed in this work may be applicable to other test design strategies, the main concern is on responsibility-based test design. As expected, this has had repercussions on our methodology for pattern mining. This is illustrated for example by the fact that test effectiveness - i.e. the capability of tests to reveal faults on the SUT - only plays a marginal role for test quality in responsibility-based test, although it's a key factor in implementation-base test design.

Basically, pattern mining for test design patterns can be driven by one or both of the following questions:

- Question 1: What is the best way for designing tests, so that they would help uncovering as many errors of the SUT as possible, before it is delivered to end customers [14]?
- Question 2: What is the best way for designing and modelling tests so that the resulting test specification and/or solution matches best main quality criteria such as reusability, maintainability, understandability etc.?

A pattern template is a list of subjects (sections) that comprise a pattern [2]. The content of the test pattern template depends on which of question 1 or question 2 above is the main driving force for pattern mining. In [2], Binder proposes a test pattern template, which is driven by question 1. Our pattern mining activities are mainly driven by question 2, although question 1 is considered as well. Therefore, we took the test pattern template provided by Binder [2] as the base for our own template, but modified it to reflect the fact that our focus is more on reuse towards more automation than on effectiveness of the tests.

As a consequence subjects such as the *subjects fault model*, the *entry criteria* and the *exit criteria* proposed by [2] were removed from the template. Instead, we added the *applicable test scope* subject to capture the preconditions for applying test patterns. Our test modelling pattern template consists of the following subjects:

- *Pattern name*: A meaningful name for the test pattern.
- *Context*: To which specific context does it apply? This includes the kind of test pattern (organisational vs. design, generic, architectural, behavioural or test data etc.) as well as the test scope for <sup>2</sup>.

---

<sup>2</sup>The test scope describes the granularity of the item under test [19], which may vary from a low-level entity such as class (for unit testing) to a whole software system (for system testing).

- 
- *Problem*: What is the problem, this pattern addresses and which are the forces that come into play for that problem?
  - *Solution*: A full description of the test pattern.
  - *Known Uses*: Known applications of the test pattern in existing test solutions (e.g. test specifications, test models, test suites, or test systems) or by test modelling approaches.
  - *Resulting context*: What impact does this pattern have on test design in general and on other patterns applicable to that same context in particular?
  - *Related patterns*(optional): Test design pattern related to this one or system design patterns in which faults addressed by this test pattern might occur. This section is optional and will be omitted, if no related pattern can be named.
  - *References*(optional): Bibliographic references to the pattern. This section is also optional and will be omitted, if no reference can be provided.

Figure 1 displays an overview of those patterns we have identified so far for each of the categories mentioned above. In the following sections, we present a selected subset of those patterns.

---

## 2.1 Pattern: Separation of Test Design Concerns

### 2.1.1 Context

This pattern is a generic organisational test design pattern and is applicable at any test scope for large size test projects. It is assumed that test development is process running in parallel to the development of the SUT or integrated to it, with both of them having the requirements as a common starting point.

### 2.1.2 Problem

How to organise the file structure of test artifacts. Test artifacts are resources used for storing the design and implementation of a test automation solution. They include high level design models, documentation artifacts through to source code of executable test scripts. The size and the complexity of those test artifacts can grow considerably, raising questions as to how to organise properly to keep a good overview and facilitate collaborative work.

### Forces

- To avoid test design activity becoming a bottleneck to the development process, having different teams working in collaboration on the will speed up that process.
- Synchronisation and version control conflicts between the actors involved in test design may cause resources being wasted to address them.
- Large compilation units increase the risk of potential version control conflicts among parallel developers/designers.

### 2.1.3 Solution

Divide the various tasks over several test designers, by organising modules accordingly. Each task is addressed separately to allow parallel processing. Applying this pattern requires that the technologies involved (e.g. the notation used for designing the tests) provide such mechanisms. Modules may be organised based on the aspect they cover(e.g. Test data, test architecture) or based on the SUT feature they target.

### 2.1.4 Known Uses

Instantiations of this test pattern can be observed in numerous test automation solutions. The code snippet below from the IPv6 conformance test suite [22] displays an example in TTCN-3 of a test script importing elements of other test modules to design test behaviour.

---

```
1 module AtsIpv6_Common_Functions {
2   // Importing Generic Libraries
3   // LibCommon
4   import from LibCommon_BasicTypesAndValues all;
5   import from LibCommon_DataStrings all;
6   . . .
7   // Importing test data modules
8   // LibIpv6
9   import from LibIpv6_Interface_Templates all;
10  import from LibIpv6_CommonRfcs_TypesAndValues all ;
11  . . .
12  // Importing test architecture modules
13  // AtsIpv6
14  import from AtsIpv6_TestSystem all;
```



---

```
15 import from AtsIpv6_TestConfiguration_TypesAndValues all ;
16 . . .
17 } //end module AtsIpv6_Common_Functions
```

---

### 2.1.5 Discussion

A difficulty in applying this pattern consists in ensuring that the number of separate modules remains within sensible limits. Otherwise, the effort of managing all parallel activities can reduce the positive impact of the pattern and even lead to less productivity. However a small number of modules will inevitably lead to more version controlling conflicts, with several people potentially working in parallel on the same modules. In such cases the usage of an appropriate version controlling system, along with clearly defined policies is highly recommended.

### 2.1.6 Related Patterns

This pattern is an application of the Separation of Concern, a.k.a *Divide and Conquer* design pattern known both in generic software engineering, as well as in test design [7].

---

## 2.2 Pattern: Prioritization of test objectives

### 2.2.1 Context

This pattern is an organizational pattern that addresses test objectives design

### 2.2.2 Problem

Due to resource limitations, often not all test cases can be implemented and/or executed at a within the decided deadline. Some key decisions need to be taken confidently for planning the testing activities and to be able to react to changes in a proper way. Example of key decisions include:

- Which test cases need to be implemented and executed first and which ones can be left aside for later stage in the testing process?
- When can test activities be considered sufficient to provide a level of confidence in the SUT, that is high enough to allow its release?

### 2.2.3 Solution

As recommended by IEEE 829[16], introduce a prioritization scheme for test objectives in the test model. Prioritization should be provided for a test objective taken individually or for a group of test objectives. Prioritization of test objectives can be based on factors such as:

- Priority level of the feature or requirement(s) covered by the test objective.
- Level of criticality of the errors targetted by the test objective.

Testing activities (e.g. design, implementation, execution) can then be planned based on the priority level of the test objectives and taking the time and resources constraints into account, to ensure that test cases with highest priority are available on time before product delivery.

### 2.2.4 Discussion

The size of the testing project and the time constraints it faces will be taken into account, whenever the application of this pattern is considered. Obviously, applying the pattern for large scale projects yields more benefits than doing so for smaller ones.

### 2.2.5 Known Uses

Prioritization of test cases is used implicitly in several instances, though it is not always supported by a specific test notation. Generally a separate tool is used to manage that aspect of the test process. However, it would be highly beneficial to integrate it into the test design process, so that relating it to other tasks in the product development process would be more straightforward.

### 2.2.6 References

[8, 9, 6]

---

## 2.3 Pattern: Traceability of Test Objectives to Requirements

### 2.3.1 Context

This pattern is an organizational pattern that addresses the management of large test suites under restrictive time and resource constraints.

### 2.3.2 Problem

To keep control of your development process, you want to be able at any point in time to evaluate the progress of the test project to gain objective criteria for making decisions on the project.

#### Forces:

- 100% code coverage is an illusion
- 100% requirements coverage is achievable, but needs to be supported with clear and sensible metrics.
- Being able at any time to give an estimation of the current coverage of requirements by the specified test objectives will facilitate decision making for releasing the product.

How to achieve traceability between tests and system requirements to enable automatic coverage analysis?

### 2.3.3 Solution

Provide a mean for linking each test objective to a (set of) requirements or features of the SUT that it addresses. Those include functional as well as non-functional requirements. The test objectives will be designed based on potential risks for the SUT related to a particular feature or as a mean for verifying that the SUT meets the requirements

### 2.3.4 Known Uses

Please refer to [24] for an overview of requirements traceability that includes numerous examples of traceability to test artifacts as proposed in this pattern.

### 2.3.5 Discussion

Benefits of this pattern include the fact that the selection of test cases to address specific products or features is facilitated based on the requirements they support. Furthermore, automated requirements coverage analysis of the test cases can be achieved at any time in the lifecycle.

One key difficulty in applying this pattern is to ensure that changes to the test model are propagated in both directions of the link to avoid dead links and keep the test model consistent. The test design tool should take care of that and update a test objective element accordingly, if one of the covered system requirements is altered (e.g. deleted, moved to another location, renamed etc.). Such a propagation of changes could be facilitated by the usage of the same notation or of the same modeling technology (e.g. EMF, MOF) for those aspects being linked with each other. Otherwise, some serious maintainability issues might emerge.

---

## 2.4 Pattern: Traceability of Test Objectives to Fault Management

### 2.4.1 Context

This pattern is an organizational pattern that addresses the management of large test suites under restrictive time and resource constraints.

### 2.4.2 Problem

In spite of all testing efforts, errors in software are inevitable and will eventually occur. We want to avoid experiencing and fixing the same errors many times. How can it be ensured, that the information gathered in analyzing and fixing errors identified at the user end or through testing can be exploited for the benefit of future testing activities and for improving the overall quality of the software product under test?

#### Forces

- Fixing errors is generally granted higher priority than documenting them.
- Besides, who cares about fixed issues?
- Developers lack of time to do such additional presumably activities. So they tend to postpone them until they pop-up again as higher priorities.

### 2.4.3 Solution

Provide a mechanism for ensuring traceability between entries in the fault management system and elements of the testing process. The mechanism should fulfill the following requirements:

- The mechanism should be integrated in the test development/management tool to ensure that it can the process does not cost too much additional effort.
- Every time a failure is (inadvertently or deliberately) discovered on a version of the SUT, make sure that while creating a new entry for that failure in the fault management system, that it is associated with a test objective addressing the root cause of the bug and that test cases are implemented to cover that test objective.
- Provide technical means for enforcing that policy automatically online (i.e. in the process of creating the entries in the model repository) or offline (after the elements have been created)
- Automatically integrating the newly added tests in subsequent regression tests would yield additional benefits.

### 2.4.4 Known Uses

Agile methods apply this pattern by making test development an integrated part of the development lifecycle (e.g. Test-driven development in XP).

### 2.4.5 Discussion

The same type of potential issues identified for the *traceability of test objectives to system requirements* pattern (section 2.3) also apply for this pattern.

## 2.5 Pattern: One-on-One Test Architecture

### 2.5.1 Context

This pattern addresses test architecture design for an SUT that can be viewed as one entity providing a well-known set of entry points and interacting with its environment following a sequential non-concurrent behaviour. Functional testing at unit or system level is the goal.

### 2.5.2 Problem

How to design a static test architecture for achieving testing the SUT with highest possible efficiency.

#### Forces

- Resources planned for testing are generally and straightforward solutions are always welcome.
- The level of complexity of the test system should be kept as low as possible, to keep maintenance and associated efforts as low as possible.
- Usage of concurrency in the test system increases the risk of introducing erroneous test behaviour and the cost of the test system, because a coordination mechanism is required to control the choreography of parallel test components.

### 2.5.3 Solution

Design the test architecture consisting of one single test component connected to the SUT in a way that it can stimulate the SUT and verify its response to those stimuli. One possible way of achieving that is by making the test component a mirrored image of the SUT, e.g. by providing interfaces required by the SUT and using interfaces the SUT provides.

Figure 2 displays two examples resulting from applying that pattern. The upper part of the figure shows a test architecture consisting of a single test component that uses one port both for sending impulses to and receiving responses from the SUT to verify its correct behaviour. On the other hand, the lower part of the figure illustrates a test architecture for an SUT providing three different entry points for stimuli and responses. Benefits: Having a

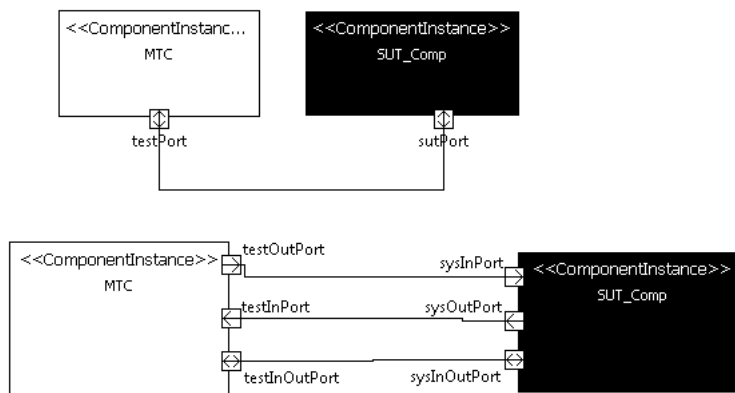


Figure 2. Test architecture Diagram for One-on-One Pattern

single test component implies that synchronization mechanisms based on message exchange or other Remote Procedure Control(RPC) or similar mechanisms do not have to be implemented at the testing side. Variables defined

---

in the test component can be used to describe states based on which decisions can be made on the test verdict. Shortcomings: The test component has to emulate the complete behaviour of system component it replaces. Depending on the level of complexity of that behaviour, this might be more or less difficult to achieve. Furthermore, having a single component makes it difficult to deal with concurrency at the testing side, if required.

#### **2.5.4 Known Uses**

This pattern is applied in numerous conformance test suites, e.g. the collection of IPv6 test suites [22] used e.g. for the IPv6 logo brand ,the IMS benchmark test suite [5] used for performance testing IMS server equipment or the CORBA component test suite [1] used for integration testing of CORBA components

#### **2.5.5 Discussion**

Potential difficulties in handling concurrent behaviour from the SUT and to emulate similar behaviour to stimulate the SUT.

#### **2.5.6 Related Patterns**

This pattern is the logical opposite to the *Centralized Test Coordinator* test pattern described in section 2.6. It is also referred to as the *Centralized tester* test pattern [10].

#### **2.5.7 References**

[10]

---

## 2.6 Pattern: Centralized Test Coordinator for Concurrent Test Components

### 2.6.1 Context

- This pattern addresses test architecture design.
- This pattern is more applicable to integration and system testing. It is less the case for unit testing at the class level. However, it can be applied for system testing, whereby a unit testing framework is instrumented for that purpose.

### 2.6.2 Problem

How to model a test architecture, that is suitable for load- , performance- or conformance testing on an SUT requiring parallel and possibly distributed processing.

**Forces** The motivations for this pattern are:

- An SUT featuring concurrent behaviour cannot be verified through a test system supporting only sequential behaviour.
- Certain requirements of software (e.g. robustness, load, performance) can hardly be addressed using test architectures that allow only sequential behaviour.
- Simple test architectures (e.g. the *One-on-One test architecture pattern*) restrict the level of flexibility for the test system with regard to deployment. The fact that a distributed testing setup would not be possible is an example of those restrictions. A consequence of those restrictions is that certain test scenarios would not be possible.

However, this pattern comes with its liabilities that should be considered as well:

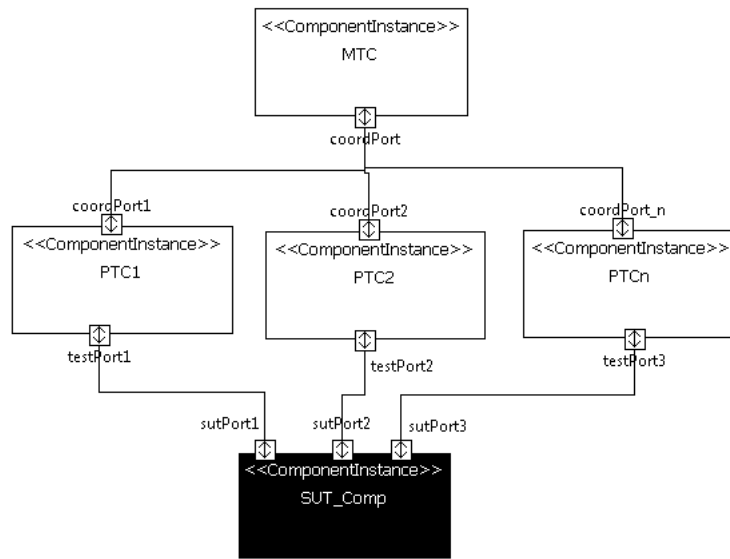
- It must be ensured that, despite the introduction of concurrency in the test system, the tests remain reproducible and deterministic.
- The introduction of concurrency will require some form of coordination between the entities involved. The effort for providing that coordination scheme should be taken into account as well.

### 2.6.3 Solution

As depicted on figure 3, this pattern features a test component acting as test coordinator and thus controlling the life cycle other components it controls. Each of the controlled test components is connected to the controlling component via a connection through which coordination messages can be exchanged to control the components' behaviour. To keep the overhead of processing those coordination messages as low as possible, to not affect the proper test behaviour, coordination messages should be kept as simple as possible in their structure. The real testing activities are performed by the controlled test components, which are directly connected to the SUT.

### 2.6.4 Known Uses

Several TTCN-3 projects such as [20] involving UTML protocol testing (Siemens) and [4] involving BCMP protocol performance testing.



**Figure 3. Test architecture Diagram for Centralized Test Coordinator Pattern**

### 2.6.5 Discussion

A coordination scheme is required between the main test component and the parallel test components to control the latter's behaviour according to the overall test choreography. The additional load and delays created by that communication should be taken into account while evaluating the SUT component's test results.

### 2.6.6 Related Patterns

This pattern is the opposite of the *One on One test architecture* pattern defined in section 2.5

### 2.6.7 References

[10]

## 3 Evaluation of the Approach: IMS Case Study

### 3.1 MDTester: A Pattern-Oriented Test Design Tool

Integrating patterns in a process requires a suitable concept, that would allow the creation of new artifacts in a flexible and efficient way, while at the same time ensuring that the rules defined by the patterns are followed in the creation process or can be verified afterwards. Domain Specific Modeling Languages (DSML) provide a good mean for integrating patterns to a given process. Firstly, because they operate at a level of expression, that is abstract enough to express concepts in a solution-independent, but yet formal manner. Secondly, because they can be tailored precisely to define model templates and associated rules, that are specific to the targeted process' domain. Therefore, to evaluate the impact of the patterns we presented in section 3, we defined a UML MOF Meta-model for a DSML dedicated to black box test engineering. The particularity of this DSML is that, tests are modelled based on meta-elements representing the patterns we mentioned earlier.



Test Pattern	Implementation Status	Application to Case Study
Separation of test design concerns	Yes	Yes
Grouping of concerns	Yes	Yes
Naming convention	Yes	Yes
Prioritization of test objectives	Yes	No
Traceability of test objectives to requirements	Yes	Yes
Traceability of test objectives to fault management	Yes	No
One on One test architecture	Yes	No
Proxy test component	Yes	Yes
Centralized test coordinator	Yes	Yes
Purpose-driven test data design	Yes	Yes
Flexible test data definition	Yes	Yes
Dynamic test data pool	No	No
Focus on expected test behaviour	Yes	Yes
Test component factory	No	No
Time constraints	Yes	Yes

**Table 1. Summary of Test Patterns and Status**

### 3.2 The IMS testing case study

Following an MDE process, we developed MDTester, a tool to support pattern-oriented test design with automated generation of test designs according to selected patterns and automated model transformations of the high-level test design into test scripting or test specification notation for specific target test environments (e.g. TTCN-3, JUnit). The MDTester tool was used to design functional tests for the IP Multimedia Subsystem (IMS) architecture. Table 1 lists all test patterns and their implementation status in the prototype tool, as well as their application to the IMS case study test model.

The impact of model-driven and pattern-oriented test development can be analyzed from a quantitative and a qualitative view point. The purpose of quantitative analysis is to evaluate how productivity is affected by the introduction of the methodology. On the other hand, qualitative analysis aims at measuring the effect on quality factors, both of the process itself and of its output, i.e. the generated test scripts. The goal of the case study was to analyse both the qualitative and the quantitative aspects of that impact and at the same time, to compare the results with those obtained with a “traditional” test development approach.

#### 3.2.1 Quantitative Analysis

A key metric for quantitative analysis of any development process is productivity. Evaluating the productivity of pattern oriented test development is a relatively straightforward task. For that purpose, we simply have to correlate the output (e.g. number of implemented test cases) to the invested effort (e.g. number of person-days/person-months involved) for a project or a series of projects. However, to measure the impact of introducing a new approach on that productivity is a less trivial task, because productivity data before and after the introduction of the new approach need to be compared with each other. Ideally, to ensure a fair comparison, at least the following conditions need to be fulfilled:

- 
- Both methodologies should be applied on the same case study: The starting point for both test development approaches should be the same system specification or test plan, targeting the same SUT
  - Separate teams should apply the methodology, each on its side in a separate project.
  - The same time frame will apply to both projects and results will be collected at the end for evaluation.
  - Both teams should have comparable level of expertise in their respective field.

However, we could not provide such an ideal setup for our IMS case study. Therefore we had to base our quantitative comparison on assumptions resulting from statistical analysis of past TTCN-3 test development projects.

Taking into account that the project duration was set to 5 person-days and that a total result of 19 test cases were implemented at its end, productivity factor is  $19/5 = 3.8$  test cases/day. It should be pointed that, this result was obtained with team of designers with a rather low level of testing and modelling expertise. Therefore, it can be assumed that slightly higher results would be obtained with experienced test designers.

To measure the productivity gain generated by our approach, we compare our results with those generally obtained through “traditional” test development approaches. Generally, for TTCN-3 test development, realistic estimations of productivity range between 2 and 5 test cases/day. The obtained results indicate that, if the existing process allows a production rate of more than 4 test cases/day (including test objectives definition, test procedure design and documentation), then applying our methodology would rather cause a productivity loss. On the other hand, the productivity could be significantly improved (30 to 90%), when the production rate of the existing methodology is between 2 and 4 test cases/day.

Moreover, if we estimate that, the specification of a test plan (test objectives) and of test procedures consumes 20% of the effort in pattern-oriented test development and are generally not taken into account, when estimating the productivity of the test development process, then the productivity gain is even higher.

### 3.2.2 Qualitative Analysis

Using model-driven approach to test development offers a wide range of qualitative benefits, compared to traditional development approach. Test models offer a higher level of readability, maintainability, documentation and flexibility than plain test scripts and non-formal notations. Furthermore, existing MDE frameworks (e.g. Eclipse EMF, TOPCASED) provide a wide range of functionalities for creating, managing, validating and transforming models, that can be used to provide powerful tool chains to support the process. However, a source of general concern is the quality of the test scripts generated automatically from the process. For our case study, we used the TRex [25] tool to measure the quality of the generated TTCN-3 test scripts. The authors of TRex define a metric called *Template coupling* (ranging between 1 and 3) to measure the maintainability of TTCN-3 scripts. The automatically generated IMS test scripts scored 1.015 on that metrics, indicating the high level of maintainability of those scripts (1.0 is best).

## 4 Related Works

The potential benefits of cataloguing best practices and patterns in test design has been advocated by several authors before. Binder [2] discusses a test pattern template, based on a pattern language of object oriented testing (PLOOT) proposed by Firesmith [11] and introduces a collection of test patterns from the object-oriented software design domain. Meszaros [18] presents a collection of test patterns for unit testing. Howden [14] presents a collection of patterns in selecting tests for maximum error detection. It appears that existing work on test patterns tend to focus on interactions at the object level and are hardly applicable for higher level (i.e. integration, system, and acceptance-level) testing whereby the applied programming paradigm are less relevant. Delano et al [3] present a collection of patterns focussing more on the organisational aspects of test development as a process, rather

---

than on test design itself. On the other hand, Dustin [7] covers all aspects of test development, with one chapter dedicated to test design and documentation. In 2005, the European Telecommunications Standards Institute (ETSI) started an initiative on patterns in test development (PTD) in which some of the patterns defined in this work were introduced and discussed. However, to the best of our knowledge, none of the existing work attempts to formalise test patterns, so that they could be instrumented to support the test development process in an automated way.

## 5 Conclusion and Outlooks

This paper has presented first ideas on a collection of patterns of test design based on a template defined for that purpose. First experiences with that prototype tool chain have shown some promising results. However, model-driven test development has not reached a high level of popularity yet. Therefore, some of the patterns described here can only be considered as mere candidates and will require further analysis with regard to their usability and their consequences. Also, we have presented a case study in which those patterns have been applied to develop tests for IMS. An analysis of the approach through that case study indicates that it can significantly improve the test process, both quantitatively and qualitatively. In the future, we intend to conduct further case studies to analyze the impact of the approach, when developing tests for other domains.

## 6 Acknowledgements

We would like to thank our shepherds Uwe Zdun and especially Christian Kohls (on-site shepherd) who were both very helpful in the process of improving this paper through their challenging comments, their patience as well as their interesting ideas. Also, we would like to thank all participants to writer's workshop E at the EuroPLoP 2009 conference for their contributions to bring this paper into shape. A special thank you to Dietmar (Didi) Schtz, Michael Kircher, Heiko Hashizume, klaus Marquardt and last but not least, Markus Voelter!

## References

- [1] Harold J. Batteram, Wim Hellenthal, Willem A. Romijn, Andreas Hoffmann, Axel Rennoch, and Alain Vouffo. Implementation of an open source toolset for ccm components and systems testing. In Roland Groz and Robert M. Hierons, editors, *TestCom*, volume 2978 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2004. [cited at p. 11]
- [2] Robert V. Binder. *Testing Object Oriented Systems: Models, Patterns and Tools*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999. [cited at p. 3, 15]
- [3] David E. Delano and Linda Rising. System test pattern language copyright 1996 ag communication systems corporation permission is granted to make copies for plop '96., 1996. [cited at p. 15]
- [4] Sarolta Dibuz, Tibor Szabó, and Zsolt Torpis. Bcmp performance test with ttcn-3 mobile node emulator. In *TestCom*, pages 50–59, 2004. [cited at p. 12]
- [5] George Din. An ims performance benchmark implementation based on the ttcn-3 language. *Int. J. Softw. Tools Technol. Transf.*, 10(4):359–370, 2008. [cited at p. 11]
- [6] Hyunsook Do, Gregg Rothermel, and Alex Kinner. Prioritizing junit test cases: An empirical assessment and cost-benefits analysis. *Empirical Softw. Engg.*, 11(1):33–70, 2006. [cited at p. 7]
- [7] E. Dustin. *Effective Software Testing. 50 Specific Way to Improve Your Testing*. Addison-Wesley, 2003. [cited at p. 6, 16]
- [8] Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Test case prioritization: A family of empirical studies. *IEEE Trans. Softw. Engg.*, 28(2):159–182, 2002. [cited at p. 7]
- [9] Sebastian Elbaum, Gregg Rothermel, Satya Kanduri, and Alexey G. Malishevsky. Selecting a cost-effective test case prioritization technique. *Software Quality Control*, 12(3):185–210, 2004. [cited at p. 7]

- 
- [10] M. Frey et al. Etsi draft report: Methods for testing and specification (mts); patterns for test development (ptd). Technical report, European Telecommunications Standards Institute (ETSI), 2005. [cited at p. 11, 13]
- [11] D.G. Firesmith. Pattern language for testing object-oriented software. *Object Magazin*, 1996. [cited at p. 15]
- [12] Methods for Testing and Specification (MTS). The testing and test control notation version 3; part1: Ttcn-3 core language. Technical report, European Telecommunications Standards Institute (ETSI), 2003. [cited at p. 1]
- [13] Lars Hagge and Kathrin Lappe. Sharing requirements engineering experience using patterns. *IEEE Software*, 22:24–31, 2005. [cited at p. 2]
- [14] William E. Howden. Software test selection patterns and elusive bugs. In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 1*, pages 25–32, Washington, DC, USA, 2005. IEEE Computer Society. [cited at p. 3, 15]
- [15] Andy Hunt and Dave Thomas. *Pragmatic Unit Testing in Java with JUnit*. The Pragmatic Programmers, September 2003. [cited at p. 1]
- [16] IEEE. Draft ieee standard for software and system test documentation (revision of ieee 829-1998). Technical report, IEEE, 2008. [cited at p. 2, 7]
- [17] ISO/IEC. Iso/iec standard no. 9126: Software engineering product quality; parts 14. Technical report, Organization for Standardization (ISO) / International Electrotechnical Commission (IEC), Geneva, Switzerland, 2001-2004. [cited at p. 3]
- [18] Gerard Meszaros. *XUnit Test Patterns: Refactoring Test Code*. Addison-Wesley, 2007. [cited at p. 15]
- [19] Helmut Neukirchen. *Languages, Tools and Patterns for the Specification of Distributed Real-Time Tests*. PhD thesis, Dissertation, Universität Göttingen, November 2004 (electronically published on <http://webdoc.sub.gwdg.de/diss/2004/neukirchen/index.html> and archived on <http://deposit.ddb.de/cgi-bin/dokserv?idn=974026611> . Persistent Identifier: urn:nbn:de:gbv:7-webdoc-300-2), November 2004. [cited at p. 3]
- [20] Andrej Pietschker. Automating test automation. *Int. J. Softw. Tools Technol. Transf.*, 10(4):291–295, 2008. [cited at p. 12]
- [21] OMG ptc. Unified modeling language: Testing profile, finalized specification. Technical report, Object Management Group, 2004. [cited at p. 1]
- [22] Stephan Schulz. Test suite development with ttcn-3 libraries. *Int. J. Softw. Tools Technol. Transf.*, 10(4):327–336, 2008. [cited at p. 5, 11]
- [23] Alain Vouffo-Feudjio and Ina Schieferdecker. Test patterns with ttcn-3. In *FATES*, pages 170–179, 2004. [cited at p. 1]
- [24] Stefan Winkler and Jens von Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling*, December 2009. [cited at p. 8]
- [25] Benjamin Zeiß, Helmut Neukirchen, Jens Grabowski, Dominic Evans, and Paul Baker. TRex - An Open-Source Tool for Quality Assurance of TTCN-3 Test Suites. In *Proceedings of CONQUEST 2006 – 9th International Conference on Quality Engineering in Software Technology, September 27-29, Berlin, Germany*. dpunkt.Verlag, Heidelberg, September 2006. [cited at p. 15]
- [26] Benjamin Zeiß, Diana Vega, Ina Schieferdecker, Helmut Neukirchen, and Jens Grabowski. Applying the ISO 9126 Quality Model to Test Specifications – Exemplified for TTCN-3 Test Specifications. In *Software Engineering 2007 (SE 2007). Lecture Notes in Informatics (LNI) 105. Copyright Gesellschaft für Informatik*, pages 231–242. Köllen Verlag, Bonn, March 2007. [cited at p. 3]

# A Good Fort Has a Gap

Arto Juhola  
VTT Technical Research Centre of Finland

17. December 2009



Figure 1: Old painting of Himeji castle [24][21]

## 1 Name

“A good Fort Has a Gap”.

## 2 Problem

*You are not (and can't be) absolutely certain that the principal defences (of a system or a structure) you have set up are impenetrable.*

In other words, you cannot out rule the possibility of a successful break-in, since the attacker might be able to circumvent or confuse the provided access control points by some clever means. This might be due to the breadth and complexity of the system or structure, or to some other inherent weakness.

## 3 Context

You have to defend a site with a well defined perimeter, be that a physical fort, a network or something else.

You have taken care of the more straightforward means of defence. But, confidence in the infallibility of the principal defence might be misplaced, rendering your system or structure to be just a little better off than a sitting duck; You have to cover all possible vulnerabilities, while the attacker needs only a single one.

## 4 Forces.

1. The site cannot parallel the kind of imperviousness a European, pre-cannon era (1100-1300) castle had against the attackers of the period, when the castle defenders usually could most often just simply wait until the attackers lost heart and returned home[22].
2. You lack applicable methods, resources, manpower, funds and/or time to realise an impenetrable protection, or it is a sheer impossibility.
3. However, you have authority and means, within reasonable bounds, to introduce improvements.
4. Deploying improved (but not perfect) protection prompts intruders to devise and use more intricate forms of attack, including ones not foreseen and subsequently lacking means of detection.
5. To make things worse, potential malefactors have often many advantages over you that make foreseeing attacks difficult: Covert collection of preparatory information, gathering of “mass”, concealed identity, concentration of forces, surprise, freedom of means, particularly with regard to law, latitude in manoeuvres and timing, and possibilities to cover up their tracks.
6. Because of the above disadvantages, it might be necessary to diminish the extent of the “fort’s” perimeter”
7. The manpower you have is capable, disciplined and ample.
8. Awareness of any stealthy, on-going break-in attempt is essential in the utmost, since if continuing unnoticed it might eventually lead to the compromising of the “fort”. Also, the sooner the attempt will be noticed the sooner the actions to neutralise it can be started, and consequently, the lesser the damages will be.

## 5 Solution

*Your system or structure should have an intentional weakness (real or fake), visible to the outside. This way you can lure attackers to enter in a way you can predict and detect.*

Behind the “gap”, you are able to select and set the “stage” to your liking, watch it over with the arsenal of your choice, and persuade anyone paying a visit to dance according to your tune.

### 5.1 Caveat

The presented pattern is complementary to the principal defences, not a substitute. Its worth is in preparing for the attack to come; a stratagem to make the intruder to reveal himself, his resources, methods and intentions. The pattern is useful for cases where watertight protection cannot be achieved. In such cases, however :

- It does not remove the need to have the other means to counter whatever havoc the malefactor has in mind to play.
- The protection will be improved, but it still will not be unshakable.
- Presenting a weakness can direct the attack towards it; care will be needed to avoid this weakness to be successfully exploited by the attackers.

## 5.2 Application

### 5.2.1 Generic

An aspect facilitating the universal use of the pattern is that every site, however strong, does have a weakest point, as exemplified by the European medieval castles mentioned earlier; with them it was the gate, but even this weak point was very strong indeed [23]).

Also, “To catch a thief, you must think like a thief”:

First, to make the behaviour of your unwanted guest a little more predictable, you must step into the “boots” of the potential malefactors. You must take care to present something that will hopefully suit their appetite, but nothing too obvious, since a sophisticated intruder might be alerted when encountering a “too easy” target (In computer setting, a “low hanging fruit”), and decide to resort to something more intricate. Also, after a break-in, it should not be easily deductible that a trap is closing.

There are at least the following possibilities:

- Determine existing weaknesses in the defences. In case that there is no cure available (yet) these should be definitely taken in account.
- Open your defences slightly, so as not to arouse suspicion, and at a select point, so that you will know in advance the likely ingress point of an attack. This is an obviously an option that requires preparedness.
- In addition, the “gap” can be a fake. Say, by reporting misinformation about the “fort”; like location of an opening where no exists can entice an attack devised to make use of it, leading to an easy detection (with computers e.g. a wrong version of a certain software, one known to present a particularly popular vulnerability)

The presented “gap” should be irresistible to the intruder, yet easy to guard. Usually, an intruder hopes to gain the control of some central part, to allow him to investigate the surroundings and to gain foothold in as many other parts as possible. The ones which seem to be the “hubs” of any kind of traffic are natural picks.

Creating “a gap” means lowering your defences or tolerating some known vulnerabilities. And/or at least (falsely) exhibiting them. This means that there will be a certain element of risk involved, since malevolent activity will be allowed some leeway. It is up to the defender to see that he can keep his insidious guests within bounds.

The presented solution requires that the “fort” needs to be well armed and manned; in the handling of the unprecedented no automatic mechanisms has proven to be satisfactory (so far). In other words, the pattern requires substantial effort to deploy, maintain and operate.

Finally, it cannot be stressed too much that the other, principal security mechanisms must be in top condition, the “gap” being just an addition.

### 5.2.2 Computers

Generally, the “first line” of defence cannot be trusted to withhold a sufficiently determined attack. For example, in network security the defence is based on the models of attacks known beforehand, or detection of anomalies. So it is within possibilities that novel variants of attacks could appear, not deviating too much from the normal traffic to cause an anomaly alarm to be raised. Also, defence might fail by misconfiguration and too numerous “false positives”, the filtering of which can throw some real alerts aside as well.

The “gap” can be presented in a special “bait” machine, external to the production system, as is the case with most “honeypots”. A machine can be too obviously a “low hanging fruit”, though. A double bluff is also a possibility: A production system truly residing in an overtly tempting looking host (but still well guarded), and a somewhat less enticing target offered by some nearby machine.

Second, be aware of the prime targets of the “black hats” potentially assaulting your system.

Guard all the above keenly, but opaquely.

If a malefactor gains an access into the system through your “gap”, try to confuse him; say, let him believe that he’s in a honeypot. Or present a “forced deal” to him (e.g. “honeytokens” [11]). When deliberately offering a target you must thread carefully so that the systems intended to protect you will not be made to turn against you, or if this happens, they should not be able to cause excessive harm. This is a very real threat, since a takeover of the protective systems is the dream of a cracker.

Although a bait would be a bit too evident for an intruder, his less sophisticated colleagues will still be trapped, though, and if there is no cure for his next move, nothing additional will be lost (the system will be lost with or without implementing this pattern). Also, the “black hat” might think it’s a case of a “double bluff”.

So, while in operation, a good decoy should seem to be a realistic one, with authentic looking traffic. As such, offering and watching over “baits” is not something that a small site could consider, so a balance needs to be stricken between the needed effort and the risks (risk = probability of an incident multiplied by its impact).

Collect incriminating, non-repudiable evidence. Logging time-stamped security events to a non-rewritable medium is a good start.

## **5.3 Known Uses**

### **5.3.1 Discovery of the dtspcd exploit**

First documented capture of an unknown exploit with a “Honeypot” (from the “HoneyNet” project [10], declared as the first in [16]), was noted in CERT advisory CA-2002-01[14]

In this document the network traces provided by the HoneyNet Project were specified as the source material that revealed the active exploitation of the dtspcd vulnerability.

The vulnerability in question was a remotely exploitable buffer overflow, affecting the Common Desktop Environment’s (CDE, a graphical user interface on \*nix) Subprocess Control Service network daemon (dtspcd) that accepts requests from clients to execute commands and launch applications remotely.

The details of the capture of the exploit, including the actions of the intruder, can be found in [13].

In short, the traffic of the Honeypot, a machine with SunOS 5.8, was registered with the Ethereal tool and the attack itself followed the usual procession of reconnaissance -> exploit -> reinforcement (bring in the rootkit)-> consolidation (use newly installed back-doors for further communications) as outlined in [15] .

### **5.3.2 Project Honey Pot**

Project Honey Pot [17] is an on-going and operational (at the time of writing, March 2009) effort to snare e-mail address harvesters, spammers, dictionary attackers and comment spammers [18]. “Harvesters” are automated collectors of addresses from web-pages, “dictionary attackers” are in search of e-mail addresses by mailing to potential user names and waiting for possible acknowledgements, and “comment spammers” are targeting blogs and forums. The idea of the Honey Pot is to present decoy e-mail addresses and html forms in www-pages. When these (unique) e-mail addresses are “harvested”, the “harvester’s” IP-address will be stored with the harvested address. Immediately after this the decoy e-mail address will be changed. The arrangement facilitates later correlation of the harvesters IP-address with the email sent by a spammer to this address (the spammer and harvester will have most probably a “subcontractor” relationship, and therefore a different IP address).

An noteworthy aspect of the activity is that the system is global, orchestrated by the Project Honey Pot that publishes statistics and results, cooperates with law enforcement officials and accepts donations. The donations might be e.g. MX records that can be used in decoys and www-pages in servers along with plain funding.



### 5.3.3 Tom Liston's "LaBrea" Tar Pit vs. "CodeRed" worm

The "CodeRed" worm [20] was observed July 13 2001 and Tom Liston's site was one of the victims. He came to think about ways to slow it down, and settled on offering bogus machines to the worms entering reconnaissance phase, that is, starting to find new victim machines by scanning. The bogus machines were announced at the communication protocol level (starting with ARP and ending with TCP) to any request to connect to an address with no machine attached. Since the protocols at the worm's side were persistent about opening a connection to the new found "virtual machine", substantial delays could be effected by careful choice and timing of the protocol replies. When the worm finally moved on (to the next address), the same procedure could be repeated. [19]. The fruition of this idea was "LaBrea" [1].

## 5.4 Exemplary cases

The pattern has a wide area of application:

- You might run a networked site, with connections available for external traffic. Most often you have to provide services (like web) with an indeterminate set of external users/machines so a VPN (Virtual Private Network) solution is not applicable. So, you need to fend against the threat from the "outside". As for the remedy, the solution might include "Honeypots", "Tarpits" and Logging arrangements, see 5.5
- Another case could be a user interface provided for the public, say a desktop system in libraries, embedded systems for payments like at petrol stations or ATMs (Automatic Teller Machines). A potential intruder might try to sneak his/hers way around the official user interface. In this case the intruder might be "rewarded" for his persistence, by having sufficiently hidden items like "system control" or "authorised users only" to be available to him or her, triggering appropriate alarms and video cameras etc. as a matter of course. The items should have varying (preferably random) discovery "paths" and announcements to prevent their learning.
- Also within singular applications, like databases, users apt for unauthorised access need to be trapped. Here suitable "Honeytokens" [11], e.g. extra tables with tempting looking content (fake credit card numbers etc.) but with access announced as illicit to ordinary users might be the answer.
- The pattern applies beyond computerised systems, say to physical buildings where an attempt needs to be made to oversee the comings and goings of people. In buildings, a special, unlocked door, with alarms, and with a sign like "Cashiers office, for personnel only" could do the trick,
- And, if you ever find yourself in a situation calling for the planning of the fortifications for a village, in circumstances resembling medieval Japan, you can also take the name of the pattern quite literally (see 5.5)

## 5.5 Suggested Further Reading

### 5.5.1 On Honey Things

**Honeypots:** Usually thought to be separate machines, but the idea fits to separate entities within machines as well (processes, virtual machines). These are not so effective anymore, since these can be detected [3][2], even easily[5], since they need to abide with the law and it is trivial to test this. Simulated "unlawful" environments might be an improvement as proposed in [5], but are not likely to be an answer since the attackers can test against several known (to them) domains, and not all of them can be fakes. Undoubtedly, the (counter) detection methods will become more sophisticated and/or inventive with time. Aside the simulated unlawfulness, the "honeypots" need to be made to resemble normal systems as closely as possible, including fake traffic with normal looking traffic patterns.

**Honeytokens:** Associated with honeypots are “honeytokens”, i.e. a piece of information used as a bait, essentially something that the intruder can “take away” and that can act after its capture as a means of tracking of the actions taken by the intruder. As noted in [11], the honeytoken can be made to incorporate also the aspect of “agent provocateur”, i.e. to contain something capable of enticing its capturer to commit subsequent, incriminating actions (say, an e-mail address, with a promise of extra information, given in the honeytoken only, for the “special guest”, could tempt the “black hat” actually to send a message and thus reveal himself).

**Honeynets:** Comprise of several connected honeypots, see [10]

**Honeymonkey:** Of Microsoft origin [7], the idea is to actively invite marauders into your system. Honeymonkey (there are also open source efforts, “honeyclient” and “HoneyC[8][9]) contacts websites, downloads their content and checks afterwards whether any “monkey business” (or worse) was perpetrated.

### 5.5.2 On Parallel Approaches

**Tarpits:** “Tarpits” [1] are based on the idea of slowing down sessions to deter intruders. In case of “LaBrea”[1] the main idea is to offer empty promises of hosts to contact, kind of “virtual” honeypots, so that the assailants attempts to reach the non-existent will waste his/hers time.

**\*nix syslog:** The logs can be periodically and silently (unseen by the rest of the system) stored in a secured place for (sufficiently frequent) periodic comparison. Logs being a valued target, attempts to cover traces of illicit activity are thus easily detected. This idea is described in [3], and its nature is a “last resort” affair, since it implies that all preceding safeguards have failed.

## 6 Resulting Context

A warning has been received and the “cat and mouse” play between the intruder and the admin may begin.

Since it is increasingly difficult to get any reliable information about the intruders’ ultimate traffic source (series of “bots” used for mediating), the malefactor should be kept unaware of his/hers exposure, so that useful information about the “modus operandi” can be accumulated. This, and not the simple act of filtering packets based on source addresses, can be used to prevent any subsequent similar attempts in the future.

There are three main cases:

1. The “mouse” continues to believe in the hoax, and after performing whatever it had in its mind, leaves. In a lucky case it might further contribute to its downfall, say, in a computer setting, a “honeytoken” arouses its interest so it scurries away with it between it’s teeth.
2. After a while, the “mouse” begins to detect that something is amiss, retreats, maybe after an attempt to conceal any telltale signs, and lists the “gap” as intentional. Perhaps it distributes this information to its kind.
3. The “mouse” was specifically in search of defence related arrangements, or was otherwise prepared to handle the situation, and is determined to make use of them.

As for the “cat”, in the first case the follow-up actions are to analyse the evidence left behind to learn the tricks of the intruder, implement any necessary improvements, and if appropriate, wait for “hooks” in a possibly swallowed honeytoken to catch, or the honeytoken itself to show up.

The second case will lead to a need for further actions, since the plot has been revealed. The attacker might ponder returning with new vigour at some future date, or some of the kindred souls that have received or bought the information might pay a visit, avoiding or targeting the “gap” as they wish.

With the third case great care is needed, to avoid the bait to turn against its masters. If the going gets touch the gatecrasher can be forcibly expelled to avoid him or her gaining any foothold. In any of the cases, after an noticed attack attempt, thorough checking of the relevant systems is commendable.

## 7 Origins of the idea

The basic insight has ancient roots; as it happens, the principle was used in medieval Japanese warfare. According to Karl F. Friday, a professor in Japanese history, the early fortifications (12th - early 13th century) ubiquitously employed wooden gates, *kido* or *kidoguchi*, designed to act as focal points of a battle, since they were the only conceivable entrances to the attackers (as well as a means of counterattack to the defenders) of the time [12].

The idea is also depicted in the Kurosawa’s famous film “The Seven Samurai” (1954). The name chosen for this pattern is taken from a line of one of the main characters of that movie, in a scene where the construction of the defences for a village threatened by marauders is pondered: “A good fort needs a gap. The enemy must be lured in. So we can attack them. If we only defend, we lose the war.”[6].

Also, Japanese castle town’s layout often featured winding streets and blind alleys designed to delay and deceive the attackers [21] .

## 8 Acknowledgements

It was Mr Juha Pärssinen who suggested to me to write some patterns, an exercise I have not tried before. While I have been aware of patterns and have read a bunch of them, it was his enthusiasm and advice that prompted me to produce the current text. I thereby present my thanks to Juha for guiding me in the world of patterns.

I also owe thanks to my “shepherd”, Kristian Elof Sørensen for many valuable suggestions concerning the form and the content of the pattern. And, I am gratefull for Tim Wellhausen, Ville Reijonen, Veli-Pekka Eloranta, Marko Leppänen, Anjali Das, Farah N. Lakhani, Martin Wagner and Stefan Sobernig for offering many heplfull comments during the EuroPloP-09 conference.

## 9 Copyright

Copyright retain by author(s). Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.

## References

- [1] A tarpit: <http://labrea.sourceforge.net/labrea-info.html> (retrieved 13. Nov 2009)
- [2] <http://www.send-safe.com/honeypot-hunter.html> (retrieved 13. Nov 2009)
- [3] Anti-honeypot technology, Security & Privacy Magazine IEEE, <http://ieeexplore.ieee.org/ie15/8013/28290/01264861.pdf?arnumber=1264861> (retrieved 13. Nov 2009)
- [4] On logging [http://www.sans.org/reading\\_room/whitepapers/logging/1168.php](http://www.sans.org/reading_room/whitepapers/logging/1168.php) (retrieved 13. Nov 2009)
- [5] Detecting honeypots, the easy way <http://www.springerlink.com/content/dq3t9btumw6gvrk5/> (retrieved 13. Nov 2009)

- [6] [http://en.wikiquote.org/wiki/The\\_Seven\\_Samurai](http://en.wikiquote.org/wiki/The_Seven_Samurai) (retrieved 13. Nov 2009)
- [7] <ftp://ftp.research.microsoft.com/pub/tr/TR-2005-72.pdf> (retrieved 13. Nov 2009)
- [8] <http://www.honeyclient.org/trac> (retrieved 13. Nov 2009)
- [9] <https://www.client-honeynet.org/> (retrieved 3. Feb 2009)
- [10] <https://www.honeynet.org/> (retrieved 13. Nov 2009)
- [11] <http://www.securityfocus.com/infocus/1713> (retrieved 13. Nov 2009)
- [12] Samurai, Warfare and the State in Early Medieval Japan, by Karl F. Friday, Published by Routledge, 2004, ISBN 0415329620, 9780415329620 (Chapter “Fortifications and Strongholds”, page 125).
- [13] <http://old.honeynet.org/scans/scan28/sol/29/sotm28.pdf> (retrieved 13. Nov 2009)
- [14] <http://www.cert.org/advisories/CA-2002-01.html> (retrieved 13 Nov 2009)
- [15] Richard Bejtlich, The Tao of Network Security Monitoring: Beyond Intrusion Detection, ISBN-10: 0-321-24677-2, ISBN-13: 978-0-321-24677-6
- [16] Honeypots: Tracking Hackers, by Lance Spitzner, Publisher: Addison-Wesley Professional, Pub Date: September 10, 2002, ISBN-10: 0-321-10895-7, ISBN-13: 978-0-321-10895-1
- [17] <http://projecthoneypot.org/index.php>
- [18] <http://projecthoneypot.org/faq.php> (retrieved 13. Nov 2009)
- [19] <http://labrea.sourceforge.net/Intro-History.html> (retrieved 13. Nov 2009)
- [20] [http://en.wikipedia.org/wiki/Code\\_Red\\_\(computer\\_worm\)](http://en.wikipedia.org/wiki/Code_Red_(computer_worm)) (retrieved 13. Nov 2009)
- [21] [http://en.wikipedia.org/wiki/Japanese\\_castle#Layout](http://en.wikipedia.org/wiki/Japanese_castle#Layout) (retrieved 13. Nov 2009)
- [22] Jurgen Brauer and Hubert van Tuyll: Castles, Battles, and Bombs: How Economics Explains Military History! 2008, ISBN: 978-0-226-07163-3 (ISBN-10: 0-226-07163-4), pages 45–66
- [23] <http://en.wikipedia.org/wiki/Castle#Gatehouse> (retrieved 13. Nov 2009)
- [24] [http://upload.wikimedia.org/wikipedia/commons/a/a9/Old\\_painting\\_of\\_Himeji\\_castle.jpg](http://upload.wikimedia.org/wikipedia/commons/a/a9/Old_painting_of_Himeji_castle.jpg) (retrieved 13. Nov 2009)

# **Towards a Pattern Language which Supports the Migration of Systems from Event-Triggered Pre-emptive (ETP) to Time-Triggered Co-operative (TTC) Software Architectures**

**Farah N. Lakhani<sup>1</sup>, Anjali Das<sup>2</sup> and Michael J. Pont<sup>1</sup>**

*<sup>1</sup> Embedded Systems Laboratory, University of Leicester,  
University Road, LEICESTER LE1 7RH, UK.*

*<sup>2</sup> TTE Systems Ltd,  
106 New Walk, LEICESTER LE1 7EA,, UK.*

`fn11@le.ac.uk; a.das@tte-systems.com; M.Pont@le.ac.uk`

## **Abstract**

We have previously described a “language” consisting of more than seventy patterns. This language is intended to support the development of reliable embedded systems: the particular focus of the collection is on systems with a time triggered (TT) system architecture.

The present paper has a focus on techniques for converting an event-triggered (ET) system to an equivalent TT system.

The introduction to this paper describes the approach that we have taken to migrate from an ET to a TT architecture, and the motivation for making such a change. The core of the paper then goes on to describe some new patterns which represent the first parts of what is intended to be a small pattern collection.

## **Acknowledgements**

We are grateful Bob Hanmer, to our Shepherd at EuroPLoP 2009, for his numerous helpful comments on this paper (and for his patience).

This work is supported by an HEC scholarship awarded to Farah N Lakhani from the Pakistan government. Additional support is provided by TTE Systems Ltd.

## **Copyright**

Copyright retained by Farah N. Lakhani, Anjali Das and Michael J. Pont. Permission is granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for publication on the Hillside Europe website.

# Introduction

We have previously described a “language” consisting of more than seventy patterns, which will be referred to here as the “PTTES Collection” (see Pont, 2001). This language is intended to support the development of reliable embedded systems: the particular focus of the collection is on systems with a time triggered (TT) system architecture. Work began on these patterns in 1996, and they have since been used in a range of industrial systems, numerous university research projects (e.g. see Kurian and Pont, 2005; Phatrapornnant and Pont, 2006; Short and Pont, 2007; Bautista-Quintero and Pont, 2008; Gendy and Pont, 2008; Hughes and Pont, 2008) as well as in undergraduate and postgraduate teaching on many university courses.

Our focus throughout this work has been on systems with a “time-triggered architecture”, the main alternative to which is an “event triggered architecture”. We distinguish between these two alternatives in as shown in Figure 1 below:

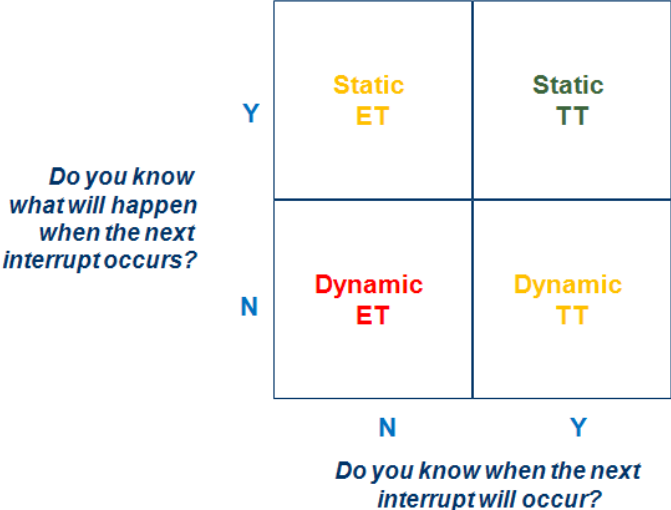


Figure 1: One way of distinguishing between time-triggered (TT) and event-triggered (ET) software architectures in a system design.

We assume that - in a static TT system - we always know (i) when the next interrupt will occur, and (ii) exactly what the system will do in response to this interrupt. At the other extreme, we have dynamic ET systems: in such designs we assume that (i) we never know when the next interrupt will occur, and (ii) that we do not know exactly what the system will do in response to this interrupt. In reality, many systems lie somewhere between these two extremes.

## When and why should you migrate?

Our underlying assumption is that, in most cases, making a system “more TT” is likely to make it easier to predict how the system will behave and will – therefore – improve reliability.

If you'd like further information about the differences in behaviour between ET and TT systems, Short et al, 2008 describe, in detail, the results from a 4-year study which compared ET and TT architectures when used in a multi-processor automotive system. In summary, the results demonstrated that ET systems had a greater failure rate. For systems which are not safety related (for example, simple consumer products), we find that the greatest single benefit obtained through the use of TT architectures is a reduction in testing times.

## Organisation of this paper

We have structured this paper in the form of a new (abstract) pattern (EVENTS TO TIME (TTC)), two new design patterns (BUFFERED OUTPUT and POLLED INPUT), plus two pattern implementation examples (RS 232 DATA TRANSFER and SWITCH INTERFACE).

Please note that EVENTS TO TIME (TTC) describes conversion to one possible TT architecture ("TTC") provides a single tasking system architecture. In the future we will explore conversions to a range of different TT architectures.

## References

- Bautista-Quintero, R. and Pont, M.J. (2008) "Implementation of H-infinity control algorithms for sensor-constrained mechatronic systems using low-cost microcontrollers", *IEEE Transactions on Industrial Informatics*, 16(4): 175-184.
- Gendy, A.K. and Pont, M.J. (2008) "Automatically configuring time-triggered schedulers for use with resource-constrained, single-processor embedded systems", *IEEE Transactions on Industrial Informatics*, 4(1): 37-46.
- Hughes, Z.M. and Pont, M.J. (2008) "Reducing the impact of task overruns in resource-constrained embedded systems in which a time-triggered software architecture is employed", *Trans Institute of Measurement and Control*, 30(5): 427-450.
- Kurian, S and Pont, M.J. (2005) "Building Reliable embedded systems using abstract patterns, patterns and pattern implementation examples", Proceedings of the second UK embedded forum (Birmingham UK 2005) pp. 36-59. Published by University of Newcastle upon Tyne [ISBN: 0-7017-0191-9].
- Phatrapornnant, T. and Pont, M.J. (2006) "Reducing jitter in embedded systems employing a time-triggered software architecture and dynamic voltage scaling", *IEEE Transactions on Computers*, 55(2): 113-124.
- Pont, M.J. (2001) "Patterns for Time-Triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers", Addison-Wesley / ACM Press. ISBN: 0-201-331381.
- Short, M.J and Pont, M.J. (2007) "Fault-tolerant time-triggered communication using CAN" *IEEE Transactions on Industrial Informatics*, 3(2):131-142
- Short, M.J, Pont, M.J and Fang, Jiangzhong. (2008) "Assessment of performance and dependability in embedded control systems: methodology and case study" *Control Engineering Practice*, Vol. 16, pp. 1293-1307, July 2008

---

## EVENTS TO TIME (TTC)

---

{abstract pattern}

### Context

- You and / or your development team have programming or design experience with “event-triggered and / or pre-emptive” (ET/P) system architectures: that is, architectures which may involve use of conventional real-time operating system (RTOS) and / or multiple interrupt-service routines (linked to different interrupt sources) and / or task pre-emption.
- You are in the process of creating or upgrading an embedded system, based on a single processor.
- You already have at least a design or prototype for your system based on some form of ET/P architecture.
- Because predictable and highly-reliable system operation is a key design requirement, you have opted to employ a “time-triggered co-operative” (TTC) system architecture in your system, if this proves practical.

### Problem

How can you convert event triggered / pre-emptive designs and code (and mindsets) to allow effective use of a TTC scheduler as the basis of your embedded system?

### Background

If we were forced to sum up the difference between “embedded” and “desktop” systems in a single word we’d say “interrupts”.

Event triggered behaviour in systems is usually achieved through the use of such interrupts. The system is designed to handle interrupts associated with a range of sources (e.g. switch inputs, CAN interface, RS-232, analogue inputs, etc). Each interrupt source will have an associated priority. Each interrupt source will also require the creation of a corresponding “interrupt service routine” (ISR): this can be viewed as a short task which is triggered “immediately” when the corresponding interrupt is generated.

Creating such (ET/P) systems is – on the surface at least – straightforward. However, challenges often begin to arise (in non-trivial designs) at the testing stage. It is generally impossible to determine what state the system will be in when any interrupt occurs, which makes comprehensive testing almost impossible.

A time-triggered system also requires an understanding of interrupts, but the operation is fundamentally different. Of particular concern in this pattern is a “time-triggered co-operative” (TTC) architecture which supports single- tasking system. In such systems only one task is active at any point in time: this task runs to completion and then return control to the scheduler. At the heart of a TTC system is a cooperative scheduler which determines when the tasks in the system will be executed: once the tasks start running they “run to



completion”: they cannot be pre-empted by another task. In such a system, there is only a single interrupt source (usually a periodic timer “tick”): this is used to drive the scheduler.

## Solution

Here’s what you need to do to migrate to a TTC design:

- You need to ensure that only a single – periodic - timer interrupt is enabled (all other interrupt sources will be converted to flags, which will be polled as required).
- You have to determine an appropriate “tick interval” for your system (that is, you need to determine how frequently the timer interrupts need to take place).
- You have to convert any ET ISRs into periodic tasks and add these to the schedule.
- You need to deal – cleanly – with any “long tasks” (that is, tasks which may have an execution time greater than your chosen tick interval).

To illustrate part of the translation process consider a simple ET system running two tasks, X and Y. These tasks are invoked by separate interrupts and implemented by associated ISRs. Table 1 illustrates one alternative TTC system, implemented using a standard TTC scheduler (Pont, 2001).

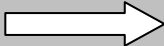
Events		Time
<pre>void main(void) {   X_init();   Y_init();   EA = 1 ; // Enable all interrupts    while(1)   {     PCON  = 0x01;   } }  void X_ISR(void) interrupt IEIndex { }  void Y_ISR(void) interrupt IEIndex { }</pre>		<pre>void main(void) {   SCH_Init(); // Set up the scheduler   X_Init();   Y_Init();    // Add tasks to scheduler   SCH_Add_Task(X_Update(), 0, 100);   SCH_Add_Task(Y_Update(), 20, 200);    // Start the scheduler   SCH_Start();    while(1)   {     SCH_Dispatch_Tasks();   } }</pre>

Table 1: Converting a system from Event triggered pre-emptive to Time triggered Cooperative

### *Dealing with long tasks*

To deal with the problem of long task, we need to find an elegant way of splitting up long tasks (which are called infrequently) into a series of much shorter tasks (called frequently). The pattern BUFFERED OUTPUT [later in this paper] describes a solution to this problem.

As “TTC” allows only single interrupt to be enabled in the system, we need to convert all other interrupt sources to flags, and we need to “poll” (that is, check periodically) to see if these flags have been set. The pattern POLLED INPUT [later in this paper] describes how we can achieve this.

## **Related patterns and alternative solutions**

### *The PTTES collection*

The PTTES collection (Pont, 2001) describes, in detail, a range of techniques which can be used to implement embedded systems with TTC architecture. This book can now be downloaded (free of charge) from the following WWW site:

<http://www.tte-systems.com/books/pttes>

### *TT Schedulers*

The pattern TT SCHEDULER\* provides relevant background information on tasks, basic scheduling concepts and the situations in which it may be appropriate to use a TTC scheduler in your application.

## **Reliability and safety implications**

When compared to pre-emptive schedulers, co-operative schedulers have a number of desirable features, particularly for use in safety-related systems (Allworth, 1981; Ward, 1991; Nissanke, 1997; Bate, 2000).

For example, Nissanke (1997, p. 237) notes: “[Pre-emptive] schedules carry greater runtime overheads because of the need for context switching—storage and retrieval of partially computed results. [Co-operative] algorithms do not incur such overheads. Other advantages of [co-operative] algorithms include their better understandability, greater predictability, ease of testing and their inherent capability for guaranteeing exclusive access to any shared resource or data”.

Allworth (1981, pp. 53–54) also notes: “Significant advantages are obtained when using this [co-operative] technique. Since the processes are not interruptable, poor synchronisation does not give rise to the problem of shared data. Shared subroutines can be implemented without producing re-entrant code or implementing lock and unlock mechanisms”.

---

\* Wang, H., M. J. Pont, et al. (2007). Patterns which help to avoid conflicts over share resources in time-triggered embedded systems which employ a pre-emptive schedule the 12th European Conference on Pattern Languages of Programs (EuoPLoP 2007), Irsee, Germany

Although not the main focus of this pattern, the advantages of a TT(C) approach also apply in distributed systems: see, for example, Scarlett and Brennan (2006).

### **Overall strengths and weaknesses**

- ☺ Use of TTC architecture tends to result in a system with highly predictable patterns of behaviour.
- ☹ Inappropriate system design using this approach can result in applications which have a comparatively slow response to external events.

### **Examples**

In applications where systems have been designed using event-triggered architecture it is occasionally later found to be necessary for higher reliability to migrate to time-triggered architectures. One such example is found in a recently published article (Turley, 2009). The article has mentioned the changes in the architecture design Sony Electronics have done to improve the performance. It states

“Given high packet rates Sony was hoping for, frequent interrupts turned from being a necessity to being a problem. In their experience, most network stacks are interrupt driven, especially from the hardware interface when it needs servicing. As data rates climb, these interrupts (and their attendant context switching) become so frequent that the overhead overwhelms the actual task. To fix this, the team has decided to switch from an interrupt driven to a software polled design. The resulting efficiency was dramatic”.

This pattern can be applied to a variety of systems from drive by x-wire systems to simple control systems where performance and reliability is an important issue.

### Context

- You are applying the pattern EVENTS TO TIME (TTC)
- You need to deal – cleanly – with a “long task” (that is, a task which may have an execution time greater than your chosen tick interval).
- You need to send a significant amount of data between your processor / system and an external device: the data transfer process will take some time.

### Problem

How can you structure the data-transfer tasks in your application in a manner which is compatible with TTC architecture?

### Background

We illustrate the need for the present pattern with an example.

Suppose we wish to transfer data to a PC at a standard 9600 baud. Transmitting each byte of data, plus stop and start bits, involves the transmission of 10 bits of information (assuming a single stop bit is used). As a result, each byte takes approximately 1 ms to transmit.

Now, suppose we wish to send this information to the PC:

```
Current core temperature is 36.678 degrees
```

If we use a standard function (such as some form of `printf()`) the task sending these 42 characters will take more than 40 milliseconds to complete. In a system supporting task pre-emption, we may be able to treat this as a low-priority task and let it run as required. This approach is not without difficulties (for example, if a high-priority task requires access to the same communication interface while the low-priority task is running). However, with appropriate system design we will be able to make this operate correctly under most circumstances.

Now consider the equivalent TTC design. We can't support task pre-emption and a long data-transmission task (around 40 ms) is likely to cause significant problems. More specifically, if this time is greater than the system tick interval (often 1 ms, rarely greater than 10 ms) then this is likely to present a problem as shown in Figure 2. The RS-232 task is a “long task” has duration greater than the system tick and so is missing the next tick intervals.

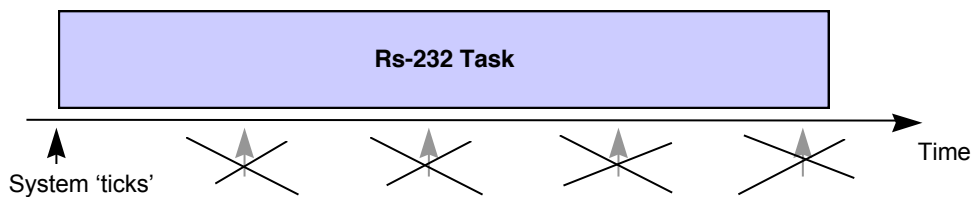


Figure 2: A schematic representation of the problems caused by sending a long character string on an embedded system. In this case, sending the message takes 42 ms while the System tick interval is 10 ms.

Perhaps the most obvious way of addressing this issue is to increase the baud rate; however, this is not always possible, and - even with very high baud rates - long messages or irregular bursts of data can still cause difficulties.

More generally, the underlying problem here is that the data transfer operation has a duration which depends on the length of the string which we wish to submit. As such, the worst-case execution time (WCET) of the data transfer task is highly variable (and, in a general case, may vary depending on conditions at run time). In a TTC design, we need to know all WCET data for all tasks at design time. We require a different system design. As Gergeleit and Nett (2002) have noted “ Nearly all known real-time scheduling approaches rely on the knowledge of WCETs for all tasks of the system.” The known WCET of tasks will be helpful for developers in designing the offline schedule and preventing task overrun.

## Solution

Convert a long data-transfer task (which is called infrequently and may have a variable duration) into a periodic task (which is called comparatively frequently and which has a very short – and known – duration).

A BUFFERED OUTPUT consists of three key components:

- A buffer (usually just an array, implemented in software)
- A function (or small set of functions) which can be used by the tasks in your system to write data to the array.
- A periodic (scheduled) task which checks the buffer and sends a block of data to the receiving device (when there are data to send).

Figure 3 below provides an overview of this system architecture. All data to be sent are first moved to a software buffer (a very fast operation). The data is then shifted – one block at a time – to the relevant hardware buffer in the microcontroller (e.g. 1 byte at a time for a UART, 8 bytes at a time for CAN, etc): this software-to-hardware transfer is carried out every 1ms (for example), using a (short) periodic task.

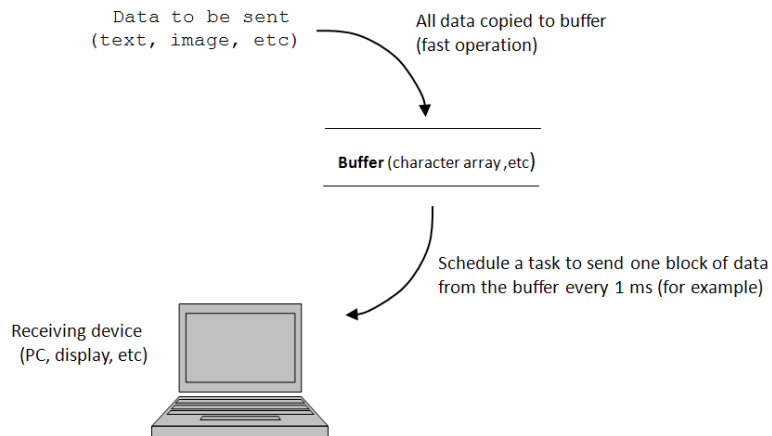


Figure 3: An overview of the BUFFERED OUTPUT architecture.

From Figure 3, it should be noted that the long task “RS-232 task “ in Figure 2 has been replaced by two short (high frequency) tasks, as shown in Figure 4:

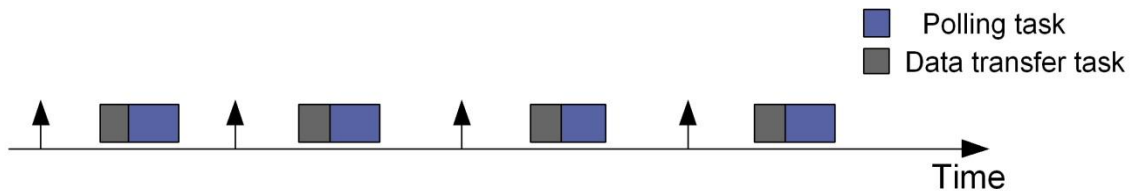


Figure 4: Long “RS-232” task is now transformed into two short tasks

## Hardware resource implications

In most cases, the CPU requirements for BUFFERED OUTPUT are very limited, provided we take reasonable care at the design stage. For example, if we are sending message over a CAN bus and we know that each message takes approximately 0.15 ms to transmit; we should schedule the data transmission task to check the buffer at an interval  $> 0.15$  ms. If we do this, the process of copying data from the software buffer to the (CAN) hardware will take very little time (usually a small fraction of a millisecond).

For very small designs (e.g. 8-bit systems) the memory requirements for the software buffer can prove significant. If you can’t add external memory in these circumstances, you will need to use a small buffer and send data as frequently as possible (but see the comment above).

In some cases, hardware support can help to reduce both memory requirements and processor load. For example, if using UART-based data transmission, UARTs often have 16-byte hardware buffers: if you have these available, it makes sense to employ them.

## Portability

This technique is generic and highly portable.

## **Reliability and Safety Issues**

- Special care must be taken while defining buffer length, the data transfer should not cause any buffer overflow
- Applications that involve high amount of data transfer like video and DSP applications or data acquisition systems the use of buffer might not be a viable solution.

## **Overall strengths and weaknesses**

- ☺ Use of buffered output is an easy solution for faster data transfer from a task running in an embedded application
- ☹ One has to be very careful while defining the buffer length, inappropriate buffer definitions may cause buffer overflow and data loss.

---

## RS-232 DATA TRANSFER (BUFFERED OUTPUT)

---

{pattern implementation example}

### Context

- You need to transmit data from your embedded processor using “RS-232”.
- You wish to transmit the data using a BUFFERED OUTPUT [this paper].
- Your chosen implementation language is C.
- Your chosen implementation platform is 8051 family of microcontrollers.

### Problem

How can you implement a BUFFERED OUTPUT in C for the 8051 family of microcontrollers?

### Background

RS232 is a standard asynchronous protocol used for serial communication between the processor and the peripherals; it is character oriented and is intended to be used with single 8-bit blocks of data. To transmit a byte of data over a serial link the data frame consists of a start bit to indicate start of transmission, the data itself(5 to 8 bits) and one or more stop bits to indicate the end of data block. RS232 can operate at different baud rates ranging from value of 75 to 330,000, however a baud rate of 9600 (a value lies in mid of the range) is recommended for safe use in our context.

### Solution

Use of Buffered output can greatly reduce the time for data transfer task as compared to printf() function. The printf() function sends data immediately to UART. As a result the duration of the transmission is usually too long. The pattern buffered output makes use of an intermediate software buffer (character array) in between. Sending data to buffer is a fast operation. Our code library of RS232 data transfer as shown in Listing 1 replaces the printf() function. A brief functionality of each function is shown in Table 2:

<i>Function</i>	<i>Functionality</i>
PC_LINK_IO_Write_String_To_Buffer()	This function copies a null terminated string to the character buffer. The contents of the buffer then passed over a serial link.
PC_LINK_IO_Write_Char_To_Buffer()	Stores a character in the 'write' buffer, ready for later transmission.
PC_LINK_IO_Get_Char_From_Buffer()	Retrieves a character from the (software) buffer, if one is available.
PC_LINK_IO_Update()	Checks for characters in the UART (hardware) receive buffer and sends next character from the software transmit buffer.

Table 2: RS-232 code library functions descriptions



```

void PC_LINK_IO_Write_String_To_Buffer(const char* const STR_PTR)
{
    tByte i = 0;

    while (STR_PTR[i] != '\0')
    {
        PC_LINK_IO_Write_Char_To_Buffer(STR_PTR[i]);
        i++;
    }
}

void PC_LINK_IO_Write_Char_To_Buffer(const char CHARACTER)
{
    // Write to the buffer *only* if there is space
    if (Out_waiting_index_G < TRAN_BUFFER_LENGTH)
    {
        Tran_buffer[Out_waiting_index_G] = CHARACTER;
        Out_waiting_index_G++;
    }
    else
    {
        // Write buffer is full
        // Increase the size of TRAN_BUFFER_LENGTH
        // or increase the rate at which UART 'update' function is called
        // or reduce the amount of data sent to PC
        Error_code_G = ERROR_USART_WRITE_CHAR;
    }
}

char PC_LINK_IO_Get_Char_From_Buffer(void)
{
    char Ch = PC_LINK_IO_NO_CHAR;

    // If there is new data in the buffer
    if (In_read_index_G < In_waiting_index_G)
    {
        Ch = Recv_buffer[In_read_index_G];

        if (In_read_index_G < RECV_BUFFER_LENGTH)
        {
            In_read_index_G++;
        }
    }

    return Ch;
}

void PC_LINK_IO_Update(void)
{
    // Deal with transmit bytes here

    // Is there any data ready to send?
    if (Out_written_index_G < Out_waiting_index_G)
    {
        PC_LINK_IO_Send_Char(Tran_buffer[Out_written_index_G]);

        Out_written_index_G++;
    }
    else
    {
        // No data to send - just reset the buffer index
        Out_waiting_index_G = 0;
        Out_written_index_G = 0;
    }
}

```

```

    }

    // Only dealing with received bytes here
    // -> Just check the RI flag
    if (RI == 1)
    {
        // Flag only set when a valid stop bit is received,
        // -> data ready to be read into the received buffer

        // Want to read into index 0, if old data has been read
        // (simple ~circular buffer)
        if (In_waiting_index_G == In_read_index_G)
        {
            In_waiting_index_G = 0;
            In_read_index_G = 0;
        }

        // Read the data from USART buffer
        Recv_buffer[In_waiting_index_G] = SBUF;

        if (In_waiting_index_G < RECV_BUFFER_LENGTH)
        {
            // Increment without overflowing buffer
            In_waiting_index_G++;
        }

        RI = 0; // Clear RT flag
    }
}

void PC_LINK_IO_Send_Char(const char CHARACTER)
{
    tLong Timeout1 = 0;
    tLong Timeout2 = 0;

    if (CHARACTER == '\n')
    {
        if (RI)
        {
            if (SBUF == XOFF)
            {
                Timeout2 = 0;
                do {
                    RI = 0;

                    // Wait for uart (with simple timeout)
                    Timeout1 = 0;
                    while ((++Timeout1) && (RI == 0));

                    if (Timeout1 == 0)
                    {
                        // USART did not respond - error
                        Error_code_G = ERROR_USART_TI;
                        return;
                    }
                } while ((++Timeout2) && (SBUF != XON));

            if (Timeout2 == 0)
            {
                // uart did not respond - error
                Error_code_G = ERROR_USART_TI;
                return;
            }
        }
    }
}

```

```

        RI = 0;
    }
}

Timeout1 = 0;
while ((++Timeout1) && (TI == 0));

if (Timeout1 == 0)
{
    // uart did not respond - error
    Error_code_G = ERROR_USART_TI;
    return;
}

TI = 0;
SBUF = 0x0d; // output CR
}

if (RI)
{
    if (SBUF == XOFF)
    {
        Timeout2 = 0;

        do {
            RI = 0;

            // Wait for USART (with simple timeout)
            Timeout1 = 0;
            while ((++Timeout1) && (RI == 0));

            if (Timeout1 == 0)
            {
                // USART did not respond - error
                Error_code_G = ERROR_USART_TI;
                return;
            }

        } while ((++Timeout2) && (SBUF != XON));

        RI = 0;
    }
}

Timeout1 = 0;
while ((++Timeout1) && (TI == 0));

if (Timeout1 == 0)
{
    // USART did not respond - error
    Error_code_G = ERROR_USART_TI;
    return;
}

TI = 0;

SBUF = CHARACTER;
}

```

Listing 1: RS232 Data transfer code Library using Buffered output

### Context

- You are applying the pattern EVENTS TO TIME(TTC)
- You need to poll inputs from the available interfaces (switches, keypads, sensors, ADCs) etc

### Problem

How do I build a TT system which is equivalent of my ET system such that it can respond to all (external/internal) input interfaces?

### Background

Designing a TT system requires more planning efforts. In a “TTC” design the possible occurrence and the execution times of all the tasks needs to be known in advance. The designer has to plan a task schedule which must execute all the tasks periodically at their allocated time intervals. This effort makes the system more predictable. In contrast to this, in an event triggered system the schedule executes the tasks dynamically as the events arrive thus no guarantee that they meet any timeliness constraints. This is the reason that ET designs are not recommended for safety critical applications. The event triggered behaviour in systems is achieved through the use of interrupts. To support these interrupts, Interrupt Service Routines (ISRs) are provided. Whenever an interrupt occurs it stops the currently running task and ISR executes to respond to the interrupt. This “context switching” is an overhead that sometimes raised serious complications in systems.

The abstract pattern EVENTS TO TIME [this paper] provides more relevant background information.

### Solution

A POLLED INPUT should meet the following specification:

- It should include a periodic task which polls for the occurrence of the event.
- The period of the above task should be set to some value less than or equal to minimum inter-arrival time\* of the event in question.
- The interrupt associated with this event should not be enabled. In fact only one interrupt associated with the timer responsible for generating system “ticks” should be enabled.

---

\* In ET systems the exact arrival time of events is not known so we assume a minimum distance between the arrivals of two consecutive events.

## **Hardware resource implications**

Different interfaces have different implications under various circumstances. Reading a switch input imposes minimal loads on CPU and memory resources whereas scanning the keypad interface imposes both a CPU and memory load.

## **Reliability and safety issues**

One major concern here in migrating from event triggered to time triggered is to make systems more predictable. Characteristic for the time triggered architecture is the treatment of (physical) real time as a first order quantity (Kopetz and Bauer 2002) this implies to the fact that time triggered systems must be very carefully designed, the task activation rates must be fixed according to the system dynamics i.e. how frequent an input needs to be polled.

## **Portability**

This technique is generic and highly portable.

## **Overall strengths and weaknesses**

- ☺ A flexible technique, programmer can easily do changes in code for example if auto repeat is required in case of SWITCH INTERFACE
- ☺ It is simple and cheap to implement.
- ☹ Provides no protection against out of range inputs or electrostatic discharge (ESD)
- ☹ More processor utilization in polling for tasks which are unlikely to occur, because the tasks are always tested for readiness whether actually enable or not.

---

## SWITCH INTERFACE (POLLED INPUT)

---

{pattern implementation example}

### Context

- You need to respond to a switch press from your embedded application
- Your chosen implementation language is C
- Your chosen implementation platform is NXP LPC2000 family of ARM7-based microcontrollers

### Problem

How can you implement POLLED INPUT for a simple switch press in C for NXP LPC2000 family of microcontrollers?

### Background

Simple push button switches as given in Figure 5 are very common in embedded applications. Pressing them causes a voltage change from  $V_{cc}$  to 0 volts at the input port. (For a detailed explanation see pattern SWITCH INTERFACE in PTES (Pont 2001)

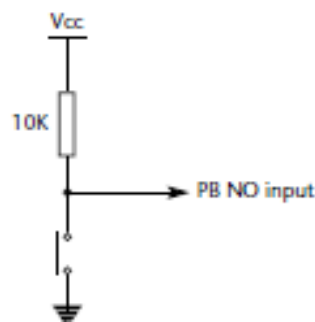


Figure 5: A simple push button switch with no internal pull-up

Note: There could be various types of switches (reset, on-off, multistate) for simplicity we are considering here only the simple interface push button switch with debounce support.

In an ideal world the change in voltage would take the form as shown in Figure 6:

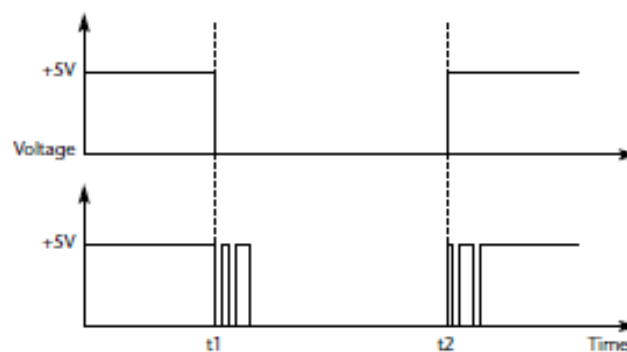


Figure 6: The voltage signal resulting from switch

In practical all mechanical switch contacts bounce (that is, turn on and off, repeatedly, for a short period of time) after the switch is closed or opened. As a result the actual waveform looks more like that shown in Figure 6 (bottom).

## Solution

Polling a switch for an input involves the following steps:

1. A relevant port pin is read and if a switch depression is detected it will be read again after their debounce period (provided in datasheets) to confirm the detection.
2. If it is confirmed that switch is pressed a task is run to respond to the switch press.

A simple code example illustrating simple switch detection and debounce system is given in Listing 2. This example demonstrates a simple switch interface with debounce support. The interface is implemented as a task which periodically checks a switch pin to see if it is pressed or not.

```

/*-----*-
switch.c (v1.00)
-----*/

Simple switch detection and debounce system.

#include "main.h"
#include "switch.h"
#include "port.h"
#define PORTS (1)

// ----- Private variable definitions -----
static uint32_t gSwitchListeners[PORTS],gLastState[PORTS],gCurState[PORTS];
// ----- Private constants -----
//Allows NO or NC switch to be used (or other wiring variations)
#define SW_PRESSED (0)

/*-----*-

Switch_Pin_Read()

Checks the read register bits locally to avoid excess overhead.
-----*/

#define Switch_Pin_Read(l,m,n) \
(((l) ? (IOPIN1) : (IOPIN0)) & n) ? (SW_PRESSED) : (1 - SW_PRESSED))

/*-----*-

Switch_Init()

Initialising the switches.

```

```

-*/-----*/
void Switch_Init()
{
    uint8_t port;

    for (port = 0; port < PORTS; port++)
    {
        gSwitchListeners[port] = 0;
        gLastState[port] = 0;
        gCurState[port] = 0;
    }

    //Initialise the output LED pin (LED_pin2) to indicate
    //switch is pressed
    PORT_Pin_GPIO_Set_Direction(LED_pin2, 1);
    PORT_Pin_Write(LED_pin2, 1);

    //Add Button_Pin to the switch listener
    Switch_AddListener(Button_Pin);
}
-*/-----*-

Switch_AddListener()

Adds a switch to the listen list for pins.

-*/-----*/

void Switch_AddListener(uint16_t pin)
{
    // Get the port
    uint8_t
        port = (uint8_t)(pin / 100);
    if (port < PORTS)
    {
        pin %= 100;
        gSwitchListeners[port] |= 1 << pin;
        PORT_Pin_Set_Mode(pin, 0, 0);

        PORT_Pin_GPIO_Set_Direction(pin, 0);
    }
}

-*/-----*-

Switch_RemoveListener()

Removes a switch from the listen list for pins. Does not remove the pin's mode.

-*/-----*/

void Switch_RemoveListener(uint16_t pin)
{
    // Get the port
    uint8_t
        port = (uint8_t)(pin / 100), bit;
    if (port < PORTS)
    {
        pin %= 100;
        bit = ~(1 << pin);
        gSwitchListeners[port] &= bit;
        gLastState[port] &= bit;
        gCurState[port] &= bit;
    }
}

```



```

    }
}
*-----*/

Switch_Update()

Detects and debounces switch presses on hooked pins. Could be recorded VERY
efficiently with direct HW access as this already has the bits to check in the pin
registers.

*-----*/
void Switch_Update()
{
    // Used for debugging with RapidITy
    TOTCR = 0;
    TOTCR = 1;

    uint8_t port, pin;
    uint32_t temp, bit;
    for (port = 0; port < PORTS; port++)
    {
        //Check for switches on this port

        if ((temp = gSwitchListeners[port]))
        {
            pin = port * 100;
            bit = 1;
            do
            {
                if (temp & bit)
                {
                    if (Switch_Pin_Read(port, pin, bit))
                    {
                        if (!(gCurState[port] & bit) && (gLastState[port] & bit))
                        {
                            // Was held before, debounced
                            // Call the callback
                            gCurState[port] |= bit;
                            Switch_Pressed(pin);
                        }
                        else
                        {
                            // Start debouncing
                            gLastState[port] |= bit;
                        }
                    }
                }
                else
                {
                    if ((gCurState[port] & bit) && !(gLastState[port] & bit))
                    {
                        // Was released before, debounced
                        // Call the callback
                        gCurState[port] &= ~bit;
                        Switch_Released(pin);
                    }
                    else
                    {
                        // Start debouncing
                        gLastState[port] &= ~bit;
                    }
                }
                temp &= ~bit;
            }
            pin++;
            bit <<= 1;
        }
        while (temp);
    }
}

```

```

// Used for debugging with RapidITy
TOTCR = 0;
TOTCR = 1;
}

/*-----*/

Switch_Pressed()

Called when a switch is first pressed and debounced.
Passes the full pin id of the activated input.

/*-----*/

void Switch_Pressed(uint16_t pin)
{
    if (pin == Button_Pin)
    {
        // Set to 0 to turn LED on
        PORT_Pin_Write(LED_pin2, 0);
    }
}

/*-----*/

Switch_Released()

Called when a switch is released. Passes the full pin id of the activated input.

/*-----*/

void Switch_Released(uint16_t pin)
{
    if (pin == Button_Pin)
    {
        // Set to 1 to turn LED off
        PORT_Pin_Write(LED_pin2, 1);
    }
}

/*-----*/

Switch_IsPressed()

Checks if a given pin is currently pressed and debounced.

/*-----*/

uint8_t Switch_IsPressed(uint16_t pin)
{
    uint8_t
        port = pin / 100;
    if (port > PORTS)
    {
        return 0;
    }
    return (gCurState[port] & (1 << (pin % 100))) ? (1) : (0);
}

/*-----*/
----- END OF FILE -----
/*-----*/

```

Listing 2: Simple switch press and debounce system using Polled Input

## Further reading

- Albert, A. and R. Bosch GmbH, (2004) "Comparison of Event-Triggered and Time-Triggered Concepts with regard to Distributed Control Systems", in *Embedded World*. : Nurnberg. p. 235-252.
- Allworth, S.T., 1981. *An Introduction to Real-Time Software Design*. , Macmillan, London.
- Audsley, N., Tindell, K. and Burns, A. (1993), "*The end of the line for static cyclic scheduling?*" Proceedings of the 5th Euromicro Workshop on Real-time Systems, Finland, pp. 36-41.
- Baker, T.P. and Shaw, A. (1989) "The cyclic executive model and Ada", *Real-Time Systems*, 1(1): 7-25.
- Bate, I.J. (1998) "Scheduling and timing analysis for safety critical real-time systems", PhD thesis, University of York, UK.
- Bate, I.J. (2000) "Introduction to scheduling and timing analysis", in "*The Use of Ada in Real Time Systems*" (6 April, 2000). IEE Conference Publication 00/034
- Bennett, S. (1994) "*Real-Time Computer Control*" (Second Edition) Prentice-Hall.
- Buschmann, F., Henney, K. and Schmidt, D.C. (2007) "Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing" (Volume 4). Wiley. ISBN: 978-0-470-05902-9
- Buttazzo, G. C. (2004), "*Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*", 2nd ed, Springer.
- Cottet, F. and David, L. (1999) "A solution to the time jitter removal in deadline based scheduling of real-time applications", 5th IEEE Real-Time Technology and Applications Symposium - WIP, Vancouver, Canada, pp. 33-38.
- Edwards, T., Pont, M.J., Scotson, P. and Crumpler, S. (2004) "A test-bed for evaluating and comparing designs for embedded control systems". In: Koelmans, A., Bystrov, A. and Pont, M.J. (Eds.) Proceedings of the UK Embedded Forum 2004 (Birmingham, UK, October 2004). Published by University of Newcastle.
- Fohler, G. (1999) "Time Triggered vs. Event Triggered - Towards Predictably Flexible Real-Time Systems", Keynote Address, Brazilian Workshop on Real-Time Systems, May 1999.
- Gergeleit, M. and Nett, E. 2002: Scheduling TRANSIENT OVERLOAD with the TAFT Scheduler. GI/ITG specialized group of operating systems.
- Hartwich F., Muller B., Fuhrer T., Hugel R., Bosh R. GmbH, (2002), Timing in the TTCAN Network, Proceedings 8th International CAN Conference.
- Herzner, W., Kubinger, W. and Gruber, M. (2006) "Triple-T - A system of patterns for reliable communication in hard real-time systems"; in D. Manolescu, M. Völter, J. Noble (eds): *Pattern Languages of Program Design 5 (PLOPD5)*; pp.89-126; *Software Engineering/Patterns Series*, Addison-Wesley, Boston. ISBN 0-321-32194-4
- Hong, S.H. (1995) "Scheduling algorithm of data sampling times in the integrated communication and control systems". *IEEE Transactions on Control Systems Technology*, 3(2): 225-230

- Kalinsky, D., 2001. Context switch, *Embedded Systems Programming*, 14(1), 94-105.
- Kopetz, H. (1997) "Real-time systems: Design principles for distributed embedded applications", Kluwer Academic.
- Kopetz, H. and Bauer, G (2002) "The Time-Triggered Architecture", *IEEE Special Issue on Modeling and Design of Embedded Software* pp.112-126.
- Kurian, S. and Pont, M.J. (2007) "Maintenance and evolution of resource-constrained embedded systems created using design patterns", *Journal of Systems and Software*, 80(1): 32-41.
- Liu, C. L. and Layland, J. W. (1973), "Scheduling algorithms for multi-programming in a hard real-time environment", *Journal of the ACM*, 20(1): 40-61.
- Locke, C.D. (1992) "Software architecture for hard real-time systems: Cyclic executives vs. Fixed priority executives", *The Journal of Real-Time Systems*, 4: 37-53.
- Maaita, A. and Pont, M.J. (2005) "Using 'planned pre-emption' to reduce levels of task jitter in a time-triggered hybrid scheduler". In: Koelmans, A., Bystrov, A., Pont, M.J., Ong, R. and Brown, A. (Eds.), *Proceedings of the Second UK Embedded Forum* (Birmingham, UK, October 2005), pp.18-35. Published by University of Newcastle upon Tyne [ISBN: 0-7017-0191-9].
- Massa, A.J. (2003) "Embedded Software Development with eCOS", Prentice Hall. ISBN: 0-13-035473-2.
- Nissanke, N., 1997. *Realtime Systems*. , Prentice-Hall.
- Pont, M.J. (2001) "Patterns for Time-Triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers", Addison-Wesley / ACM Press. ISBN: 0-201-331381.
- Pont, M.J. (2002) "Embedded C", Addison-Wesley. ISBN: 0-201-79523-X.
- Pont, M.J. (2008) "Applying time-triggered architectures in reliable embedded systems: Challenges and solutions", *Elektrotechnik & Informationstechnik*
- Shaw, A.C. (2001) "Real-time systems and software" John Wiley, New York. [ISBN 0-471-35490-2]
- Scarlett, J.J. and Brennan, R. W (2006) "Re-evaluating Event-Triggered and Time-Triggered Systems", in *IEEE conference on Emerging technologies and factory automation*. p. 655-661.
- Torngren, M. (1998) "Fundamentals of implementing real-time control applications in distributed computer systems", *Real-Time Systems*, vol.14, pp.219-250.
- Turley, J (2009) "Gaming the systems –high end networking on the cell processor" *Embedded.com Issue* September 2009.
- Ward, N. J. (1991) "The static analysis of a safety-critical avionics control system", in Corbyn, D.E. and Bray, N. P. (Eds.) "*Air Transport Safety: Proceedings of the Safety and Reliability Society Spring Conference, 1991*" Published by SaRS, Ltd.
- Xu, J. (1993) "Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence, and Exclusion Relations," *IEEE Transactions on Software Engineering*, 19(2), pp. 139-154.

- Xu , J. and Parnas, D.L. (1990) "Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations," IEEE Transactions on Software Engineering, 16(3), pp. 360-369.
- Xu , J. and Parnas, D.L. (1993) "On Satisfying Timing Constraints in Hard-Real-Time Systems," IEEE Transactions on Software Engineering, 19(1), pp. 70-84.
- Xu , J. and Parnas, D.L. (2000) "Priority Scheduling Versus Pre-Run-Time Scheduling," International Journal of Time-Critical Systems, 18, 7-23, Kluwer Academic Publishers.

# Invocation Assembly Lines

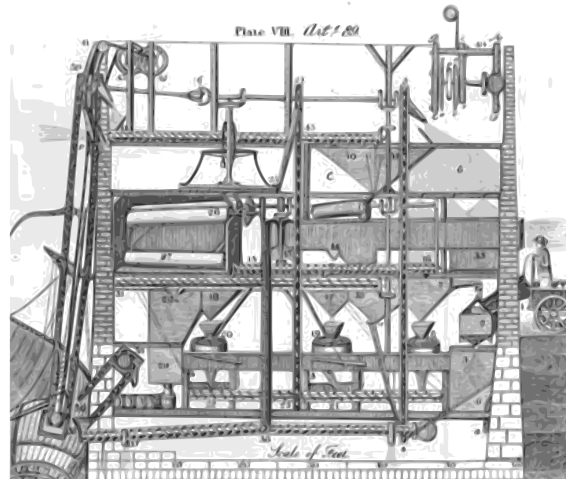
## Patterns of Invocation and Message Processing in Object Remoting Middleware

Stefan Sobernig<sup>1</sup> and Uwe Zdun<sup>2</sup>

<sup>1</sup>*Institute for Information Systems and New Media  
Vienna University of Economics and Business (WU Vienna), Austria  
stefan.sobernig@wu.ac.at*

<sup>2</sup>*Distributed Systems Group, Information Systems Institute  
Vienna University of Technology, Austria  
zdun@infosys.tuwien.ac.at*

Object remoting middleware greatly facilitates creating distributed, object-oriented systems. However, developers face many situations in which a middleware's invocation and message processing architecture fails to fully support all their requirements. This problem is caused, for instance, by limitations in realising certain invocation styles (e.g., one-way and two-way conversations) on top of a shared processing infrastructure, in adding extensions to invocation handling (i.e., add-on services such as security and inspection), and in bypassing selected steps in the invocation handling to balance resource consumption and invocation performance. Often, these limitations are caused by design and implementation decisions taken early when crafting the middleware framework. To better explain the needed decision making, and help developers to apply adaptations or guide the selection of alternatives, we present a pattern language that captures the essentials of invocation and message processing in object remoting middleware. We also outline instantiations of the patterns and their relationships in existing middleware frameworks.



Evans' improved mill [22, Plate VIII] advanced the automation of pre-industrial further processing and refinement of wheat into flour. It is commonly regarded as an early predecessor of Fordist and more recent assembly line or production flow systems [44].

# 1 Introduction

Developers who design and implement object remoting middleware are confronted with the task of supporting many different kinds of remote invocations. On the one hand, this variety is caused by the architectural setting in which the middleware is deployed and used. On the other hand, realising different kinds of remote invocations affects many spots of your middleware framework's design and implementation. These two issues are the root of two challenging design problems for the messaging processing infrastructure of a middleware.

The first design problem is the issue of *role distribution* (see e.g. [67, 63, 52, 3]): Applications built on top of your middleware framework must often play different roles, for example, they are expected to act both as a client issuing and as a server performing remote invocations. As these applications integrate the facilities offered by your middleware framework, your framework must facilitate the adoption by client- as well as server-side applications.

The second design problem is that there are a number of *crosscutting* concerns to be considered in your middleware framework design (see e.g. [34, 54, 70]): First, many application scenarios for your middleware require different remote invocation styles, component interaction styles, inter-component dependencies, levels of communication coupling, and so on. Second, many add-on tasks might be required, such as security or inspection add-ons. Finally, certain application scenarios are better addressed if your framework provides the flexibility to selectively omit certain steps in handling remote invocations. For example, optimisations may be applicable if signatures in interfaces of remote objects change frequently. This usually requires *bypassing* certain steps in the default control and data flow of your middleware framework. Each of these aspects, i.e., add-on services, component interaction and invocation styles, and bypassing, requires the interplay of different parts of the framework. Therefore, when developing your framework, you must anticipate a certain flexibility across essential building blocks of your framework.

This paper documents established design practises for realising versatile message processing infrastructures for remote invocations in object remoting middleware. The identified design practises set out to tackle the tensions caused by the crosscutting concerns of invocation handling and the issue of role distribution. We mined the design practises from existing object remoting middleware frameworks such as OpenORB [58], Mono .NET Remoting (Mono/R; see [40]), Mono Olive (Mono/O, [41]), Apache Axis2/Java (Axis2; see [5, 46, 21]), and Apache CXF (CXF; see [6]). We present our findings in terms of a pattern language and comment on the identified uses of the documented patterns.

This paper and the pattern language described herein are meant for those developers who must deepen their understanding of how object middleware frameworks can be designed in a manner to support varieties of remote invocations. The paper introduces the necessary terminology to foster your conceptual understanding of the inner workings of the middleware's invocation and message processing architecture. The background of this work are established remoting patterns – in the architectural context of the BROKER pattern [14, 32, 63, 55, 12]. The resulting pattern language provides links to existing remoting pattern collections [63, 12] and helps navigate in this body of established and documented design practises. An additional audience is the group of middleware users, who find structured guidance to evaluate existing object remoting middleware regarding its fit with requirements on supporting diverse invocation styles and on the middleware's extensibility. Finally, our pattern language targets developers who must further develop or adapt an existing middleware framework for complying with certain remote invocation types.

The paper is structured as follows. In the next section, we provide an overview of elementary terminology on object remoting and give some background on the BROKER pattern (see Section 2). Then, we introduce the reader to the pattern language as a whole and provide some hints on navigating between its patterns (see Section 3). Before giving the pattern descriptions themselves, we discuss the challenges of adaptable invocation and message processing (see Section 4). We provide a motivating

example in Section 4.2. In Section 5, the individual patterns of our pattern language are presented in detail. After having resolved the motivating example in Section 6, we describe known uses of the individual patterns in existing middleware frameworks (see Section 7). Finally, we conclude by discussing our pattern language and how it integrates with existing remoting patterns (see Section 8).

## 2 Some Background: Broker-Based Object Middleware

Assume you are using a middleware framework which follows the BROKER pattern [14, 32, 63, 55, 12], as it is the case for most modern object-oriented RPC middlewares, such as CORBA, .NET Remoting [40, 50], Windows Communication Foundation [41], and Web Services [9, 39, 16, 17] frameworks such as Axis2 [5] or CXF [6]. Figure 1 shows a set of basic remoting patterns from [63] and their interactions, forming a bare BROKER-based middleware framework. For a thumbnail overview of relevant object remoting patterns (and, for later reference) see Appendix A.

The exemplary BROKER is shown in the configuration for performing a remote invocation between a client component and a remote object: The client component performs an invocation on the remote object. Crossing the network boundary is handled by the middleware. That is, the REQUESTOR receives the invocation and constructs a request from the essential invocation data, i.e., the reference to remote object, the operation name, and the parameters. This results in a canonical object representation of the request. Then, the REQUESTOR uses a MARSHALLER to stream the objectified invocation data into a MESSAGE [29]. The MESSAGE is handed over to the CLIENT REQUEST HANDLER, along with the reference to the targeted remote object, for resolving the corresponding network endpoint, establishing a connection, and delivering this request MESSAGE. The MESSAGE arrives at the SERVER REQUEST HANDLER at the server side. Subsequently, the MESSAGE is forwarded to the INVOKER that initiates the disassembling of the MESSAGE. Disassembly involves demarshaling of the MESSAGE by a MARSHALLER. Demarshaling means to extract object references, contextual invocation data, and the core invocation data based on a canonical in-memory representation. Finally, the INVOKER resolves the remote object and dispatches the actual invocation. The invocation result is processed and returned in reverse order.

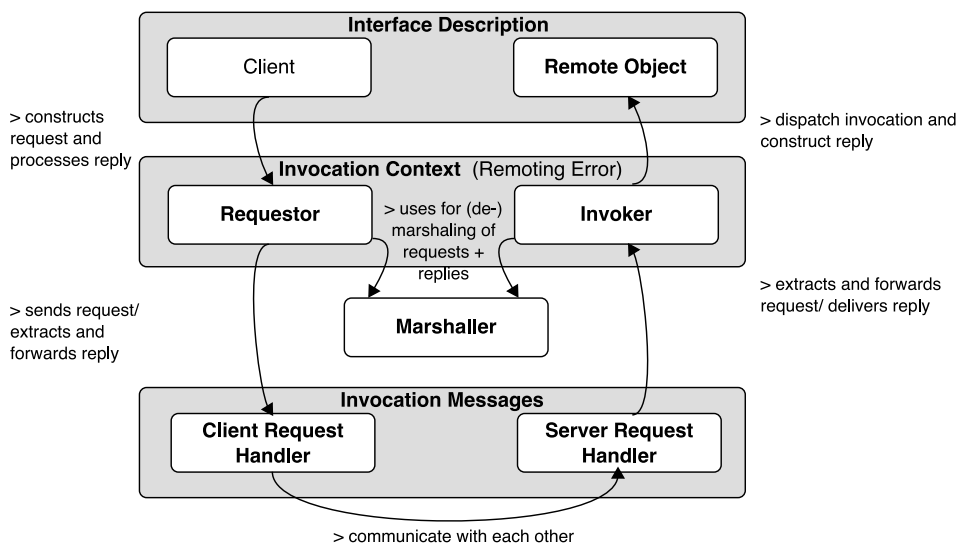


Figure 1: Relevant remoting patterns and their interactions

The structure of these remoting patterns and their interactions involve the processing and exchanging of invocation data items. This includes core invocation and result data (e.g., object references, operation names, parameters, object-type meta-data, MESSAGES [29, 12], and REMOTING ERRORS [63]), contextual invocation data (e.g., kinds of INVOCATION CONTEXTS [63]), and auxiliary invocation data



items which is not directly involved in a remote invocation (e.g., an INTERFACE DESCRIPTION [63]). In Figure 1, the invocation data items are represented by horizontal connectors that relate the instances of the remoting patterns operating on invocation data items at the same LAYER. For instance, the REQUESTOR constructs requests targeted at the INVOKER, and it processes reply objects generated by the INVOKER. The invocation data items involved are described in a number of remoting patterns. An overview of these patterns is provided by pattern thumbnails in Appendix A. More in-depth pattern descriptions and known uses are given in Appendix B.

### 3 Pattern Language Overview

This paper presents patterns focused on adaptable message processing infrastructures as they can be found in current object remoting frameworks, such as OpenORB [58], Mono .NET Remoting (Mono/R; see [40]), Mono Olive (Mono/O, [41]), Apache Axis2/Java (Axis2; see [5, 46, 21]), and Apache CXF (CXF; see [6]). The invocation and message processing infrastructures represent an integral extension mechanism of these middleware frameworks for realising support for add-on services, multiple invocation styles, and the selective bypassing of message processing steps.

The pattern language integrates the six patterns documented in this paper (see also Figure 2) with remoting patterns organised in two existing pattern languages: On the one hand, it extends the *Remoting Patterns* language documented in [63]. On the other hand, the patterns presented here link to the *Pattern Language for Distributed Computing* [12]. Whilst there is a certain overlap between these two pattern languages, in particular regarding the foundational BROKER pattern, each of these pattern languages contributes distinct patterns for better understanding of invocation and message processing. The Remoting Patterns contain extension and extended infrastructure patterns (e.g., INVOCATION INTERCEPTORS and CONFIGURATION GROUPS) which emphasise an adaptation view of a middleware framework. As a complement, the Pattern Language for Distributed Computing provides details on invocation data items processed, that is, kinds of MESSAGES. In this sense, this language stresses a data flow view of middleware frameworks. We aim at combining the adaptation and data flow views into a coherent description. Relevant patterns taken from these two pattern languages (and beyond) are presented as pattern thumbnails in Appendix A.

As a developer of a middleware framework, you usually foresee at least one of the following kinds of adaptability. In complex requirements settings, your framework must support all of them. The three kinds of adaptability correspond to the first three patterns in our pattern language:

- PARTIAL PROCESSING PATHS (see Section 5, pp. 15) are required to support different variants of one-way invocations. They allow you to lay out a complex processing scheme for invocation data and, in certain invocation scenarios, only to enact selected ranges of this scheme. This is needed for FIRE AND FORGET invocations, such as WSDL/1.1 one-way operations [16] or WSDL/2.0 in-only operations [17].
- RECONFIGURABLE PROCESSING PATHS (see Section 5, pp. 18) allow you to introduce additional processing operations, on demand and at arbitrary times in invocation processing. Thus, you can define a minimum processing scheme expected to be common to all or most invocation scenarios and permit extension developers to add processing operations as needed. Important examples of extended processing requirements are security-related, add-on invocation services, such as those described by the Web Services Security Core Specification (WSS/Core 1.0/1.1; [42]).
- PROCESSING SHORTCUTS (see Section 5, pp. 20) put you into the position to describe several possible walks through a processing scheme. While you lay out a basic scheme common to all or most invocations processed, you can still allow for deviating processing flows, for instance, by skipping a number of processing steps if required. This helps you to realise more complex forms

of exception handling, client- and server-side caching, redirecting invocations upon collocated remote objects, and so on.

Facing these requirements on adapting the message processing infrastructure, i.e., supporting previously unanticipated invocation styles, attaching add-on processing behaviour on demand, or bypassing of scheduled processing operations, you proceed by applying the INVOCATION ASSEMBLY LINE pattern. The INVOCATION ASSEMBLY LINE pattern describes your processing infrastructure in terms of *processing stations* and *processing tasks*. Processing stations denote elementary steps in the lifecycles of your REQUESTOR and INVOKER, or, similarly, the invocation data items processed. Invocation data items are characterised by certain processing states, e.g., `message constructed`, `message marshaled`, `message delivered`, and so on. Processing tasks describe processing operations performed on invocation data items in certain lifecycle states, e.g., a `marshaling` operation once entering the state `message constructed`.

Once you decided to adopt the INVOCATION ASSEMBLY LINE pattern, you face two further design problems:

- Is the number of processing stations fixed or adjustable?
- Do processing stations accept multiple task assignments or can each processing station only perform a single processing task?

The former design problem relates to designing and managing the *processing station layout* and the latter relates to the *station-task assignment*. In resolving these two issues differently, you have two choices for realising the INVOCATION ASSEMBLY LINE pattern:

- SINGLE-TASK PROCESSING STATIONS (see Section 5, pp. 22) describes the processing infrastructure by a number of processing stations and each of them is performing only a single processing task. In its extreme, you abandon the distinction of processing task and processing stations. The ultimate advantage of this variant is that there is no need for a dedicated, central management facility that allows you to map tasks to processing stations. You can adopt a decentralised organisation of invocation processing tasks.
- MULTI-TASK PROCESSING STATIONS (see Section 5, pp. 24), in contrast, allows you to assign multiple processing tasks to single processing stations. This involves applying absolute or relative ordering strategies for tasks attached to a single station. This variant puts you into the position of describing a common layout of processing stations, which is commonly fixed at design or configuration time, with distinct invocation scenarios resulting in different task configurations. The variable assignment often requires a central management of station-task mappings which entails changes at several spots (e.g., a manager and the station entities) to adjust the processing infrastructure to a new invocation scenario.

The patterns outlined above form a web of relations (see Figure 2). The INVOCATION ASSEMBLY LINE pattern is the *main* pattern of this language. It focuses on conceptualising and specifying invocation and MESSAGE processing. However, both the problem and solution of that pattern appear in a considerable number of variations so that we decided to treat these variations as distinct patterns. To begin with, we identified three *problem variants*: PARTIAL PROCESSING PATHS, RECONFIGURABLE PROCESSING PATHS, and PROCESSING SHORTCUTS. These three patterns vary from the main INVOCATION ASSEMBLY LINE pattern *by problem*, i.e., the INVOCATION ASSEMBLY LINE pattern is equally used to resolve three related, but sufficiently distinct problems. You may also think of use relationships between the three patterns and the INVOCATION ASSEMBLY LINE pattern. Similarly, there are two *solution variants*: MULTI-TASK PROCESSING STATIONS and SINGLE-TASK PROCESSING STATIONS. A

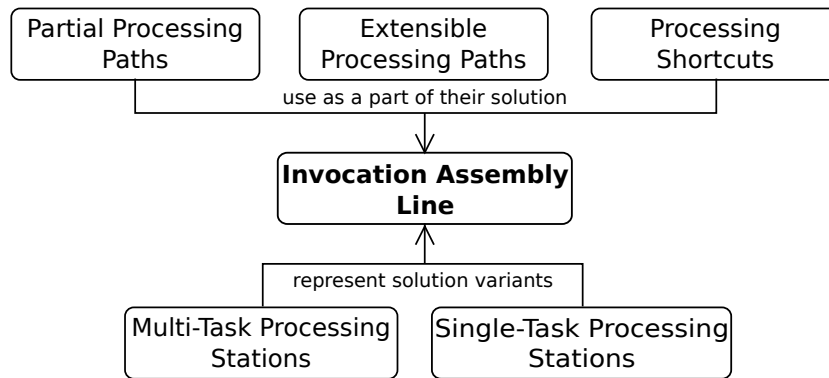


Figure 2: The patterns and their relationships

solution variant describes an alternative solution to the same problem. These variants share the problem stated in the main INVOCATION ASSEMBLY LINE pattern. Problem and solution variants form the two *fragments* [43] of our pattern language.

## 4 Challenges of Adaptable Invocation and Message Processing

The pattern language documented in this paper addresses situations in which a LAYERS-based view of a BROKER-based middleware framework (see also Figure 1; [14, 8]) turns out to be insufficient to effectively design and implement a middleware framework with particular extension capabilities. In such a view, the BROKER is segregated into layers of functional responsibilities under a rigid directionality. Components at higher-level layers, e.g., the client component, are constrained to use only functionality offered by components residing at their direct descendant layers, e.g., the REQUESTOR. As for crosscuts, applying modifications to components residing at all LAYERS or conditionally circumventing components in intermediate LAYERS is not considered. For instance, the client component is not expected to construct a request object on its own and hand it over to the MARSHALLER and CLIENT REQUEST HANDLER.

The LAYERS structure illustrated in Figure 1 also shows the client component and the remote object, as part of the server application, as conceptually separated. There is no notion of these two components exchanging their roles, so that the client turns into the server side (and vice versa). Also, since your framework is to be integrated by both client and server applications, it must integrate support for either side on a common ground. Otherwise, you risk introducing an unfavourable distinction between a *client-* and the *server-concrete framework* (see e.g. [52]), thus developing two sub-frameworks design- and implementation-wise. Such a distinction overlooks potentials for reducing design and implementation complexity by identifying shared characteristics in organising the processing of invocations and MESSAGES between, e.g., the INVOKER and the REQUESTOR components.

### 4.1 Realising Adaptability in Middleware Framework Design

Existing middleware frameworks provide variants of the INVOCATION INTERCEPTOR [63, 52, 53] pattern to extend this LAYERS structure by means of *indirection*: The INVOCATION INTERCEPTOR pattern [63] supports the definition of single processing operations that operate on invocation data items which are then transmitted using INVOCATION CONTEXTS [63] from the client to the server (and vice versa). INVOCATION INTERCEPTORS are registered with hooks placed within the processing infrastructure, for instance upon entering and upon exiting the MARSHALLER. Once reached, the hooks intercept the invocation data items processed (e.g., the request and reply objects) and have the processing operations defined by the INVOCATION INTERCEPTORS performed on them. The INVOCATION INTERCEPTORS

can be user-configured using CONFIGURATION GROUPS [63] that allow developers to define a number of related interceptors in a reusable group.

These patterns describe a common extension architecture, but they do not explain the internal mechanisms of the middleware to realise the composition of the processing tasks. In addition, not all processing tasks in middleware frameworks are INVOCATION INTERCEPTORS, and not all invocation data items are transmitted using INVOCATION CONTEXTS. For instance, some middleware frameworks define only the user-defined extensions as interceptors and use other mechanisms to define the basic processing tasks. It would be desirable both for the middleware designer and the middleware user to use one and the same internal mechanism to define and configure all processing tasks in a middleware.

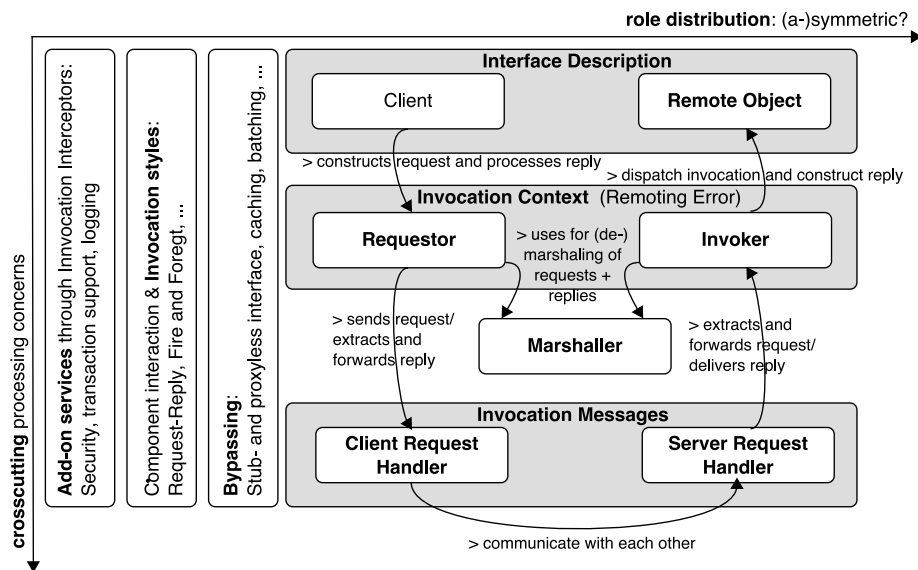


Figure 3: Requirement dimensions: role distribution and crosscutting processing concerns

There are two particular design problems related to finding such a canonical mechanism, originally discussed for the INVOCATION INTERCEPTOR pattern (see in particular [52]):

- *Hook selection:* At which LAYERS should hooks be placed? Also, at which spots should the invocation data items be indirected, e.g., on exiting, on entering, or somewhere within a given LAYER? The time of fixing this layout, e.g., the design, the configuration, or even runtime, is equally important. Providing for a wide and predetermined coverage through hooks risks causing substantial resource overhead and architectural complexity. Adopting a too limited number might prevent you or framework integrators from realising future extensions without modifying the basic INVOCATION INTERCEPTOR infrastructure.
- *Access protocol:* The access protocol regulates which elements of the invocation data items are exposed to and are made mutable by INVOCATION INTERCEPTORS. In addition, it regulates to which extent INVOCATION INTERCEPTORS influence the overall control flow. For instance, INVOCATION INTERCEPTORS might be allowed or disallowed to issue or process REMOTING ERRORS. The permissiveness of the access protocol balances the extensibility of the framework and the need for reliability when handling invocations on behalf of the middleware users. For instance, a too permissive access protocol might allow extension developers to manipulate core invocation data (e.g., the number of parameters), causing behaviour unexpected by the client application developer.

The pattern language presented in this paper helps you to understand and to address these two design problems. We focus on four classes of concerns which complicate the decision finding process due

to crosscutting the LAYERS structure of your middleware framework (see the vertical axis shown in Figure 3). They relate to components at several or all LAYERS: orthogonal add-on services, support for component interaction and their underlying invocation patterns, and kinds of LAYERS bypassing, and role distribution. The crosscutting structure of these concerns make it difficult to implement add-on services, different invocation styles, and kinds of bypassing in a fixed, foreseen way. Also, your framework design must be integratable by applications which take the client and server roles in turn. Following from this, the realisation of add-on services, invocation styles, and LAYERS bypassing must take into account framework support for both the client and server sides. We discuss this escalation in design complexity as the issue of role distribution (see the horizontal axis of Figure 3). A more detailed discussion of these four areas of challenge is provided in Appendix C.

## 4.2 A Motivating Example

Let us consider the situation shown in Figure 4 as an example. You need to secure remote invocations by means of encryption. There are many alternatives available to tackle such a requirement, most notably, transport-level encryption (for instance, TLS [20] or SSL [24]) and MESSAGE-level encryption (e.g., S/MIME; see [51]). However, both lack guarantees for secure *end-to-end* deliveries between a client and server application regarding transport intermediaries and MESSAGE authenticity. In addition, invocation data passes lower LAYERS of the BROKER unencrypted (see also Figure 1). Also, you are expected to add an invocation-level facility which permits client and server application developers to make use of *selective encryption and decryption* of MESSAGES sent and received. By selective, we mean that only parts of the MESSAGES are to be sealed, in particular the core invocation data. Data transmitted as the INVOCATION CONTEXT, which is also relevant for negotiating an en- and decryption scheme and routing MESSAGES, is to be left untouched. The requirements of end-to-end security and selectivity ruling out basic transport-level and MESSAGE-level options. Consequently, you must address this issue as an integral part of your invocation and MESSAGE processing infrastructure.

The UML activity diagram in Figure 4 presents a control and data flow view of the client side of processing invocations and MESSAGES (see also Figure 1). While the essential processing steps (e.g., `Construct request`, `Stream request`) are modelled as activities, activity partitions are used to identify the responsible remoting pattern for each of these steps (e.g., REQUESTOR, MARSHALLER). In addition, we represent data flow artefacts by means of input and output pins to activity nodes. For instance, `Invocation data` is the required input, and a `Request object` is the expected output of the `Construct request` activity owned by the REQUESTOR. Note that this activity diagram visualises the two-pass processing involved for the client side. That is, once the processing steps are performed on the request, and once on the reply (shown using the swimlane notation for activity partitions).

Given this control and data flow view of your given framework, you must decide how to realise the selective encryption facility, the `Security provider` (see Figure 4). This is essentially determined by the input required by such a facility, namely a streamed MESSAGE. The object representation of either request and reply are not suitable for applying encryption or decryption. This is mainly because their state is usually subject to further mutation and because they can't be properly constructed from the encrypted data enclosed by the MESSAGE. Therefore, you must provide means to operate on either the `Request` or `Reply` message, i.e., the output of the MARSHALLER for the outgoing request and the output of the CLIENT REQUEST HANDLER for the incoming reply. Which options are available to you?

1. **Security-aware MARSHALLER:** You might want to consider refining your MARSHALLER instantiation to perform the selective encryption. While this appears as a viable option at first sight, it would soon turn out impracticable because of code cluttering and constrained extensibility. Code cluttering would result from introducing conditional branching in the MARSHALLER to al-

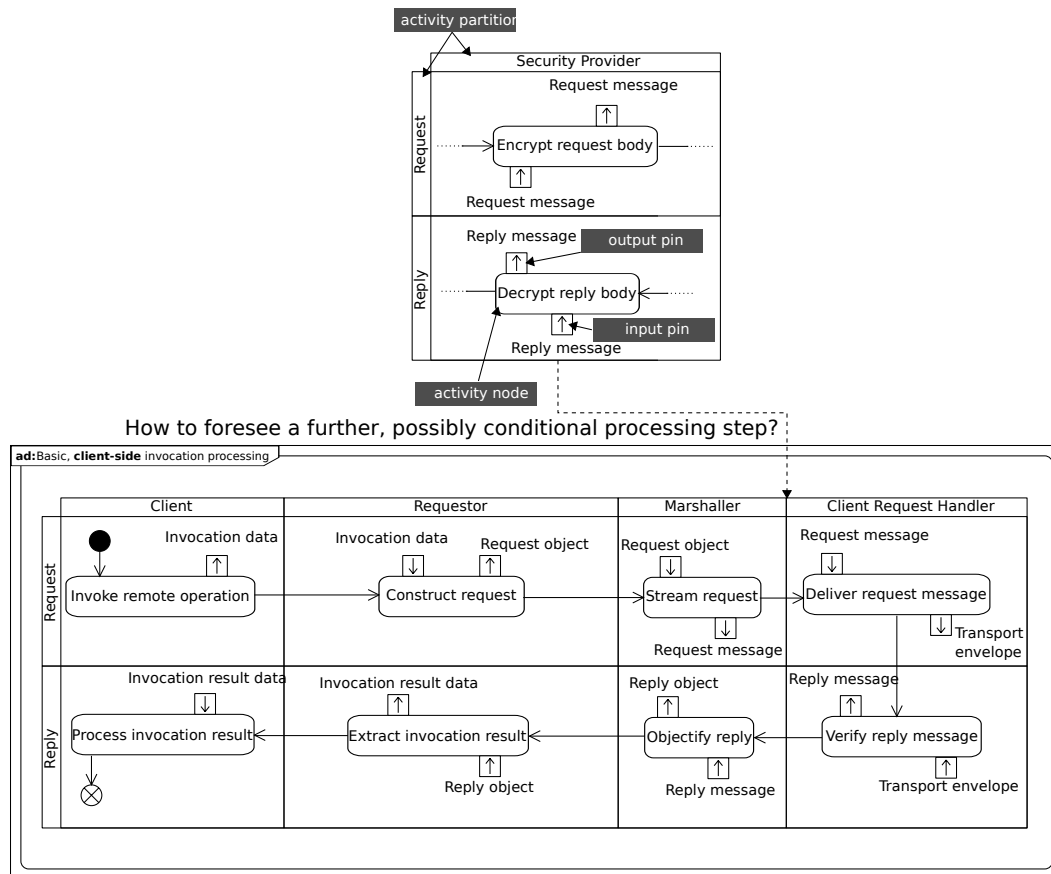


Figure 4: Example of an add-on service: Security provider for selective message encryption

low for selectively enabling or disabling encryption by clients. Extensibility and maintainability would be constrained if you supported a set of MARSHALLERS for different MESSAGE formats. In that case, future adjustments would have to be tracked by each of them.

2. Security-aware REQUEST HANDLERS: Alternatively, you could turn selective encryption into a responsibility of the CLIENT REQUEST HANDLER which has access to the required MESSAGE representations. Here, the critique put forth against the first, MARSHALLER-only refinement strategy applies as well. However, the situation turns out even worse: Given CLIENT REQUEST HANDLER variants for different transport protocols, you would also couple the transport to the message handling concern. This is due to the fact that the Security Provider requires intimate knowledge of the MESSAGE format (e.g., a SOAP/XML dialect) in order to select the parts meant to be encrypted or decrypted. Transport handling and transport protocol adoption (e.g., TCP [47], HTTP [23], or SMTP [48]) would be limited in their reusability. They would lose their independence from the MESSAGE format applied.
3. Security provider through INVOCATION INTERCEPTORS: Provided that your framework provides an INVOCATION INTERCEPTOR infrastructure (or, you plan to equip it with one), you can solve this extension problem by turning the Security provider into a set of INVOCATION INTERCEPTORS. You need at least two interceptors; one for providing the encryption and the other the decryption service. However, it is important that your INVOCATION INTERCEPTOR infrastructure is compatible with the specific requirements of a selective en- and decryption service (as shown in Figure 4). First, we require hooks placed at the end of the processing activities performed by the MARSHALLER or, alternatively, before the CLIENT REQUEST HANDLER. If either was missing, the Security provider would not be realisable by means of INVOCATION INTERCEPTORS because the MESSAGE representations needed could not be intercepted.

Second, the access protocol to the invocation data (i.e., the `Request` and `Reply` messages) must be permissive enough. For instance, if the access to and the mutation of the request and reply bodies was denied and only context data (e.g., for setting the encryption scheme and its details) was exposed to the `INVOCATION INTERCEPTORS`, realising the `Security provider` would not be feasible.

Given the three strategies outlined above, their inadequacies and possible constraints, what mechanisms remain open to you if you want to incorporate the `Security provider` into the processing control and data flow, as shown in Figure 4? When looking at a general-purpose extension infrastructure such as `INVOCATION INTERCEPTORS`, how can its design anticipate a wider range of extension requirements? How can it be made adjustable to fit those emerging in the continued lifecycle of your framework?

Not enough, this problem turns even more complicated. What if this addition must be configurable for different scopes, e.g., per-client or per-endpoint? How can you ensure that the addition of this encryption add-on preserves the modular organisation of `REQUESTOR` and `CLIENT REQUEST HANDLER`? Also, realising a `Security provider` both for client and server applications reinforces the problem. All this points to the issue whether the `LAYERS` structure still serves for the task of designing such an add-on service and the necessary framework facilities. This example reflects our motivation to mine existing middleware frameworks for the solutions adopted in response to more general classes of requirements on adaptable invocation and message processing.

## 5 Pattern Descriptions

### Invocation Assembly Line

As an application developer, you use an existing BROKER-based middleware framework and you want to extend the message processing operations offered by this middleware framework. Or, as a framework developer, you design an extensible BROKER-based middleware framework. Extensibility is required to support application developers in adapting infrastructure for message processing to their application-specific requirements. This involves adding, removing, or replacing custom processing steps on various invocation data items. Certain kinds of invocation data items are found in almost all BROKER-based middleware frameworks, e.g., different types of messages, interface descriptions, etc. In addition, these invocation data items are subject to common processing operations, such as adopting a uniform object representation or their transformation into structured character or byte streams.

#### **How do you extend a BROKER-based middleware framework with extra processing operations and application-specific refinements over invocation data items?**

Functional extensions take either the form of application-specific ones or framework refinements as such, the latter being applicable to multiple, possibly related applications built on top of the middleware framework. To create such extensions, application and framework developers require access to the processing infrastructure for invocation data encapsulated by the INVOKER and the REQUESTOR. The processing infrastructure should be configurable and adaptable through a canonical programming model, at design time by the extension developer and at runtime by means of reflection, respectively. However, providing such a canonical programming model for configuring and adapting the processing infrastructure bears the risk of lacking the needed flexibility if the variety of invocation data items (e.g., MESSAGE kinds), the invocation patterns, the processing operations, and their interdependencies are not known when designing the middleware core. A threat to finding a balanced solution comes through the considerable variety of *processing operations* to be supported. This variety results both from different kinds of invocations to process and the processing range to cover.

To begin with, processing does not only refer to handling core invocation data. On the contrary, a BROKER must handle *auxiliary* kinds of invocations. These often require a very different processing scheme. Examples include the generation and delivery of INTERFACE DESCRIPTIONS (such as in Mono .NET Remoting) as well as certain auxiliary event MESSAGES. The latter commonly represent notifications orthogonal to the underlying remote invocation. In Web Services Reliable Messaging (WS/RM; [19]), for instance, message sequencing is implemented by particular notification messages which are exchanged completely transparent to the actual invocation messages. Also REMOTING ERRORS form a distinct group of invocation data to process.

When being processed, invocation data items pass different processing stages. When designing a programming model providing access to the processing infrastructure, it is difficult to decide which processing stage at which level of granularity should be incorporated into this programming model, in terms of extension points. Candidates are the REQUESTOR and INVOKER which describe their range of processing in terms of demarshaled and marshaled core invocation data and results, their delivery and reception, and so on. In addition, the CLIENT PROXY and groups of remote objects could be incorporated into the design of such an extensible processing infrastructure. The latter is exemplified by *contexts* in Mono/R which are controlled execution environments for remote objects.

The design task for a versatile processing infrastructure in your middleware framework is further complicated by the *processing roles* (simply *role* hereafter) to be supported. In different component interaction and invocation styles, the middleware framework needs to take different roles (e.g., the *client* or *server* role in a REQUEST-REPLY invocation). A processing role is described by a set of processing operations and a sequencing of these operations which are specific to this role. The number and



characteristics of processing roles escalate with each component interaction style, invocation pattern, and their variants specific to a remoting technology family supported. This motivates to identify similarities and points of variations between processing roles in terms of processing operations and their dependency relations.

You also risk introducing unwanted design complexity through *invocation style emulation*: This problem describes the situation when the design of your BROKER incarnation is centred around a predominant invocation pattern, e.g., REQUEST-REPLY. From the perspective of evolvability, adding support for e.g. FIRE AND FORGET means to build an additional MESSAGE processing scheme on top of a processing infrastructure aligned to the predominant pattern. This strategy of emulation has been reported to bear the risk of introducing complexity. This extra complexity is due to static and dynamic crosscutting. On the one hand, there is the risk of increased code interlacing (see e.g. [34, 54]). On the other hand, the additional need for conditional branching (see, e.g., [70]) complicates the control flow inherent to such a processing infrastructure.

Therefore:

**Organise the invocation and MESSAGE processing of your middleware in terms of INVOCATION ASSEMBLY LINES. INVOCATION ASSEMBLY LINES are configurable and extensible chains of message processing tasks used both on the client (e.g., owned by the REQUESTOR) and the server side (e.g., owned by the SERVER REQUEST HANDLER). These processing chains are put in place for both request and reply MESSAGES. They are partitioned into processing stations which form the message processing flow. Each processing station contains one or more processing tasks in a specific order. The processing tasks are to be performed on certain invocation data items (e.g., MESSAGES or INVOCATION CONTEXTS) to be handled by the station. An INVOCATION ASSEMBLY LINE exposes a programming model to adjust the number of processing stations and offers different strategies to assign processing tasks to the available processing stations. INVOCATION ASSEMBLY LINES should be constructable at design and at activation time (e.g., through deployment descriptors), as well as changeable through the programming model at runtime.**

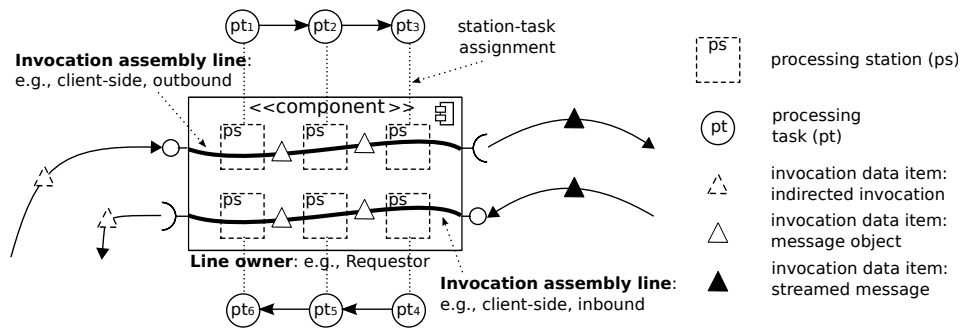


Figure 5: Conceptual sketch of INVOCATION ASSEMBLY LINES in a REQUEST-REPLY scenario

The INVOCATION ASSEMBLY LINE pattern describes the BROKER or a selection of its component patterns as a reconfigurable, flow-oriented processing infrastructure (see Figure 5). There are the following participants:

- **Processing station (ps):** An INVOCATION ASSEMBLY LINE consists of processing stations. They perform specific assembly, disassembly, and transformation operations on invocation data items. For each incoming or outgoing message (request or reply), invocation data items are transported along the INVOCATION ASSEMBLY LINE by passing them from station to station. Different stations can potentially process different kinds of invocation data items, The reconfiguration of an INVOCATION ASSEMBLY LINE means adding or removing processing stations

from a given configuration. In addition, a processing station can participate in several independent INVOCATION ASSEMBLY LINES. This permits us to create processing layouts beyond mere serial processing.

- **Processing task (pt):** Different kinds of invocation data items are subject to a set of related *processing tasks*. The tasks necessary to process these invocation data items vary considerably in different component interaction styles, dependency couplings, invocation styles, add-on infrastructure services, and remoting technology families. The kind of assembly, disassembly, or transformation operation described by a task is depending on the kind of invocation data item, such as core, contextual, or auxiliary invocation data. A particular processing task may be relevant for several MESSAGE kinds, possibly leading to reusing task descriptions and their implementations. Looking at the REQUEST-REPLY example in Figure 5, for instance, we find the processing tasks of marshaling and demarshaling the objectified MESSAGES.

Processing tasks are interrelated by an explicit ordering. The example in Figure 5 shows, for instance, that a canonical MESSAGE object representation must be constructed by the INVOKER or REQUESTOR before being marshaled into the actual MESSAGE by the MARSHALLER. The ordering structure of processing tasks are represented as a graph. This graph is formed by the *pt*-labelled vertices representing processing tasks, while edges denote their directed dependency relations. These precedence structures often reveal a *symmetry* of tasks within a given processing roles or between two processing roles. Looking at Figure 5, the marshaling (*pt<sub>2</sub>*) operation performed in the outbound direction mates with a demarshaling (*pt<sub>5</sub>*) operation on incoming invocation messages.

- **Invocation data item:** An *invocation data item* represents a workpiece to be processed by the processing stations grouped into an INVOCATION ASSEMBLY LINE. Relevant invocation data items are the introspection data of the indirected invocation and the result. They are then extracted from or transformed into MESSAGES. Further examples are INVOCATION CONTEXTS and context-related MESSAGES. Furthermore, auxiliary invocation data kinds such as INTERFACE DESCRIPTIONS are important to consider. The REQUEST-REPLY invocation example in Figure 5 shows MESSAGES representing invocation requests and invocation replies in various processing states, i.e., runtime information about an invocation, an invocation object, and a streamed invocation message. In addition, REMOTING ERRORS must be processed, if they occur. They must be signalled between the remoting endpoints.
- **Line owner:** INVOCATION ASSEMBLY LINES can be applied to structure the processing activities described by certain component patterns of the BROKER. For example, INVOCATION ASSEMBLY LINES are used by the REQUESTOR and SERVER REQUEST HANDLER as a part of their solution. We refer to the using framework components as *line owners* in the collaboration described by the INVOCATION ASSEMBLY LINE pattern. Also, we find INVOCATION ASSEMBLY LINES applied to INVOKER implementations to keep the invocation dispatch mechanisms reconfigurable and extensible.
- **Binding scope and times:** Binding, in this context, refers to the scopes and the times for activating and deactivating a particular INVOCATION ASSEMBLY LINE configuration. An INVOCATION ASSEMBLY LINE configuration describes a number of processing stations and the processing tasks assigned to them according to their precedence requirements.

Valid and relevant scopes for binding a processing configuration are single remote objects. If shared by many remote objects, the REQUESTOR and the INVOKER are appropriate scopes. A more general and reuse-driven approach is to bind a processing configuration to a CONFIGURATION GROUP. A CONFIGURATION GROUP [63] describes the configuration of the MESSAGE processing and the lifecycle management infrastructures shared by a set of remote objects. In addition, a CONFIGURATION GROUP can act as controlled execution environments for remote objects. Examples include *contexts* in Mono .NET Remoting (Mono/R; see [40]).

Besides the scope of enactment, various points in the lifecycle of a BROKER instance are candidates for defining the binding times to specify, activate, and deactivate a processing configuration. In the middleware frameworks reviewed, the deployment time of either remote objects or CONFIGURATION GROUPS has been chosen as the binding time. The issue of binding times also leads us to ask for an appropriate specification and deployment technique. Options include registration through the programming model at runtime (PASSIVE and ACTIVE REGISTRATION [37]) or by means of deployment descriptors [63] at activation time. The late negotiation and acquisition of such a configuration, i.e., at invocation time, can be a further requirement. For instance, invocation pattern variants (such as SOAP/1.2 or WSDL/2.0 Message Exchange Patterns, MEPs; [17, 25]) can be lazily negotiated between a client and a remote object. Either, the INTERFACE DESCRIPTION stipulating the invocation pattern is introspected just in time (e.g., in forms of dynamic invocation) or the invocation pattern is communicated through the INVOCATION CONTEXT. In either case, the INVOCATION ASSEMBLY LINE needs to be assembled and activated at runtime.

INVOCATION ASSEMBLY LINES organise the control and data flow for their line owners, such as the SERVER REQUEST HANDLER or REQUESTOR. There are two dimensions of control and data flow to consider: the *placement* and the *centralisation* of the control and data flow layout.

The *placement* of control and data flow information can either be extrinsic or intrinsic to the processing stations. An *extrinsic* placement refers to capturing the overall processing-related control flow information in dedicated runtime entities different to processing stations. In particular, relevant control and data flow information can be stored and managed by invocation data items. The INVOCATION CONTEXT is sometimes used for this purpose. An *intrinsic* placement challenges the above view by locating control and data flow information in the processing stations directly.

With regard to *centralisation*, the control and data flow layouts can be organised in a centralised or dispersed manner. A *centralised* INVOCATION ASSEMBLY LINE lays out the control and data flow in single, state-carrying entities different from the actual processing stations. For instance, the AxisEngine object-class as an essential component of this combined INVOKER and REQUESTOR variant in Axis2 stipulates the overall control and data flow centrally, in its `send` and `receive` operations. Conversely, a *dispersed* specification of the control and data flow allows for distributing responsibilities among several processing stations. In Mono/R, for instance, *sinks* realise this idea of dispersed control and data flow management for INVOCATION ASSEMBLY LINES. Each sink is only aware of its neighbour sinks, leaving the ultimate succession of sinks managed in a decentralised manner.

Combining these placement and centralisation strategies shows different effects on the targeted adaptability of processing behaviour, the locality of modifications and the perceived complexity of the processing infrastructure. This is particularly important when considering the need to provide introspective facilities upon the control and data flow to realise runtime and invocation time adaptability of INVOCATION ASSEMBLY LINES. Also, you may consider assigning one or multiple processing tasks to a given processing station. Different strategies of station-task assignments, placement, and centralisation as well as an assessment of their consequences are covered by the SINGLE- (pp. 22) and the MULTI-TASK PROCESSING STATIONS (pp. 24) patterns.

## Partial Processing Paths

Assume that you have decided to build or adapt a certain BROKER incarnation. The kinds of remote invocation (e.g., REQUEST-REPLY and FIRE AND FORGET RPCs) and the remoting styles (e.g., functional subsets of ICE [27] and WS [9, 39, 16, 17]) which this BROKER realisation is meant to support are given. Assume further that your BROKER-based framework is intended to serve as a surrogate for both client- and server-side applications. In order to be sure that the processing of MESSAGES is suitable for serving the given set of invocation patterns for the range of remoting styles and roles (i.e., client and server) supported, you must decide how to lay out the MESSAGE processing infrastructure. Invocation patterns (e.g., REQUEST-REPLY, FIRE AND FORGET) do not reveal insights on the different processing needs of MESSAGES within an invocation pattern realisation. Processing operations and sequencing rules between single operations (e.g., marshaling, MESSAGE construction, etc.) could be reused for realising different invocation patterns. Others might be conflicting in the context of a particular invocation pattern.

**Your invocation and MESSAGE processing infrastructure must allow to perform selected ranges of designed paths of processing operations alone. How can a processing infrastructure be designed out of a set of composable path sections which can be combined into a bound variety of processing paths?**

Consider an example: A processing scheme originally designed to realise REQUEST-REPLY invocation variants might be required to put only the request-specific range into effect to enact kinds of FIRE AND FORGET invocations (see also Figure 6). If, in a FIRE AND FORGET scenario, reply-specific processing steps would be scheduled, the responsible entities would, at least, have to operate on the control flow in a manner to be effectively skipped. The risk of excessive conditional branching would be the result, to give a single concrete example. Besides, maintainability is potentially reduced because these decision points which implement the processing line in a requested invocation pattern are squattered over several implementation entities. More generally speaking, this compositional flexibility is demanded by different types of one-way invocation, such as forms of synchronisation decoupled and non-blocking invocations (e.g., FIRE AND FORGET) as well as timely decoupled one-way acts in MESSAGING and PUBLISH-SUBSCRIBE [12].

Also, you might want to consider support for batch processing of invocations [59]. To realise batching support, you must define a processing scheme which permits the repetitive execution of single processing operations early in the lifecycle of remote invocations. Either the CLIENT PROXY or the REQUESTOR must be capable of accumulating invocation dispatches. Accumulation involves the incremental assembly and later disassembly of batch MESSAGES. Only upon signalling the end of the batch invocation (e.g., through an explicit *flush* operation), the REQUESTOR proceeds in further-processing the accumulated MESSAGE, i.e., its marshaling and delivery. In an inverse manner, the INVOKER has to disassemble the batch MESSAGE, perform the individual invocation dispatches, compile a reply MESSAGE, and, finally, have it streamed and returned. Hence, batching causes single processing steps to be repeatedly performed in a row (e.g., MESSAGE construction) while others are only scheduled once (e.g., marshaling).

In addition, you are often faced with the requirement of handling and even recovering from REMOTING ERRORS [63] as well as exceptional conditions when processing MESSAGES. This very requirement can be raised from different angles:

1. To begin with, REMOTING ERRORS can be raised at different locations (e.g., the client, server application or the transport logic). In particular, they can be issued from within the MESSAGE processing infrastructure, for instance, upon sensing exceptions when turning a MESSAGE object into a structured byte stream representation (e.g., an IIOP [45] message). Depending on the concrete type of REMOTING ERROR, they must be propagated back to the remote endpoint, i.e., the client application or its underlying BROKER, where the encapsulated REMOTING ERROR is

injected into the control flow. This involves processing REMOTING ERRORS as MESSAGES to be delivered to the remote endpoint. The processing steps for these REMOTING ERRORS and their sequencing are essentially similar to those of core invocation data (i.e., the request and reply data).

- Forms of reliable messaging (e.g., the error recovery model in ICE [27] or WS/RM [19]) presuppose that the MESSAGE processing infrastructure can automatically replay sequences of processing steps to realise quality constraints on MESSAGE delivery, e.g., at-most-once delivery guarantees. Replaying refers to performing varying ranges of processing steps depending on the point of exception or failure, possibly in a repeated manner.

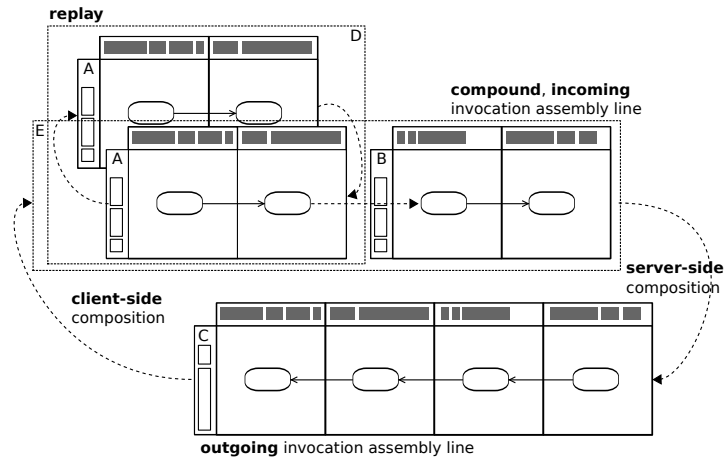


Figure 6: Composable processing of invocations

Therefore:

**Organise your MESSAGE processing infrastructure in terms of PARTIAL PROCESSING PATHS. Each PARTIAL PROCESSING PATH is realised by a dedicated INVOCATION ASSEMBLY LINE representing a set of functionally linked processing operations which recurs in different targeted invocation pattern realisations. As INVOCATION ASSEMBLY LINES are joinable, combine their instantiations for expressing different processing roles (client or server side) and invocation pattern implementations.**

PARTIAL PROCESSING PATHS promote the idea of structuring the processing infrastructure in smaller parts which can be reused by composing the actual processing paths as compounds from such building blocks. Each part groups processing tasks (e.g., marshaling, decryption, etc.) which are found recurring jointly in different invocation scenarios. By joint recurrence, we mean that they are applied in a sequence which is found stable for a number of invocation scenarios. Each part is represented by a dedicated INVOCATION ASSEMBLY LINE. An INVOCATION ASSEMBLY LINE can take the form of a compound (e.g., line E in Figure 6), assembled from other INVOCATION ASSEMBLY LINES (e.g., lines A and B in Figure 6). While the number of possible scenarios is certainly vast, you will find certain, characteristic decomposition strategies across current middleware examples:

- Per processing direction:* Provide INVOCATION ASSEMBLY LINES which represent processing directions, i.e., outward- and inward-directed invocations and MESSAGES. In Figure 6, there are the outgoing line C and the incoming line E. Such a decomposition strategy makes it convenient to express the client and server roles in the light of a given invocation patterns. As for REQUEST-REPLY invocations, the composition set (C, E) represents the client-side, the set (E, C) the respective server side of processing. If FIRE AND FORGET is requested, the processing infrastructure is limited to the line C at the client and the line E at the server side.

- *Per kind of invocation data:* Certain processing tasks appear shared between different types of invocation data items, while others are type-specific. Sets of shared processing tasks are then composed into compound INVOCATION ASSEMBLY LINES to handle a specific type of invocation data item. Important examples are REMOTING ERRORS, INTERFACE DESCRIPTIONS, and certain auxiliary MESSAGE kinds underlying advanced invocation patterns (e.g, notifications in WS/RM [19]).
- *Per lifecycle strategy:* Certain sections of the processing path are performed repeatedly. This is particularly important for strategies of failure recovery and invocation batching. The repetitive character of a set of processing steps qualifies these to form INVOCATION ASSEMBLY LINES which then are grouped into compound lines. In Figure 6, the component line A provides an example in the context of incoming invocations requests. This maps to a server-side infrastructure which is capable of recovering from errors encountered in early processing steps (e.g., transport and demarshaling).

Use PROCESSING SHORTCUTS (p. 20) to provide the links between the resulting INVOCATION ASSEMBLY LINES. For example, INVOCATION ASSEMBLY LINES operating on core invocation data should include PROCESSING SHORTCUTS to those responsible for possible REMOTING ERRORS. Make sure that the decomposition into PARTIAL PROCESSING PATHS is not hindered by a lack of genericity of the components realising the actual processing tasks. Consider a MARSHALLER which is bound to the canonical object representation of core invocation data (i.e., request and reply objects) and which is only capable of processing these. As a result, you cannot create a PARTIAL PROCESSING PATH to be reused for various data kinds (e.g., INTERFACE DESCRIPTIONS).

## Reconfigurable Processing Paths

When you plan to provide framework extensions to a middleware, it is important to verify that you can actually weave the extension behaviour into the framework and that the framework provides the necessary extension points to realise the add-on behaviour. Similarly, if your role is that of the framework developer, you will certainly find yourself in the situation that prospective extension developers express requirements on extension points to be exposed by the processing infrastructure. It is particularly hard to foresee the number and kind of extension points becoming necessary at the time of creating the processing infrastructure. Each step in a processing path can be turned into an extension point by refining it into a hook for INVOCATION INTERCEPTORS [63, 52]. However, not each processing step should be fixed as a hook. This patterns helps avoid predetermining a web of hooks at design time. Such a predetermination bears the risks of excessive resource consumption and an increased design complexity which reduces the communicability of the hooks and their interdependencies to extension developers.

**Providing support for extensions and add-on services plays a central role for the adoption and the further-development of a middleware framework. Unless you can add or remove extension points to the processing infrastructure for invocations and MESSAGES in a principled manner, the middleware will not fit unanticipated deployment scenarios while preserving its maintainability.**

A middleware framework, as a variant of an object-oriented application framework, is not meant to be a ready-made piece of software. Rather it is to be completed through integration by client and server applications. Integration also means to attach framework extensions, shared by a group of client or server applications. Designed processing paths must remain adaptable to cover a modified set of processing operations and changing dependency relations between them. This is particularly important regarding your framework's extension infrastructure which might be built around an INVOCATION INTERCEPTOR [63, 52] variant. From this perspective, processing operations are potential points of interception. In order to extend (or reduce) the INVOCATION INTERCEPTOR's reach, processing steps must be addable (or removable). Important examples are found in securing remote invocations because security-related, orthogonal extensions such as implementations of the Web Services Security Core Specification (WSS/Core; [42]) commonly operate on MESSAGES before and after core processing steps, such as demarshaling and marshaling.

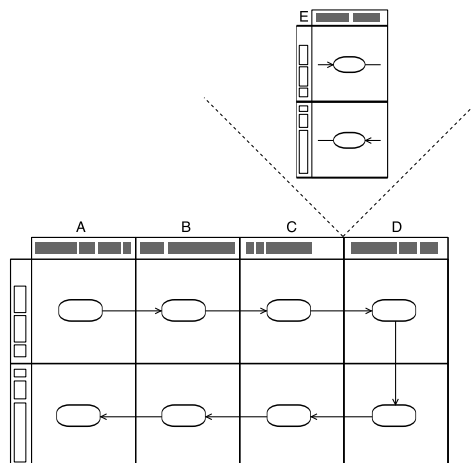


Figure 7: Extending processing paths

Therefore:

**Preserve RECONFIGURABLE PROCESSING PATHS by realising each processing path as an INVOCATION ASSEMBLY LINE. Allow to insert into or remove processing stations from these INVOCATION ASSEMBLY LINES. Alternatively, if the layout of processing stations is fixed, make sure that**

**processing stations can either be effectively discharged from all processing tasks (i.e., a de facto removal) or that they can be attached more than one processing task (i.e., a de facto insertion).**

At design time of your framework, make sure that you apply the PARTIAL PROCESSING PATH pattern to obtain an initial set of relatively robust INVOCATION ASSEMBLY LINES. They can then be exposed for refinement into RECONFIGURABLE PROCESSING PATHS. The sketch provided in Figure 7 shows a compound of two INVOCATION ASSEMBLY LINES to realise a client-side processing configuration for a REQUEST-REPLY invocation variant. In its initial configuration, there are four processing steps (i.e., A, B, C, D) and, thus, candidate extension points. By adding a processing station to each line, you obtain the further processing step E. Note that this exemplary insertion preserves the processing symmetry [63, 52], by considering two symmetric processing stations. This is, however, not strictly necessary. It would be equally possible to amend only one of the two INVOCATION ASSEMBLY LINES in Figure 7. For instance, if only logging of outgoing invocation data items was required, this would suffice as an extension point.

Arrange the protocol to reconfigure the processing paths in a way that adaptations to the processing path can be performed by the extension developers themselves. This avoids conflicts between reconfiguration requirements of different extensions. This requires respective hooks being activated and deactivated at configuration and runtime. Details for these issues of binding scope and binding time are treated by the INVOCATION ASSEMBLY LINE pattern (pp. 11).

The need for RECONFIGURABLE PROCESSING PATHS is exemplified by the scheduled design element of *dynamic phases* in Axis2 [5]. So far, Axis2 extension developers cannot revise and introduce their own set of phases, i.e., processing steps, through their CONFIGURATION GROUPS contributed. This, however, turned out critical because the global phase configuration must be adjusted or various variations thereof need to be shipped in order to support individual extensions. This breaks the fundamental idea of orthogonal extensibility and introduces an unwanted coupling between the core framework and its extensions. The planned design revision will allow for per-extension phases to overcome this limitation [30].





**lines and you can have the receiving line continue (or even complete) the invocation and message processing.**

The requirement of bypassing is realised by redirecting the control and data flow between two INVOCATION ASSEMBLY LINES. Details on how PROCESSING SHORTCUTS can be laid out and how these line-crossing processing stations can be constructed are dependent on the realisation variant of the INVOCATION ASSEMBLY LINE pattern used (see also Figure 8): If the layout of processing stations is global and fixed during runtime (i.e., the MULTI-TASK PROCESSING STATION pattern applies; pp. 24), PROCESSING SHORTCUTS can bridge between different processing steps. Consider the example in Figure 8: The outgoing activity of processing step C could redirect to the incoming activity of processing B. This is because processing stations can be made aware of each other. Conversely, if the station layout is not predetermined (i.e., the SINGLE-TASK PROCESSING STATION pattern applies; pp. 22), a cross-line station can only be created within a single processing step. Hence, the outgoing activity can only point to the incoming activity in step C (see Figure 8).

Use this pattern jointly with PARTIAL PROCESSING PATHS (pp. 15) to provide the necessary connectors between INVOCATION ASSEMBLY LINES specific to a processing direction and to a type of invocation data. The PROCESSING SHORTCUT pattern implies that, when applied along with RECONFIGURABLE PROCESSING PATHS (pp. 18), inserted processing steps are to be symmetric.

## Single-Task Processing Stations

You chose to adopt the INVOCATION ASSEMBLY LINE (pp. 11) pattern. It remains to select a strategy for laying out the processing stations and the assignment of processing task to these stations. This strategy is largely about where to place the control for the station layout and the task assignments; *and* whether the control and data flows are centrally managed. Your main objective is to ease the future addition of framework extensions (e.g., an encryption add-on) which come in form of CONFIGURATION GROUPS. Your middleware framework is required to facilitate the extensibility towards orthogonal add-on services (e.g., for securing the BROKER). Framework extensions risk entailing hidden interdependencies, when being deployed together, which can cause single extensions to fail unexpectedly.

### How can you realise an extensible processing infrastructure which minimises the risk of introducing hidden interdependencies between framework extensions?

To avoid unwanted interdependencies between extensions, it is recommended to decouple the add-on behaviour introduced by two extensions. To achieve this, you must balance two forces: the placement and the centralisation of the control and data flow in your processing infrastructure. Decoupling is best achieved through a decentralised specification of the control and data flow layout, i.e. each extension remains unaware of other, currently active ones. Recording and storing control flow information should also be kept within an extension's realm (i.e., by an intrinsic placement of information).

Following from this, an extension is inserted without full knowledge of the global configuration. Extension developers will not have to be concerned with the global state of the processing infrastructure, the resulting processing behaviour can only be stated (and verified) upon runtime. In other words, you must design your processing infrastructure according to a strict LAYERS structure.

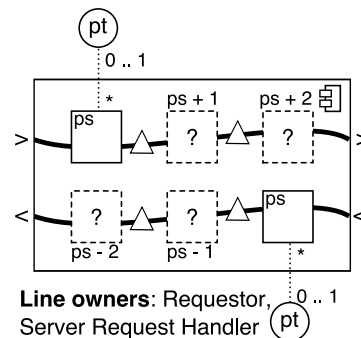


Figure 9: Single-task assignments, variable, and decentralised station layout

Therefore:

**Arrange the INVOCATION ASSEMBLY LINES with a variable number of processing stations which chain themselves in a decentralised manner, i.e., through forward references owned locally by each processing station. Attach a single processing task to each station. RECONFIGURABLE PROCESSING PATHS are realised by varying the number and positioning of the processing stations alone.**

In some detail: You can learn the essentials of this INVOCATION ASSEMBLY LINE variant from Figure 9 which shows two INVOCATION ASSEMBLY LINES. While they are made of a number of processing stations, each being assigned a single processing task, their exact number and quality is not known to any central entity which is extrinsic to the INVOCATION ASSEMBLY LINES themselves. Most importantly, the line owners do not manage, nor are they aware of the station layout to be enacted when processing an invocation. Rather, each processing station ( $ps$ ) points to a successor station ( $ps + 1$ ), if available. In turn, inserting processing stations on demand requires identifying the right location in terms of its direct antecedent station as the registrar. Therefore, the control and data flow is the result of a decentralised composition process.

This has important consequences, especially for realising PROCESSING SHORTCUTS (pp. 20) and RECONFIGURABLE PROCESSING PATHS (pp. 18). Shortcuts can basically be achieved by resolving the targeted processing step by following the forward references along the station chain. However, the target station must be known to the extension developer and, thus, the source processing station. Also reconfigurations of the station chain happen under conditions of decentralised flow control. It is possible to allow processing station may insert or remove subsequent ones by tracking the forward references. Note, however, that the absolute positioning of a processing stations is not guaranteed in the presence of multiple, active framework extensions which manipulate this station layout.

This SINGLE-TASK PROCESSING STATION strategy, incarnating a LAYER variant for the processing infrastructure, is found for Mono .NET Remoting (Mono/R, see [40, 50]) and Mono Olive (Mono/O; see [41]). Details are given in Section 7 on known uses.

## Multi-Task Processing Stations

You are in a situation in which invocation pattern variants (e.g., an in-only MEP [17, 25]) are negotiated through INTERFACE DESCRIPTIONS or, even more lazily, through the INVOCATION CONTEXT. Once registered, this invocation pattern is mapped to a configuration to be applied to the processing infrastructure. Such a configuration involves a specific set of INVOCATION ASSEMBLY LINES. Also, you are required to track the state of the underlying MESSAGE exchanges precisely, e.g., in order to verify completion and failure conditions.

**The middleware framework must be able to devise arrangements of INVOCATION ASSEMBLY LINES which can be monitored for specified events (i.e., completions, failures, and notifications) and for processing states. At the same time it must not lose its capability of forming PARTIAL PROCESSING PATHS. How can a processing infrastructure, which appears predetermined in terms of processing steps and operations covered, preserve the adaptability still required?**

Similar to the SINGLE-TASK PROCESSING STATIONS (pp. 22), you must review the design dimensions of organising the control and data flow in the light of the above requirements: the centralisation and the placement. A centralised organisation of the processing infrastructure fits the monitoring requirement better than a decentralised one. Therefore, the INVOCATION ASSEMBLY LINES should be managed and tracked by a central *controller entity*.

Where to place the flow information (e.g., processing state flags), which is used to monitor and regulate the control and data flow, is more difficult to answer. An intrinsic placement would store this kind of control information with the core elements of the processing infrastructure, i.e. processing stations. In that sense, they would turn stateful. Statefulness, however, limits the reusability of processing paths in the sense of PARTIAL PROCESSING PATHS (pp. 11). An extrinsic placement would have this information bits managed with the invocation data items, e.g., the INVOCATION CONTEXT. This, however, makes it more challenging to provide for the introspection of the processing state from the angle of the controller entity.

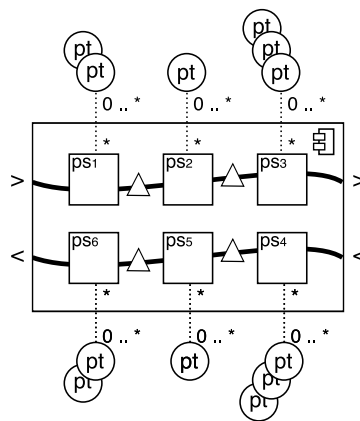


Figure 10: Multi-task assignments, fixed, and centralised station layout

Therefore:

**Devise a fixed layout of processing stations which is then shared by different INVOCATION ASSEMBLY LINES. As for fixing, arrange for a manager entity which stores the processing layout as well as the task assignment. Provide for a protocol to access and query the station layout. Make processing stations capable of managing and performing multiple processing tasks. This permits you to enforce dependency constraints between processing tasks without stations being arrangeable.**

The solution involves a fixed number of multi-task processing stations: Figure 10 shows two INVO-

CATION ASSEMBLY LINES, each containing three explicitly labelled processing stations (i.e.,  $ps_1, ps_2$ , etc.). The processing stations are managed by and their participation in realising the two INVOCATION ASSEMBLY LINES is registered with the line owners, either the REQUESTOR or the SERVER REQUEST HANDLER. Thus, the current processing state (in the light of a ruling invocation pattern) can be introspected at any time from the line owners as manager and monitoring entities. The line owner are also responsible to serve with a task registration and introspection interface to client applications. To give the processing infrastructure the needed flexibility, you must allow for fine grained techniques for assigning multiple processing tasks to this set of processing stations. The assignment mechanism needs to be versatile enough to express the precedence constraints as a particular ordering upon assigning tasks. This strategy is often found realised by or deeply integrated with INVOCATION INTERCEPTORS.

Known uses of this INVOCATION ASSEMBLY LINE variant are Apache Axis2/Java (Axis2; see [5, 46, 21]) and Apache CXF (CXF; see [6]). These two frameworks lay out a predetermined, though reconfigurable, arrangement of processing stations referred to as *phases* in both cases. These layouts can only be accessed through and are effectively managed by central manager entities. While these layouts comprise default sets of processing stations, the manager delivers predefined subsets thereof for forming special-purpose INVOCATION ASSEMBLY LINES on demand, e.g., for handling REMOTING ERRORS. In these two variants, the processing stations serve for interception points. So, processing tasks are realised through CXF's and Axis2's INVOCATION INTERCEPTORS. The registration protocol for INVOCATION INTERCEPTORS permits the developer to assign processing tasks in a fine-grained manner, including positioning relative to other INVOCATION INTERCEPTORS and absolute positioning rules. The latter is important to provide guarantees that core processing tasks are executed at the appropriate positions. Details follow in the subsequent section.

## 6 Motivating Example Resolved

Let us return to the motivating example considered in Section 4.2 and let us briefly walk through applying the small pattern language presented here to structure this design decision space. As framework developers, the example confronts us with the requirement of providing support for optionally securing end-to-end message delivery. This requirement was evaluated against three possible strategies: the refinement of the MARSHALLER or the REQUEST HANDLER components alone, as well as an INVOCATION INTERCEPTOR variant. Each of these approaches reflects requirements and forces captured by the RECONFIGURABLE PROCESSING PATHS pattern (see Section 5, pp. 18). An application which requires delivery encryption should be able to add both encrypting and decrypting operations at processing stages which provide read and write access to the streamed outgoing and the streamed incoming messages. In addition, such an application might act both as client- and server application so that this end-to-end encryption service must be realised for the client- and server roles taken by our middleware framework. Also, this framework extension should only be activated when being used by this application. That is, this application-specific extension should not interfere with other remoting applications integrating our middleware framework. Finally, the extension should be applicable under the entire range of possible service configurations, e.g., different marshaling and transport strategies.

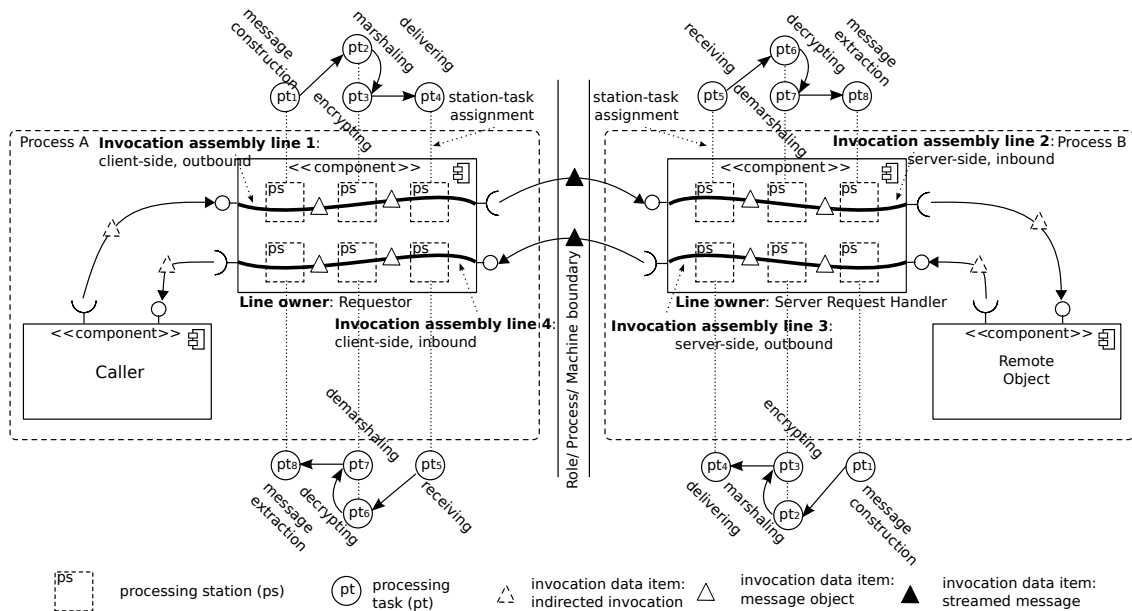


Figure 11: Using INVOCATION ASSEMBLY LINES in a REQUEST-REPLY scenario with message-level encryption

Adopting a RECONFIGURABLE PROCESSING PATHS variant implies applying the INVOCATION ASSEMBLY LINE pattern (see Section 5, pp. 18). Hence, we conceptualise and design the message processing infrastructure in terms of processing stations and processing tasks. Figure 11 provides an exemplary instantiation of the INVOCATION ASSEMBLY LINE pattern. We devise outbound and inbound INVOCATION ASSEMBLY LINES for both the client- and server-side processing infrastructure of our framework. Each INVOCATION ASSEMBLY LINE consists of three processing stations, reflecting the elementary life-cycle stages of the invocation requests and replies processed, such as objectified, marshaled, and delivered (or, received). As for the basic processing tasks in a secured REQUEST-REPLY invocation, as shown in Figure 11, both the client-side, outbound and the server-side, outbound INVOCATION ASSEMBLY LINES perform a set of four processing tasks: message construction ( $pt_1$ ), marshaling ( $pt_2$ ), encrypting ( $pt_3$ ), and delivering ( $pt_4$ ). Similarly, the client-side, inbound and the server-side, inbound INVOCATION ASSEMBLY LINES are characterised by the succession of receiving ( $pt_5$ ), decrypting ( $pt_6$ ), demarshaling ( $pt_7$ ), and extracting

(*pt<sub>s</sub>*) invocation data from incoming MESSAGES. This sharing of processing tasks between the client and server side, though in different configurations, reflects a certain processing role symmetry. This symmetry mates with the ideas of freestanding MARSHALLER and PROTOCOL PLUG-IN [63, 53] components which can be reused to realise either framework role.

In a next step, we plan to support secure delivery under invocation patterns other than REQUEST-REPLY. Also, secure delivery requires a centralised monitoring of the ongoing processing operations. Hence, we apply a strategy of MULTI-TASK PROCESSING STATIONS (see Section 5, pp. 24). The second processing stations in each of the four INVOCATION ASSEMBLY LINES is responsible for performing two tasks. The sequencing within the two resulting pairs of tasks (i.e., marshaling/encrypting and decrypting/demarshaling) realises our motivating example of a Security provider (see also Figure 4). Again, the symmetry between the client- and server-side roles is preserved. When applying a MULTI-TASK PROCESSING STATIONS strategy, the processing station layout is not only fixed, but also centrally controlled. That is, the REQUESTOR and SERVER REQUEST HANDLER components as owners of the INVOCATION ASSEMBLY LINES store the station layout, provide a management interface to maintain the station configuration as well as the processing task assignments of the stations, and organise the dispatch upon the processing stations. This strategy is commonly implemented by an INVOCATION INTERCEPTOR variant (see, e.g., [53]). By adopting the MULTI-TASK PROCESSING STATIONS pattern, we obtained a design which facilitates realising PARTIAL PROCESSING PATHS (see Section 5, pp. 18).

## 7 Known Uses

INVOCATION ASSEMBLY LINES are found in a selection of existing middleware frameworks: OpenORB [58], Mono .NET Remoting (Mono/R; see [40]), Mono Olive (Mono/O, [41]), Apache Axis2/Java (Axis2; see [5, 46, 21]), and Apache CXF (CXF; see [6]). In the following, we reflect on instantiations of member entities (i.e., processing stations, processing tasks, line owners, etc.) and their interactions characteristic for the INVOCATION ASSEMBLY LINE pattern.

### 7.1 OpenORB

The Java-based CORBA implementation OpenORB realises an INVOCATION ASSEMBLY LINE based on a TEMPLATE CLASS [49, 53] collaboration between *client* and *server managers*, on the hand, and the CORBA-specific COMMAND MESSAGE objects (i.e., `ClientRequest` and `ServerRequest`; see [29, 13]), on the other hand.

OpenORB realises the SINGLE-TASK PROCESSING STATION variant of the INVOCATION ASSEMBLY LINE pattern, characterised by (a) a weak distinction between processing stations and tasks and (b) a rigidly fixed number of processing stations or tasks. **Processing stations** are realised as operation records of the COMMAND MESSAGE object-classes. Each COMMAND MESSAGE stipulates a signature interface with a set of deferred operations, that is, HOOK METHODS to be completed by protocol-specific implementations (e.g., RMI [57], IIOP [45], etc.). These operation implementation take the role of **processing tasks**. Notably, processing tasks pertaining to protocol-specific MARSHALLERS can be assigned this way. The processing flow is laid out by the client and server managers in terms of an abstracted call sequence of HOOK METHODS.

The flexibility of task assignments, however, is limited to the possible method combinations along a hierarchy of object-types compliant to the COMMAND MESSAGE interfaces. The task precedence is bound to the call sequence of deferred operations implemented in the line owners and is, therefore, fixed during runtime. The **invocation data items** actually processed are equally encapsulated by the COMMAND MESSAGE objects. The central ORB object-classes act as **line owners**. The major limita-



tions of this implementation variant of SINGLE-TASK PROCESSING STATIONS are also discussed in [53].

## 7.2 Mono .NET Remoting

In Mono/R as a F/LOSS implementation of Microsoft .NET Remoting, the INVOCATION ASSEMBLY LINE variant takes a dominant position in the overall design. Processing tasks are realised by *sinks* [63, 50]. They incorporate properties of both INVOCATION ASSEMBLY LINES and INVOCATION INTERCEPTORS which appear heavily interwoven.

As for INVOCATION ASSEMBLY LINE, the concepts of **processing station** and **processing tasks** fuse to a large extent. They are embodied as so-called *message*, *formatter*, and *channel sinks*. They are supported by *sink providers* or *sink contributors*. They provide an interface which is used by the **line owner** elements, i.e., *contexts* and *channels*, to establish chains of sinks. Each sink provider can only provide for registering a single sink. Message sinks aim at processing objectified representations of MESSAGES while channel sinks operate on streamed forms of MESSAGES. Therefore, the **invocation data items** targeted are clearly MESSAGES. Message and channel sinks are linked in chains to represent what we identified as instantiations of INVOCATION ASSEMBLY LINE variants. Sinks are organised as forward-linked lists and, hence, represent an intrinsic and dispersed design of SINGLE-TASK PROCESSING STATIONS.

Line ownership and **binding scopes** are interdependent. On the one hand, these are variants of CONFIGURATION GROUPS referred to as *contexts*. On the other hand, they cover the scope of PROTOCOL PLUG-INS known as *channels*. Contexts [50] represent controlled execution domains for remote objects and, therefore, participate in realising CONFIGURATION GROUPS. Contexts allow for attaching context-specific activation, lifecycle management, and extension behaviour to remote objects. These are all realised in terms of message sinks. At the provider side, message sinks act as ultimate invocation dispatchers (i.e., the `StackBuilderSink`) and lifecycle managers (i.e., the `LeaseSink`; see [63]). Channels provide the actual core BROKER functionality to the context-bound REMOTE OBJECTS. Provided that target remote objects are collocated, a special-purpose channel (i.e., the `CrossContextChannel`) offers a shortcut invocation path between local contexts. If machine boundaries need to be passed, channels represent CONFIGURATION GROUPS which install and set up PROTOCOL PLUG-INS for the CLIENT and SERVER REQUEST HANDLERS. For the realm of channels, the processing tasks represent core behaviour described for the BROKER and its essential component patterns: We, roughly, find correspondences of pre-, post-, and marshaling phases for either invocation direction. They are realised over the transition from message over formatter to channel sinks [63, 50].

## 7.3 Mono Olive

Two INVOCATION ASSEMBLY LINE solution variants are found in Mono/O, a F/LOSS implementation of the Microsoft Windows Communication Foundation (WCF), formerly known as *Indigo*. While the first, centred around the design elements of *channels*, is most visible and relevant for the overall design, the second is a small-scale, yet illustrative example hidden in the internals of Mono/O's INVOKER instantiation.

This first INVOCATION ASSEMBLY LINE occurrence is encountered in the collaboration of *channels* and *channel managers*. This collaboration realises the SINGLE-TASK PROCESSING STATION variant of the pattern. Channels represent **processing stations** that are configured to perform a single-only **task** on the invocation data items processed, i.e., MESSAGE operations. Stock channels that come with Mono/O are limited to performing the role of MARSHALLERS (i.e., `MessageEncoders`) and CLIENT or SERVER REQUEST HANDLERS. Certain channels, such as the REQUESTOR-specific `ClientRuntimeChannel`, are core and non-optional elements of a Mono/O setup. Channels form

INVOCATION ASSEMBLY LINES in terms of a *channel stack* or rather a forward-linked list of channels. Internally, they are referred to as *layered channels* in such a configuration.

The idea of describing processing configurations underlying certain invocation patterns, which is central to the INVOCATION ASSEMBLY LINE pattern, is clearly visible in Mono/O's channels. There are different kinds of channels which express three different invocation patterns in terms of their interfaces; REQUEST-REPLY, FIRE AND FORGET, and a PEER-TO-PEER variant. Depending on the enclosing invocation pattern, channels describe two INVOCATION ASSEMBLY LINES in terms of operation records, one for blocking and one for non-blocking scenarios. At runtime, only one INVOCATION ASSEMBLY LINE configuration manifests effectively.

The role of **line owners** is taken by the channel managers which also realise different kinds CLIENT PROXIES. The **invocation data items** processed are the object representations of MESSAGES. The **binding scope** is described by so-called *bindings*, Mono/O's taste of CONFIGURATION GROUPS. They provide custom channel managers and channels to realise certain remoting and transport protocols. They are enacted either through a programming model or by deployment descriptors.

Internally, the INVOKER incarnation of Mono/O which is referred to as `RequestProcessor` realises a second variant of the INVOCATION ASSEMBLY LINE pattern. Its processing infrastructure is organised as a chain of `ProcessorHandlers` that can be compressed or extended to describe the responsibilities of the INVOKER as a configurable set of processing tasks. Currently, this set describes the tasks of dispatching the invocation upon the servant addressed and triggering the provider-side INVOCATION INTERCEPTORS. This second occurrence of the INVOCATION ASSEMBLY LINE is an example of a MULTI-TASK PROCESSING STATION implementation.

## 7.4 Apache Axis2

Another implementation variant of the MULTI-TASK PROCESSING STATIONS pattern is found in Axis2. It is built around a strong conceptual discrimination between processing stations and processing tasks. **Processing stations** are referred to as *phases* that are laid out in the global deployment descriptor; therefore, phases are defined for the scope of the entire Axis2 BROKER. More recently, the addition of phases through extension modules, referred to as *dynamic phases*, has been considered. **Processing tasks** are represented by Axis2's INVOCATION INTERCEPTORS, i.e., *handlers*, which are organised as *modules*. Modules and per-module (i.e., dynamic) phases foster the idea of the INVOCATION ASSEMBLY LINE pattern. Processing stations are organised into different kinds of INVOCATION ASSEMBLY LINES which are also specified at deployment time as so-called *flows*. The number of flows is predetermined and restricted. Axis2 distinguishes between flows for inward and outward bound invocation data (i.e., in- and out-flow), and in- and outward bound REMOTING ERRORS (i.e., in- and out-fault flow). Each flow manifests as an ordered set of phases. This represents an example of an extrinsically and centrally organised control flow.

The **data item** processed is the INVOCATION CONTEXT which comes as a composite element in Axis2, including the per-interaction, per-operation, per-service contexts. The INVOCATION ASSEMBLY LINES are attached to the combined REQUESTOR and INVOKER entity in Axis2, i.e., the `AxisEngine`, as the **line owner**. In Axis2, we consider the INVOCATION ASSEMBLY LINE to be found in the inter-workings of flows, phases, modules, and, finally, handlers; once per-module phases are fully supported, the reach of this INVOCATION ASSEMBLY LINE variant will be substantially extended.

## 7.5 Apache CXF

A further MULTI-TASK PROCESSING STATIONS instantiation comes with Apache CXF. **Processing stations**, again first concepts in terms of *phases*, are solely deployed through PASSIVE REGISTRATION [37] upon initialisation time. Custom phase definitions would have to be provided by injecting dedicated *phase managers*. **Processing tasks** are modelled and implemented as `PhaseInterceptors` which can be assigned to (a) multiple processing stations and (b) each phase can be assigned multiple tasks. When assigning a task set to a processing station, the configuration strategy permits to specify relative ordering for tasks handled by the same processing station.

CXF organises its phases into two major INVOCATION ASSEMBLY LINES reflecting inbound and outbound directions, respectively. Each **line owner**, i.e., the REQUESTOR and INVOKER, has a pair of inbound and outbound lines. This leaves aside lines for handling REMOTING ERRORS. REMOTING ERRORS are processed by two dedicated INVOCATION ASSEMBLY LINES which branch from the main INVOCATION ASSEMBLY LINES. Due to the dependency on phase managers, this INVOCATION ASSEMBLY LINE variant is extrinsically mastered and centrally organised. While the INVOCATION ASSEMBLY LINES are bound to their owners, their task assignments can be scoped in a fine grained manner. As for the **binding scope**, `PhaseInterceptors` are registered for either the global, per-binding, per-service, per-client, or per-endpoint scope. The **invocation data items** processed are MESSAGES only.

## 8 Discussion

The pattern language for INVOCATION ASSEMBLY LINES assists in (a) identifying processing commonalities for different BROKER roles and in (b) revealing the structural equivalence of central collaborators such as the REQUESTOR and the INVOKER. Processing tasks specific to each role are fully qualified by the representational form of invocation data (i.e., kinds and strategies of marshaling and demarshaling, canonical forms of objectified representation, etc.), their processing directions (inward, outward) and the MESSAGE kinds to process. Therefore, the pattern language suggests a data flow view [8] of the middleware. It aims at describing a middleware design as a series of transformations on invocation-related data. From this angle, we aim at identifying design elements that are responsible for the transformations, the elements actually transformed, and the quality of the transformations applied. The conceptual decomposition into atomic concepts such as processing stations organised in processing lines, tasks, invocation data items, and binding scopes permits to express complementary roles (e.g., consumer/provider, publisher/subscriber/notifier, etc.) based on a shared vocabulary.

When designing and developing a middleware framework, you are often required to support more than one remoting technology based on a shared invocation and MESSAGE processing infrastructure. Web Services (WS; [9, 39, 16, 17]), the Common Object Request Broker Architecture (CORBA; [45]), Java Remote Method Invocation (Java RMI; [57]), Java Messaging Service (JMS; [56]), as well as proprietary and ad hoc styles introduce important idiosyncrasies and specify control and data flow requirements which are potentially conflicting when built on top of such a shared infrastructure. Resulting design forces are addressed by the SERVICE ABSTRACTION LAYER [62, 64, 68] pattern. Such an intermediate infrastructure distinguishes between the BROKER core infrastructure and possibly multiple *frontend channels* which mediate invocations in certain remoting styles and technology families. This shields either side, the BROKER core and the frontend channels, from details orthogonal to their concerns. Each frontend channel is realised by a CONFIGURATION GROUP [63]. CONFIGURATION GROUPS make use of the facilities offered by the INVOCATION ASSEMBLY LINE pattern to adjust the processing infrastructure for their needs and assign their processing tasks, represented by a proprietary set of MARSHALLERS, PROTOCOL PLUG-INS, and INVOCATION INTERCEPTORS, accordingly.

In exemplary detail: You might plan to comply with the CORBA [45] or JAX-WS [15] speci-

fications, beyond the core invocation and MESSAGE handling (e.g., MESSAGE formats). In this, you are expected to implement CORBA-specific *and* JAX-WS-specific INVOCATION INTERCEPTOR semantics, i.e., CORBA’s portable interceptors and JAX-WS’s handlers. This poses important design problems: Do you foresee two widely independent INVOCATION INTERCEPTOR designs co-existing in your framework? Or, do you plan to provide a common INVOCATION INTERCEPTOR instantiation which is suitable for realising portable interceptors and handlers on top? This design problem is aggravated because CORBA’s and JAX-WS’s INVOCATION INTERCEPTOR instantiations are situated in predefined *failure recovery schemes*. These recovery models (informally labelled “flow stack model” in the family of CORBA specifications [45, Section 1.6.4.3] or “handler execution model” in JAX-WS [15, Section 9.3.2]) define that special-purpose points of interception are to be enforced (in a particular ordering) once an exceptional condition is sensed. This allows developers to foresee limited recovery or, at least, cleanup tasks. INVOCATION ASSEMBLY LINES help understand and realise such deviating control and data flow designs.

A tentative survey of existing middleware frameworks gives credence to this problem statement related to SERVICE ABSTRACTION LAYERS. For example, both Mono/R [40] and Apache CXF [6] provide CORBA frontend channels (i.e., Mono/R’s IIOP *channel* and CXF’s CORBA *binding*). As for specification-compliant INVOCATION INTERCEPTORS, CXF does not only provide its framework-specific INVOCATION INTERCEPTOR variant (i.e., phase interceptors), but also JAX-WS handlers.

We want to recapitulate the relationship between INVOCATION ASSEMBLY LINE and INVOCATION INTERCEPTOR, commonly found heavily interwoven in known uses of these two patterns. We may attempt to discriminate between their matters by looking at established kinds of pattern relationships; we limit ourselves to the usage relationship as identified by [43].

- INVOCATION ASSEMBLY LINE *uses* INVOCATION INTERCEPTOR: We aim at providing a SERVICE ABSTRACTION LAYER and devise CONFIGURATION GROUPS based thereupon. A variant of INVOCATION ASSEMBLY LINE uses INVOCATION INTERCEPTORS to enforce the decoupling of processing stations and processing tasks. In a straightforward reading, INVOCATION ASSEMBLY LINES assign a single task to each processing station. However, more complex task precedence structures and the need for organising processing tasks in an atomic and modular manner require processing stations to take responsibilities for several interdependent tasks. This task assignment strategy can be realised by devising each processing station as a point of interception, and, therefore, dispatcher for INVOCATION INTERCEPTORS. Tasks, in turn, are realised as concrete INVOCATION INTERCEPTORS to be registered with a particular processing station. It should be possible to express relative orderings of the task-representing INVOCATION INTERCEPTORS to be able to enforce more complex task precedence constraints. Axis2 and CXF provide occurrences of this relationship.
- INVOCATION INTERCEPTOR *uses* INVOCATION ASSEMBLY LINE: A realisation of the INVOCATION INTERCEPTOR patterns calls for a versatile or extensible set of points of interception. As prominently stated in [52, p. 137], designing an adequate model domain of interception points which anticipates the manifold requirements of integrating applications and framework extensions is non-trivial. There is both the risk of devising too limited or too bloated a number of interception points. The INVOCATION ASSEMBLY LINE pattern can help balance these forces by allowing to vary the number of interception points (i.e., processing stations) as needed. This can be used to organise the registration and dispatch of INVOCATION INTERCEPTORS in a more flexible manner.

Known uses of both the INVOCATION ASSEMBLY LINE and INVOCATION INTERCEPTOR patterns exemplify this use relationship, e.g., *dynamic message sinks* in Mono/R [40] and *message inspectors* in Mono/O [41]. The pattern story on Axis2 in [63, pp. 238] provides an outlook on shifting the responsibility for laying out the model of interception points into the INVOCATION INTERCEPTORS themselves by means of an AspectHandler.

You may think of these two patterns as being at either end of a continuous spectrum. In its strictest variations, the INVOCATION INTERCEPTOR pattern has been documented as a hooking technique that preserves orthogonality to the surrounding invocation infrastructure of the BROKER [63, p. 130]. Add-on functionality provided by INVOCATION INTERCEPTOR is allowed constrained access to the overall BROKER state only. This characterisation applies to CORBA's *portable interceptors* and certain uses of INVOCATION INTERCEPTOR in Mono/R, i.e., *dynamic message sinks*. The more this orthogonality and sanity requirements are loosened, the more appropriate is the INVOCATION INTERCEPTOR as a solution part of the INVOCATION ASSEMBLY LINE pattern.

The inter-workings between the INVOCATION ASSEMBLY LINE and INVOCATION CONTEXT patterns must be considered thoroughly. Both as an argument-passing strategy and as a participant in the INVOCATION INTERCEPTOR [31, 65], the INVOCATION CONTEXT already serves the purpose of capturing processing information. By representing an invocation's compositional layout in the INVOCATION CONTEXT, e.g., by storing MESSAGES in different processing states, the necessary processing tasks may be inferred from the compositional state of the INVOCATION CONTEXT. The INVOCATION CONTEXT's composition as a state compound allows for varying processing behaviour.

Alternatively, the INVOCATION CONTEXT can be used to realise a FLAGS FOR STATES [26] strategy. If the processing model throughout the BROKER or parts thereof, i.e., the INVOCATION INTERCEPTOR infrastructure, is organised around state flags, the INVOCATION CONTEXT offers itself as a delivery vehicle. The state flags are accessed at important decision points and used to realise conditional branching. The use of the INVOCATION CONTEXT with state flags is an example of a central and extrinsic organisation of processing behaviour. Considering the INVOCATION CONTEXT as a state compound represents a dispersed and extrinsic approach.

Applying the INVOCATION ASSEMBLY LINE pattern comes with certain advantages and drawbacks. Major advantages result from an increased level of separating between concerns of developing a framework as an intentionally incomplete, cross-domain infrastructure and concerns pertaining to domain-specific framework extensions. Drawbacks can be explained by the increased design complexity, caused by adding a further piece of abstracted-general design to your framework. Besides, we may encounter negative effects on runtime qualities due to more extensive resource usage.

By specifying custom configurations for processing stations, tasks, task assignments, and layouts for the so-realised INVOCATION ASSEMBLY LINES, developers of framework extensions can achieve necessary behavioural variations through a piece of abstracted-general design and a dedicated programming model. This facilitates adding support for component interaction styles and invocation patterns not anticipated initially. The division of development labour, in particular between framework and extension developers, can be deepened.

The ability to adjust the processing infrastructure and the decoupling of processing stations from tasks permits to reuse existing processing facilities for developing new CONFIGURATION GROUPS. Reuse candidates are MARSHALLERS and INVOCATION INTERCEPTORS bundled with existing ones. More generally, INVOCATION ASSEMBLY LINES allow to capture symmetry and asymmetry found crosscutting the BROKER pattern compound decomposed into responsibility-based LAYERS [8]. Components residing at each layer exhibit similar or comparable requirements on the processing infrastructure. INVOCATION ASSEMBLY LINES can be used to express these commonalities and refine variations as a composition and reconfiguration problem. Framework reuse is therefore fostered by an improved composability.

There are important tensions caused by potential gains and losses in modular comprehensibility. On the one hand, actual responsibilities for processing invocation data are more clearly separated from organising them in a particular flow of processing tasks. This permits to reason about MARSHALLER, PROTOCOL PLUG-INS, etc. in a modular manner. In addition, otherwise squattered decision points which shape the control and data flow in the processing infrastructure can be concentrated at certain

places of control. At the same time, the modular self-sufficiency of CONFIGURATION GROUPS is potentially reduced. The INVOCATION ASSEMBLY LINE pattern promotes the idea of expressing a variety of processing task dependencies based on a common infrastructure of processing stations and explicit behavioural models. The expressible variety risks burdening the development of framework extensions with the extra need for scrutinising the subtle details of specifying INVOCATION ASSEMBLY LINES.

## Acknowledgments

We would like to thank our EuroPLoP 2009 shepherd Didi Schütz for his constructive and insightful feedback on this paper. Thanks are also due to the participants of the writer's workshop F for providing substantial feedback: Anjali Das, Veli-Pekka Eloranta, Arto Juhola, Farah Lakhani, Marko Leppänen, Ville Reijonen, Martin Wagner, and Tim Wellhausen.

## References

- [1] N. Abu-Ghazaleh, M. J. Lewis, and M. Govindaraju. Differential Serialization for Optimized SOAP Performance. In *Proceedings of the 13th International Symposium on High Performance Distributed Computing (HPDC)*, pages 55–64, Honolulu, Hawaii, June 2004.
- [2] L. Aldred, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. On the Notion of Coupling in Communication Middleware. In Meersman et al. [38], pages 1015–1033.
- [3] N. Allen. Asymmetry Between Listeners and Factories. Discussion Blog [Internet], [unknown location] : Nicholas Allen, 2006 Oct - [cited 2009 June 3], Available from: <http://blogs.msdn.com/drnick/archive/2006/10/25/asymmetry-between-listeners-and-factories.aspx>, 2006.
- [4] D. Andresen, D. Sexton, K. Devaram, and V. P. Ranganath. LYE: A High-Performance Caching SOAP Implementation,. In *Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)*, pages 143–150, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
- [5] Apache Foundation. Apache Axis2/Java - Next Generation Web Services. <http://ws.apache.org/axis2/>, last accessed: October 13, 2008.
- [6] Apache Foundation. Apache CXF: An Open Source Service Framework. <http://cxf.apache.org/>, last accessed: October 13, 2008.
- [7] P. Avgeriou. Run-time Reconfiguration of Service-Centric Systems. In *Proceedings of 11th European Conference on Pattern Languages of Programs (EuroPlop 2006)*, Irsee, Germany, 2005.
- [8] P. Avgeriou and U. Zdun. Architectural Patterns Revisited – A Pattern Language. In *Proceedings of 10th European Conference on Pattern Languages of Programs (EuroPlop 2005)*, pages 1 – 39, Irsee, Germany, July 2005.
- [9] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thattle, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. W3C Note, W3C, 2000.
- [10] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation, World Wide Web Consortium (W3C), 2008.
- [11] P. A. Buhler, C. Starr, W. H. Schroder, and J. Vidal. Preparing for Service-Oriented Computing: A Composite Design Pattern for Stubless Web Service Invocation. In *Proceedings of the 4th International Conference on Web Engineering (ICWE 2004)*, Munich, Germany, volume 3140 of *Lecture Notes in Computer Science*, pages 603–604. Springer Berlin / Heidelberg, July 2004.
- [12] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture — A Pattern Language for Distributed Computing*, volume 4 of *Wiley Series in Software Design Patterns*. John Wiley & Sons Ltd., New York, 2007.
- [13] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture – On Patterns and Pattern Languages*. Wiley Series on Software Design Patterns. John Wiley & Sons Ltd., Chichester, England, April 2007.
- [14] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, editors. *Pattern-Oriented Software Architecture – A System of Patterns*. John Wiley & Sons Ltd., Chichester, England, 2000.
- [15] R. Chinnici, M. Hadley, and R. Mordani. The Java API for XML Web Services (JAX-WS) 2.0. Java Specification Request 224, Sun Microsystems Inc., 2005.
- [16] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, W3C, 2001.

- [17] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 2.0. W3C Recommendation, W3C, 2007.
- [18] J. O. Coplien and D. C. Schmidt, editors. *Pattern Languages of Program Design*. Addison-Wesley, Reading, MA, USA, 1st edition, 1995.
- [19] D. Davis, A. Karmarkar, G. Pilz, S. Winkler, and Ü. Yalçinalp. Web Services Reliable Messaging (WS-ReliableMessaging) 1.2. OASIS Standard Specification, OASIS Web Services Reliable Exchange (WS-RX) TC, 2008.
- [20] T. Dierks and C. Allen. The TLS Protocol Version 1.0. Request for Comments (RFC) 2246, The Internet Society – Network Working Group, 1999.
- [21] J. Ekanayake and D. Gannon. Common Architecture for Functional Extensions on Top of Apache Axis 2. Draft report Y790, Department of Computer Science, School of Informatics, Indiana University, 2006.
- [22] O. Evans. *The Young Mill-Wright and Millers Guide: Illustrated by Twenty-Eight Descriptive Plates and a Description of an Improved Merchant Flour Mill with Engravings by C. and O. Eveys, Engineers*. Carey, Lea and Blanchard, Philadelphia, 1834.
- [23] R. T. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Request for Comments (RFC) 2616, The Internet Society – Network Working Group, June 1999.
- [24] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL Protocol Version 3.0. Internet draft, Netscape Communications, November 1996.
- [25] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. Simple Object Access Protocol (SOAP) 1.2: Adjuncts. W3C Recommendation, W3C, April 2007.
- [26] K. Henney. Methods for States. In P. Hruby and K. E. Sørensen, editors, *Proceedings of the First Nordic Conference of Pattern Languages of Programs (VikingPLoP 2002)*, Copenhagen, Denmark, September 2003.
- [27] M. Henning. A New Approach to Object-Oriented Middleware. *IEEE Internet Computing*, May-June:66–75, 2004.
- [28] M. Henning and M. Spruiell. Distributed Programming with Ice. Manual 3.3.1, ZeroC, Inc., 2009.
- [29] G. Hohpe. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2nd edition, 2004.
- [30] D. Jayasinghe. Dynamic Phase support. Mailing List Document, last accessed August 18, 2008, May 2007.
- [31] A. Kelly. Encapsulated Execution Context. In *Proceedings of EuroPLoP 2003, Workshop D*, 2003.
- [32] M. Kircher, M. Völter, K. Jank, C. Schwanninger, and M. Stal. Broker Revisited. In *Proceedings of EuroPLoP 2004*, Irsee, Germany, 2004.
- [33] P. Leitner, F. Reisenberg, and S. Dustdar. DAIOS – Efficient Dynamic Web Service Invocation. Technical Report TUV-1841-2007-01, Distributed Systems Group, Information Systems Institute, Technical University of Vienna, 2007.
- [34] C. I. V. Lopes. D: A Language Framework For Distributed Programming. Phd thesis, College of Computer Science, Northeastern University, November 1997.
- [35] O. L. Madsen. Towards Integration of State Machines and Object-Oriented Languages. In TOOLS Europe [60], pages 261–274.
- [36] D. Manolescu, M. Völter, and J. Noble, editors. *Pattern Languages of Program Design 5*. Addison-Wesley Professional, 2005.
- [37] K. Marquardt. Patterns for Plug-ins. In Manolescu et al. [36].
- [38] R. Meersman, Z. Tari, M.-S. Hacid, J. Mylopoulos, B. Pernici, Ö. Babaoglu, H.-A. Jacobsen, J. P. Loyall, M. Kifer, and S. Spaccapietra, editors. *Proceedings of the On the Move to Meaningful Internet Systems Conferences (Part I): CoopIS, DOA, and ODBASE, OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005*, volume 3760 of *Lecture Notes in Computer Science*. Springer, 2005.
- [39] N. Mitra and Y. Lafon. Simple Object Access Protocol (SOAP) 1.2. W3C Recommendation, W3C, 2007.
- [40] Mono Project. Mono .NET Remoting. <http://www.mono-project.com/>, last accessed: February 12, 2009.
- [41] Mono Project. Mono Olive. <http://www.mono-project.com/Olive>, last accessed: February 12, 2009.
- [42] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web Services Security Core Specification 1.1 (WS-Security 2004). OASIS Standard Specification, Organization for the Advancement of Structured Information Standards (OASIS), 2006.

- [43] J. Noble. Classifying Relationships between Object-Oriented Design Patterns. In *Proceedings of the Australian Software Engineering Conference 1998 (ASWEC'98)*, pages 98–109, Adelaide, Australia, November 1998.
- [44] S. Y. Nof, W. E. Wilhelm, and H.-J. Warnecke. *Industrial Assembly*. Chapman & Hall, 1st edition, January 1997.
- [45] OMG. Common Object Request Broker Architecture (CORBA). Core Specification 3.0.3, Object Management Group, Inc., 2004.
- [46] S. Perera, C. Herath, J. Ekanayake, E. Chinthaka, A. Ranabahu, D. Jayasinghe, S. Weerawarana, and G. Daniels. Axis2, Middleware for Next Generation Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 833–840, Los Alamitos, CA, USA, 2006. IEEE Computer Society.
- [47] J. Postel. Transmission Control Protocol. Request for Comments (RFC) 793, Information Sciences Institute, University of Southern California; Defense Advanced Research Projects Agency (DARPA), 1981.
- [48] J. Postel. Simple Mail Transfer Protocol. Request for Comments (RFC) 821, Information Sciences Institute, University of Southern California, 1982.
- [49] W. Pree. *Framework Patterns*. SIGS Books & Multimedia, 1996.
- [50] I. Rammer and M. Szpuszta. *Advanced .NET Remoting*. APress Computer Bookshops, 2nd edition, September 2004.
- [51] B. Ramsdell. The TLS Protocol Version 1.0. Request for Comments (RFC) 2633, IETF – Network Working Group, 2004.
- [52] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture*, chapter Interceptor, pages 109–141. John Wiley & Sons Ltd. Wiley, Chichester, England, 2000.
- [53] J. Siddle. An interactive pattern story about remote object invocation. In *Proceedings of the 16th Conference on Pattern Languages of Programs (PLOP'09)*, Chicago, Illinois, USA, 2009.
- [54] S. Soares, P. Borba, and E. Laureano. Distribution and persistence as aspects. *Software – Practice and Experience*, 36:711–759, March 2006.
- [55] M. Stal. Using Architectural Patterns and Blueprints for Service-Oriented Architecture. *IEEE Software*, 23(2):54–61, March-April 2006.
- [56] Sun Microsystems Inc. Java Messaging Service 1.1. Java Community Specification (last accessed: February 13, 2009), Sun Microsystems Inc., 2008.
- [57] Sun Microsystems Inc. Java Remote Method Invocation. White Paper (last accessed: April 14, 2008), Sun Microsystems Inc., 2008.
- [58] The Community OpenORB Project. OpenORB 1.4.0. <http://openorb.sourceforge.net/>, last accessed: November 9, 2008.
- [59] E. Tilevich, W. R. Cook, and Y. Jiao. Explicit Batching for Distributed Objects. Working paper, Department of Computer Sciences, UT Austin, 2009.
- [60] TOOLS Europe, editor. *29th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Europe 1999)*, 7-10 June 1999, Nancy, France. IEEE Computer Society, 1999.
- [61] M. T. Valente and R. Palhares. Collocation optimizations in an aspect-oriented middleware system. *Journal of Systems and Software*, 80(10):1659–1666, 2007.
- [62] O. Vogel. Service Abstraction Layer. In *Proceedings of EuroPLOP 2001*, Irsee, Germany, 2001.
- [63] M. Völter, M. Kircher, and U. Zdun. *Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware*. Software Design Patterns. John Wiley & Sons Ltd., Chichester, England, 2005.
- [64] U. Zdun. Reengineering to the Web: Towards a Reference Architecture. In *Proceedings of Sixth European Conference on Software Maintenance and Reengineering (CSMR'02)*, pages 164–176, Budapest, Hungary, March 2002.
- [65] U. Zdun. Patterns of Argument Passing. In *Proceedings of the 4th Nordic Conference of Pattern Language of Programs (VikingPLOP2005)*, pages 1 – 25, Otaniemi, Finland, 2005.
- [66] U. Zdun. Engineering Loosely Coupled Software Architectures — A Pattern-Based Approach. Habilitation thesis, Vienna University of Economics and Business Administration, Vienna, Austria, January 2006.
- [67] U. Zdun. Pattern-Based Design of a Service-Oriented Middleware for Remote Object Federations. *ACM Transactions on Internet Technology*, 8(3):1–38, 2008.
- [68] U. Zdun, C. Hentrich, and W. M. P. V. der Aalst. A Survey of Patterns for Service-Oriented Architectures. *International Journal of Internet Protocol Technology*, 1(2):132–143, 2006.
- [69] U. Zdun, M. Völter, and M. Kircher. Pattern-Based Design of an Asynchronous Invocation Framework for Web Services. *International Journal of Web Service Research*, 1(3):42–62, 2004.
- [70] C. Zhang and H.-A. Jacobsen. Resolving Feature Convolution in Middleware Systems. *SIGPLAN Notices*, 39(10):188–205, 2004.



## A Pattern Thumbnails

Pattern	Problem	Solution
SERVICE ABSTRACTION LAYER (see [62, 64, 68])	Your middleware system must allow for providing and consuming remote objects through multiple channels, i.e., remot-ing technologies and transport protocols. This channel support should be independent from the core invocation handling on remote objects. New channel should be addable on demand.	The SERVICE ABSTRACTION LAYER adds an extra layer which receives and mediates requests originating from different channels. Each channel contains a channel adapter which translates back and forth requests between the backend and frontend channel formats. This permits you to separate between a core BROKER and frontend channel extensions on top of it.
REQUESTOR (see [63, 66, 13, 32])	How can the assembly of invocation data and disassembly of invocation results (e.g., compiling the data needed, marshaling to and demarshaling from message representations) be organised in a centralised manner? Can we avoid re-implementing this functionality repeatedly when targeting an arbitrary number of REMOTE OBJECTS?	REQUESTORS broker invocation data and results between a calling and called REMOTE OBJECT. They construct canonical, objectified representations of invocation data, i.e., MESSAGES. They orchestrate the subsequent processing steps and, thereby, shield its ancestor LAYERS from remot-ing details.
INVOKER (see [63, 66, 32])	There are recurring and possibly redundant processing tasks to achieve when a MESSAGE arrives at an transport endpoint. This includes demarshaling and disassembling the invocation data, extracting contextual dispatch data etc. These tasks are general for an arbitrary number of targeted REMOTE OBJECTS. Can we avoid redundancy and resulting complexity in MESSAGE processing, invocation setup, and invocation dispatch?	The INVOKER accepts incoming MESSAGES and proceeds by disassembling them. Based on the extracted object references, transmitted by the remote REQUESTOR, the INVOKER may assemble the actual invocation accordingly and dispatch it to the correct REMOTE OBJECT. The invocation result is obtained and returned to the remote REQUESTOR in reverse processing order.
MARSHALLER (see [63, 66, 32]); relates to the MESSAGE pattern and its variants (see e.g. [12, 29])	Invocation and contextual data must be transported in the shape of MESSAGES, i.e., as structured character and, ultimately, as structured byte streams. Remoting technologies differ in their choice for constructing MESSAGES. How can the transformation back and forth between in-memory representations and MESSAGES be organised?	A MARSHALLER is in charge of transforming invocation data, invocation results, and contextual data in either representation. MARSHALLERS reside at both ends of a remote invocation and they must be customisable to support different marshaling strategies and optimisations.
CLIENT and SERVER REQUEST HANDLER (see [63, 66])	Managing the transport of MESSAGES involves redundant and recurring tasks to be achieved by the REQUESTOR/ INVOKER or REMOTE OBJECTS. Transport-level tasks include connection and resource management, as well as the accommodation of application-level and transport-level modes of synchronisation. In addition, transport errors need to be propagated or constraints enforced (e.g., timeouts). How can we dissociate this responsibility from the REQUESTOR/ INVOKER or the REMOTE OBJECT?	The CLIENT and SERVER REQUEST HANDLER are shared by all REQUESTORS and INVOKERS of a BROKER incarnation, respectively. They enclose all transport-level details, the processing of REMOTING ERRORS, and the transport-specific resource management (e.g., concurrency requirements etc.). They are either directly instructed by the REQUESTOR and INVOKER or yield the control upon reception of transport-level events.

*continued on next page*

*continued from previous page*

<b>Pattern</b>	<b>Problem</b>	<b>Solution</b>
REQUEST-REPLY [29]	Two applications communicate through an exchange of MESSAGES. Each MESSAGE realises a one-way conversation. What if the sending application requires a reply from the receiver of the initial MESSAGE?	To realise a two-way conversation, exchange pairs of request and reply MESSAGES. Depending on the intended coupling between the sender and receiver, send the reply MESSAGE either via the request's back channel or, alternatively, via its own communication channel.
FIRE AND FORGET [63]	A client application wants to notify a remote object of an event. Neither a result is expected, nor does the delivery have to be guaranteed. A one-way exchange of a single MESSAGE is sufficient.	A FIRE AND FORGET operations is performed by the REQUESTOR without acknowledging the processing or delivery status to the client. The thread of control is yielded to the client immediately.
MESSAGE [29, 12]	How can we provide (semi-) structured exchange formats for streamed invocation data between remoting ends?	Organise invocation (and contextual) data in terms of MESSAGES which annotate the streamed invocation data with certain meta-data, e.g., identifying the streamed data kind, its origin and destination, size. For processing purpose, provide for a uniform reification in object structures.
REMOTING ERROR [63]	Invocation handling by a BROKER contains many sources of failure. Particular types of error conditions are found within the BROKER, such as network and machine failures, unavailability or misconfiguration of remote objects.	Capture and propagate such error conditions as REMOTING ERRORS between the remoting middleware involved. Allow for discriminating between REMOTING ERRORS emanating from different sources such as MESSAGE handling and transmission.
INTERFACE DESCRIPTION [63]	Developers of client and server applications, and their toolkit of proxy and stub code generators, must access the interfaces of remote objects.	Prepare and offer INTERFACE DESCRIPTIONS which describe the interface of remote objects, e.g. in terms of an interface description language. INTERFACE DESCRIPTIONS negotiate operation signatures and REMOTING ERROR types between the remote ends.
CONFIGURATION GROUP [63]	Remote objects often share configuration properties regarding e.g. marshaling techniques and transport protocols. What is an appropriate scope for defining and activating such properties?	Provide CONFIGURATION GROUPS which organise INVOCATION INTERCEPTORS and MARSHALLERS shared by a coherent group of remote objects. Activating such a CONFIGURATION GROUP means enacting the contained marshaling and extension operations.
INVOCATION INTERCEPTOR [63]	Developers of client and server applications often must provide support and add-on functionality on top of remote invocations (e.g., securing remote invocations). These add-ons should be realised transparently without affecting remote invocations directly. Also, these add-ons can be shared between clients and remote objects.	Provide hooks placed along the invocation path. Have INVOCATION INTERCEPTORS operate on the invocation data directly or have them exchange information via an INVOCATION CONTEXT to realise the add-on services. INVOCATION INTERCEPTORS are managed by middleware users to create extensions.
INVOCATION CONTEXT [63]	How to organise and manage contextual or auxiliary data related to underlying remote invocations without modifying the latter and breaking orthogonality?	Contextual invocation data is embodied by a dedicated object structure, the INVOCATION CONTEXT. This object structure provides a uniform interface for manipulating and extracting this context data. In addition, it serves as a CONTEXT OBJECT [65] for arguments passing between e.g. INVOCATION INTERCEPTORS [63].

Table 1: Thumbnail sketches of relevant remoting patterns (see e.g. [63, 32, 66])

## B Invocation Data Items

The following four kinds of invocation data items can be found in BROKER-based middleware frameworks:

- A MESSAGE [29, 12] provides means to exchange core, contextual, and auxiliary invocation data between remoting ends, i.e., across machine and process boundaries. MESSAGES allow us to stream in-memory information (e.g., requests and replies) into structured byte and character sequences. By structured, we mean that MESSAGES contain annotating meta-data which describes the streamed data and facilitates its interpretation and restoration into in-memory representations. This meta-data identifies different types of streamed data (e.g., the operation name, parameters, etc.), its size, its origin, and its destination. Examples include a wide range of binary (e.g., CORBA's IOP binary encoding [45] or ICE's binary encoding [28]) or structured markup MESSAGE formats (e.g., XML [10] encodings such as the different SOAP/XML variants [9, 39]).
- An INTERFACE DESCRIPTION [63] represents the interface of the remote object accessed by the client component. These INTERFACE DESCRIPTIONS are important for client developers. On the one hand, they permit developers to inspect interface capabilities and generate requests manually. On the other hand, INTERFACE DESCRIPTIONS assist in generating client-side code representations of remote interfaces, either ahead of time (i.e., by a code generator) or just in time (i.e., by means of reflection). INTERFACE DESCRIPTIONS are often realised through an interface description language. Commonly known examples are the Web Service Description Language (WSDL 1.1/ 2.0; [16, 17]) and the CORBA Interface Description Language (IDL; [45]).
- INVOCATION CONTEXTS [63] are used in situations that demand the exchange of contextual invocation data which describes certain conditions, constraints, and processing rules imposed upon the underlying remote invocation (for instance, negotiating message exchange patterns as described by WSDL 2.0 [17]). At the same time, it is tedious and invasive to attach this information to the core invocation data explicitly. This would effectively clutter the signature interfaces of remote operations. Therefore, dedicated object structures are used to pass contextual invocation data through and between the client and server endpoints. For transport, INVOCATION CONTEXTS are streamed into dedicated parts of invocation MESSAGES. Consider SOAP headers [9, 39] or CORBA service contexts [45] as prominent examples. Note that INVOCATION CONTEXTS are also important for realising an extension infrastructure for add-on services in your middleware framework. This will be discussed in the next section.
- REMOTING ERRORS [63] are particular exceptional values reported by the REQUESTOR, INVOKER, and the REQUEST HANDLERS upon sensing errors in processing invocations and MESSAGES, as well as in connection and transport management. REMOTING ERRORS are also communicated between client and server endpoints and, thus, appear in the form of particularly structured MESSAGES. They are also commonly mapped to concrete object types in order to be injected into the local exception handling. For example, SOAP *faults* [9, 39] are used in the context of Web Services to signal REMOTING ERRORS in invocations.

## C Adaptability Requirements

### C.1 Orthogonal Add-on Services

By orthogonal add-on services, we mean extensions which wrap around remote invocations without affecting the structure and format of the core invocation data and its processing. In addition, they may apply to several variants of remote invocations at the same time. Examples include security-related

add-ons (e.g., authentication and authorisation mechanisms), transaction control, logging facilities, persistent storage, and reliable messaging solutions. The example of the `Security provider` and selective encryption (see also Figure 4) falls into the category of orthogonal add-on services. Further examples and elaborations on add-on services are provided in [52] and, in particular, [63].

## C.2 Invocation Styles

When creating a `BROKER`-based middleware, you will find yourself confronted with the requirement to support multiple invocation styles, e.g., kinds of two-way and one-way invocations. This is often demanded by families of remoting technologies. For instance, Web Services (WS) describe different message exchange patterns (MEPs; [39]) between client and server components, such as out-in, in-only, and so on. Invocation styles represent an inherently crosscutting concern, i.e., they affect several `LAYERS` at once. For instance, different invocation styles require different `REQUESTOR` and `INVOKER` interfaces and, therefore, introduce a coupling between the `BROKER` core and the integrating client and server components. Transport-level requirements often differ substantially and cause special treatment by the `CLIENT` and `SERVER REQUEST HANDLERS`.

Invocation styles fall into a set of common invocation patterns [69, 63]. For the scope of this paper, we emphasise the following two invocation patterns:

- The `REQUEST-REPLY` pattern [29] describes a two-way conversation between a client and a server component. It involves two `MESSAGES` being exchanged. This pattern is found in component interactions adopting remote procedure calls (RPCs), for instance, but is not limited to these.
- The `FIRE AND FORGET` invocation pattern [69, 63] captures kinds of one-way conversation between a client and server component. This pattern comforts scenarios in which no invocation result is expected and the reliable delivery *is not* guaranteed on behalf of the client application. There is only a single `MESSAGE` exchange between the two remote ends. `FIRE AND FORGET` operations are commonly found in event-based and publish-subscribe component interactions.

Invocation patterns are characterised by different kinds of invocation artefacts, e.g., `MESSAGES` representing invocations or notifications [29, 12], and their patterns of exchange (e.g., one-way, two-way). The multitude of design requirements and design options is further aggravated because invocation patterns, being located at the application level, map to *communication abstractions* (e.g., send/receive) at the transport protocol level. Invocation patterns are further qualified by different coupling dependencies in terms of location, time, and process synchronisation [2]. We find, for instance, both `REQUEST-REPLY` invocations that realise a process synchronisation (i.e., blocking `REQUEST-REPLY`) and those which don't (i.e., non-blocking `REQUEST-REPLY`). Finally, providing support for different invocation patterns at both the client- and server-side of your framework adds further complexity to your design.

## C.3 Bypassing of Layers

Bypassing describes the capability of selectively omitting processing steps (e.g., the processing of `MESSAGES` by the `MARSHALLER`) in the `LAYERS` structure of your middleware framework. This serves the purpose of realising optimisations and gaining a certain flexibility. Looking at the selective encryption example (see also Figure 4), once the `Security provider` has been successfully added to the processing flow of your middleware framework, you still want to preserve the flexibility of disabling the encryption on selected remote invocations or certain endpoints. More generally speaking, areas of bypassing include:

- Stub- or proxyless invocation handling [11, 33]: Using negotiated `INTERFACE DESCRIPTIONS` to perform remote invocations often means to couple client and server components to rigid signature interfaces. Upon interface changes, client and server components must track these changes (e.g., by regenerating stub or proxy code). This adds maintenance overhead and means a form of coupling to be avoided in certain component interaction styles such as event-based interactions. Therefore, client- and server-side interface proxies are bypassed so that `REQUESTOR` and `INVOKER` are directly addressed to handle and to dispatch invocations.
- Message caching [4], partial (differential) marshaling [1]: These examples try to avoid the repetitive streaming into and the redundant objectifying from `MESSAGES`. This is achieved by factoring out `MESSAGE` parts (e.g., XML markup) which remain unchanged between invocations. In other words, these strategies mean to bypass the `MARSHALLER`, at least partially.
- Batched invocations [59]: Remote invocations are often performed in an atomic manner and map directly to invocations upon single operations of remote objects. These are often small-scale operations so that the tasks performed by the client components are split into a series of successive remote invocations on the same remote object. Given that remote invocations are comparably expensive to process, negative impact on the overall client-perceived performance can be expected. Grouping invocations into batches and processing them in such lots is a viable option, provided that batched invocation results can be assigned to the individual invocation sources. Besides, you have to treat intermediate invocation results specifically. Batching, therefore, groups processing steps into blocks of request and, then, blocks of reply handling, rather than processing pairs of requests and replies as shown in Figure 4. From the perspective of a member invocation in a given batch, different processing steps (e.g., marshaling and/or transport operations) are effectively omitted. They are only performed on the containing batch once.
- Collocation [61]: In case the targeted remote object resides within the same machine or even process boundaries, important steps of processing the remote invocation can be spared. In particular, certain services offered by the `MARSHALLER` and `REQUEST HANDLERS` are superfluous.

#### C.4 Role Distribution

We have explained how invocation and `MESSAGE` handling faces extension requirements which cross-cut the `LAYERS` structure of the `BROKER`. The `LAYERS` structure, while highlighting most general aspects of the `BROKER`, introduces a strong notion of `CLIENT-SERVER` [8] relations, not only between the higher and the lower-level `LAYERS`, but also between a client application and a remote object. We say that the distribution of the client and server roles over applications and components is *asymmetric*. As a result, there are disjoint sets of client- and server-only components. Asymmetry means that (possibly multiple) client-only components are meant to initiate requests for remote invocations from server-only components, or, more precisely, from their remote objects exposed. This asymmetry between client-only and server-only components manifests in the unequal distribution of lookup information, i.e., server applications are only aware of clients for the scope of an incoming invocation. Also, different resource management strategies apply to client-only and server-only applications, the latter putting emphasis on handling multiple and possibly concurrent invocation requests. While the `CLIENT-SERVER` conceptualisation is simple yet powerful for capturing remote invocation details in distributed object systems, it is the exact opposite of what is found in certain middleware framework designs and of what is required in advanced deployment scenarios. The following examples provide evidence for different appearances of *role symmetry*. In these examples, an application takes and its underlying middleware framework supports the role of both server and client within single component interactions.

- Layer symmetry in invocation and `MESSAGE` processing: The functional decomposition of the `BROKER` into `LAYERS` itself reflects “that there is a certain symmetry between client and server

in the layered architecture. This is because the processing patterns at each layer depend on their remote counterpart” [63, p. 128]. This *layer symmetry* [52, p. 136] is yielded by a dependence between remote components within a given LAYER. This dependence is due to most processing steps being performed on invocation data at either the client or server side requiring a counter operation at the respective remote end. Applied to the exemplary BROKER shown in Figure 3, this becomes most visible for the MARSHALLER. Each marshaling operation demands a corresponding demarshaling operation, and vice versa. The MARSHALLER components can, thus, be potentially shared in the client- and server-specific infrastructures. This equally applies to processing steps for add-on services. In our motivating example of the `Security provider` (see Section 4.2), en- and decrypting operations also form pairs across the remote ends. You can translate this symmetry into a design of entities and processing behaviour shared by client- and server-specific invocation handling.

- **Asynchronous REQUEST-REPLY invocations through RESULT CALLBACKS:** You will find situations in which the interacting client and server applications do not need to be coupled through process synchronisation, yet a reply is expected by the client application. That is, the processing infrastructure of the middleware yields the thread of control to the client application after having processed the invocation request. Conversely, at the server side, the server application yields the thread of control to the processing infrastructure after having received the invocation dispatch. Now, one strategy available to communicate a result back to the originating client and, thus, completing the REQUEST-REPLY exchange, are RESULT CALLBACKS [63, 8]. Following this pattern, the server side actively notifies and delivers the invocation result to the client end. At the client side, this is made possible by a RESULT CALLBACK object. This represents a special-purpose remote object exposing a callback interface. Upon reception of a callback invocation, the callback object processes the result in accordance with the client application logic. As the callback object is not only part of the client application, but also acts as a remote object, a particular role symmetry can be stated. Consequently, your middleware framework needs to provide server-specific facilities to client applications under the conditions of asynchronous REQUEST-REPLY invocations.
- **Service-oriented, adaptive system distribution:** The requirements of adaptable configurations in distributed, service-centric systems [7] are another argument which helps explain the importance of symmetric role distributions for middleware designs. By changing configurations, we mean that, on the one hand, service consumers and providers become available or turn unavailable dynamically (at runtime) and that, on the other hand, the composition of service providers must remain adaptable at any time to deliver guaranteed quality attributes. An important example are distributed systems based on workflow engines which integrate with third-party and legacy systems through service adapters and interfaces (see e.g. [67]). In these cases, certain requirements are imposed on your middleware framework. First, if it runs the workflow engine on top, it must support both the client- and server-side of handling remote invocations. In other words, the workflow engine also offers a service interface to receive task results in a process-decoupled manner. In turn, second, the service-providing applications must be in the position to both receive invocation requests (as servers) and deliver corresponding results in an asynchronous manner (as clients). Third, the decentralised addition, removal, and upgrade of service-providing components requires not only facilitating the creation of service adapters and interfaces, but also the propagation of interface changes (e.g., dynamic invocation). This is, for instance, where LAYERS bypassing in your middleware comes into play.

Leela [67] realises such a service-oriented middleware framework. It is built around the idea of peer components which participate both in client and service applications. That is, each peer acts both as a remote object and an invocation client. Thus, the peer is in the position to connect to REQUESTORS for handling their invocation requests as well as to serve invocation dispatches from INVOKERS. Beyond this symmetry in invocation roles, peers are organised in terms of federa-

tions which provide lookup services to all peers under equal terms, so loosening the asymmetry in distributing lookup information (e.g., object references).

There are many ways to make use of these forms of role symmetry in your framework design. One example we experienced is the integration of major components of REQUESTOR and INVOKER incarnations into single framework entities. In Axis2 [5], the `AxisEngine` object-class captures essentials of the INVOKER and the REQUESTOR behaviour in its `send` and `receive` operations. The `AxisEngine` is used by the SERVER REQUEST HANDLERS, i.e., the transport listeners in Axis2, to dispatch incoming invocations and, conversely, by `ServiceClients` as a part of Axis2's client-side proxies to process outgoing invocation requests. Another example is found in the Windows Communication Foundation (WCF; see e.g. [41]). Beyond INVOKER and REQUESTOR (i.e., the *channel factory* and the *channel listener*, respectively), also the REQUEST HANDLER instantiation (i.e., the *channels*) is shared between the server- and client-specific infrastructures (see e.g. [3]). This uniform design is achieved by the WCF channel factory and the channel listener deriving from a shared *channel manager* entity.

Note, however, that under certain conditions it may be advisable to break this symmetry in framework design intentionally. To name an example, certain lifecycling requirements and, thus, resource management strategies may vary between the server- and client-specific elements of the processing infrastructure. To provide scalability at the server side, imagine that you realise a pooling of INVOKERS for scaling out the handling of concurrent invocations. This pooling, however, is not necessarily needed at the client side (see e.g. [3]). Therefore, expect a coexistence of symmetric and asymmetric design elements between the client- and server-side infrastructure of your middleware framework.

# Handling Application Properties

## Simplify Application Customization in Different Environments

**Tim Wellhausen**

kontakt@tim-wellhausen.de  
<http://www.tim-wellhausen.de>

**Martin Wagner, Gerhard Müller**

martin.wagner@tngtech.com  
gerhard.mueller@tngtech.com  
<http://www.tngtech.com>

*January 10, 2010*

Copyright retain by authors. Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.



## Introduction

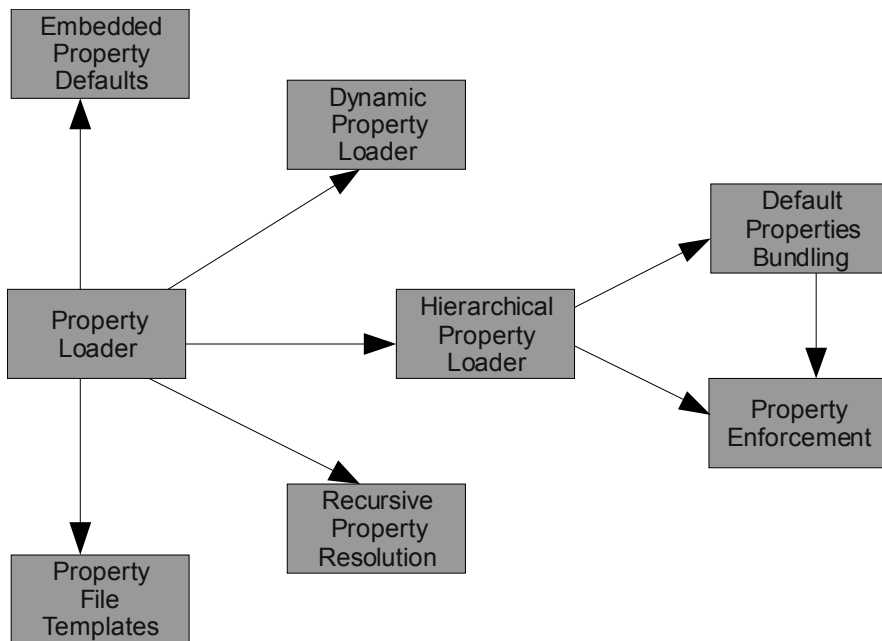
Applications are often deployed in different environments, e.g. on developers' machines, in test or production. In each of these environments, certain configuration settings differ. Examples for such settings are:

- Database logins/passwords
- Access parameters of remote systems
- Email addresses or mobile phone numbers for administrative notifications
- Localization files and configuration
- Interval settings of periodically recurring jobs
- On/off switches of specific sub-modules
- Directories for program resources

This paper aims at evolving a language of patterns dealing with these application properties. It contains patterns dealing with flexible ways of loading properties, providing common properties for multiple deployments and enforcing the explicit setting of some properties. Using the patterns facilitates maintenance of properties, allows for simplified refactoring and gives the possibility to keep varying environment settings under version control. As property storages, property files and database tables are discussed.

The paper starts with the probably well-known PROPERTY LOADER pattern and elaborates on its particular problems and shortcomings. The paper then discusses patterns that improve the handling of application properties in specific situations. At the end of the paper, a reference implementation in pseudo code shows how the patterns interact in practice.

Illustration 1 shows an overview of the patterns and how they depend upon each other. Please note that an arrow from pattern A to pattern B denotes that pattern A provides a context for pattern B.



*Illustration 1: Dependencies between patterns described in this paper.*

# Property Loader

## Context

A software application may be deployed and run in different environments. An application that is used internally in a company, for example, may need to run in different environments for development, testing, integration, and production. An application that is sold as a product may be deployed at many different customer sites.

## Problem

Varying environments usually mandate different behavior and configuration of the application. This may include different passwords or connection parameters, the configuration of caches, or the email addresses of people to notify in case of errors.

**How can you adopt an application to varying environments?**

## Forces

The following *requirements* make the problem difficult:

- *Easy to use as developer.* There should be an easy-to-use programming interface to determine the value of a property with respect to the current environment at runtime. The logic on how to retrieve a property value should not be distributed all over the application.
- *Changing behavior after deployment:* It should be possible to change the behavior of a deployed application without modifying the code and redeploying the application.

## Solution

**Extract all configurable properties from the source code into an external storage and use a PROPERTY LOADER to load and provide access to the property values at runtime.**

Analyze the application to find those parts that need adoption to different environments. Create a property, that is a key/value pair, for every such part and collect the properties in a property storage.

Create a PROPERTY LOADER that reads the properties from the storage at start-up time and makes the values of all properties available to the application at runtime via a dedicated API. The application may query the PROPERTY LOADER any time for the value of a specific property to change its behavior accordingly.

Deploy the property storage together with the application binaries and change the property values in every installation so that they fit the needs of the respective environment.

Property storages can be every form of persistent memory that is able to store pairs of property names and values. The most common property storages are text files and database tables.

A simple text file as property storage typically contains a key/value pair on each line of text as shown below:

```
db.username=user
db.password=secret
```

Accordingly, a database table for properties typically contains at least two columns: one for the property name, which might be used as primary key column, and one for the property value with both columns containing text.

The following Java interface shows an example API for a minimal PROPERTY LOADER that uses a property file as property storage.

```
public interface PropertyLoader
{
    /**
     * Loads all properties from the file with the given name.
     */
    public Map<String,String> loadProperties(String propertyName);

    /**
     * Returns the value of the property with the given key if
     * that property exists, null otherwise.
     */
    public String getValue(String key);
}
```

## Consequences

A PROPERTY LOADER offers the following *advantages*:

- The PROPERTY LOADER provides a centralized access point for all application properties and handles the loading of the properties.
- All relevant properties of an application are located in a single property storage. It is therefore easy to check and manage all properties at a glance.
- By modifying the externalized properties of a deployed system, you can change the behavior of the system without redeploying the system. In particular, it is not necessary to change the source code of the application to adopt it to another environment.

The PROPERTY LOADER also has the following *liabilities and shortcomings*:

- Application developers often cannot foresee exactly which parts of an application need to be adaptable in different environments. If a new environment or a new requirement mandates a change in the behavior of an application for which no properties yet exist, the source code must still be modified.
- By relying exclusively on an external property file, an application may be vulnerable to incompatible manual changes in the property file resulting in runtime errors. By using EMBEDDED PROPERTY DEFAULTS, the application may protect itself against missing or corrupt property values.
- Every application update may introduce new properties that need to be set or adjusted in a local installation. Bundle PROPERTY FILE TEMPLATES with your application binaries for easier and more robust local customization of the application.
- Some applications need different property values not only for different environments like development, integration, and production but also for different hosts

and different users (e.g. an application developer needs different property values while developing the application than a user needs). Maintaining a separate property storage for each possible combination of property values can be very cumbersome. In that case, consider creating a `HIERARCHICAL PROPERTY LOADER` to maintain a storage with default properties and separate storages with differing properties for each environment or user.

- After an application has been successfully deployed, new properties may be introduced at any time when the development continues. When an application upgrade is to be installed, care must be taken to also deploy the new properties. This may prove exceptionally difficult if some properties of the deployed system were manually changed. To simplify this problem, consider using a `HIERARCHICAL PROPERTY LOADER` with `DEFAULT PROPERTIES BUNDLING`.
- Once a property key has been defined and is in use, it is very difficult to change the name of the key because developers don't have access to the property storages of local installations. Using a `HIERARCHICAL PROPERTY LOADER` with `DEFAULT PROPERTIES BUNDLING` also lessens this problem a little, in particular if used in combination with `RECURSIVE PROPERTY RESOLUTION` to provide aliases.
- In order to let changed property values come into effect, the application needs to be restarted, which is not feasible in many cases. `DYNAMIC PROPERTY LOADING` may help.
- Sometimes, sets of property values need to be changed at once, i.e. if one property value of a set is changed, all other properties also need to be changed consistently (e.g. host/username/password for a remote connection). Consider `RECURSIVE PROPERTY RESOLUTION` to manage property sets.
- If a database table is used as property storage, obviously, the database connection parameters itself must be stored in a different place. As such, a hybrid approach combining files and databases as property storage is preferable.

## Known Uses

The Java platform brings its own standard properties API that can be used to easily access application properties stored in files [4]. The Log4J logging framework uses this framework to allow a flexible configuration [11].

Early versions of the Windows platform made extensive use of `.ini` files, holding properties in files. Starting with the Win32 API, this approach has been replaced by the Windows registry [5].

Most applications for the UNIX platform are configured via property files, usually stored in the `/etc` file system tree or in dot files.

# Embedded Property Defaults

## Context

In an application, properties are externalized from the source code by using a `PROPERTY LOADER` that provides an API to retrieve the values at runtime.

## Problem

A property storage may not be consistent with the source code at all times: When the content of a property storage is edited, for example, errors may arise by accidentally changing the key of a property. Or a new property may be introduced to the application but cannot be immediately added to the property storages of all installations. The application code should be made robust against such errors.

**How do you make sure that an application always receives a valid value when requesting a property from the `PROPERTY LOADER`?**

## Forces

The following *constraint* applies:

- *Robustness*: The application code that queries a property value should not cope explicitly with non-existing property values as this puts a burden on individual programmers.

## Solution

**Extend the `PROPERTY LOADER` and provide a method that takes a default value and returns that value when the requested property does not exist in the property storage.**

By calling such a method, an application makes sure that it always gets a valid value from the `PROPERTY LOADER` regardless of the content of the property storage.

The following example extends the Java interface given earlier as an example in the `PROPERTY LOADER` pattern.

```
/**
 * Returns the value of the property with the given key if
 * that property exists, the given default value otherwise.
 */
public String getValue(String key, String default);
```

## Consequences

The given solution has the following *advantages*:

- The application is more robust against inconsistencies in the property storage.
- It is not necessary to set values for all properties in every installation.

The solution also has some *disadvantages*:

- Application developers must take care to always use the API method that takes a default value.

- The default value itself is embedded in the source code, which may not be desirable. If you use a `HIERARCHICAL PROPERTY LOADER` and `DEFAULT PROPERTIES BUNDLING`, you can use externalized default properties.

### **Known Uses**

The Java platform [4] contains a class `Properties` with the method `getProperty(key, defaultValue)` whose semantics are as described by this pattern.

# Property File Templates

## Context

An application uses a `PROPERTY LOADER` and stores properties in property files.

## Problem

When configuration information is externalized in a property storage, the source code and the actual properties available in a specific deployment environment may not be consistent any more. Errors may arise when users or administrators manually change the content of the property files.

**How do you keep externalized properties consistent with the application code?**

## Forces

The following *constraints* need to be resolved:

- *Maintenance*: Property files can be copied and modified freely. It can be difficult to find out and maintain the correct version of a property file for a specific version of an application.
- *Robustness*: An application that reads a property file with properties that are inconsistent with its own version (different property keys or allowed property values) may not work properly.
- *Updating*: A new version of the application may introduce new properties that must be made available in every installation after the upgrade.

## Solution

**Include `PROPERTY FILE TEMPLATES` into the build process so that they are bundled with the application binaries and unbundle these files as part of the installation process so that they can then be modified by the administrator or user.**

A `PROPERTY FILE TEMPLATE` contains all properties that the application relies on. Every property should have a default value that enables the application to work.

When a new version of an application is installed, the new `PROPERTY FILE TEMPLATE` is also installed, possibly overwriting an existing, earlier version of the same file. The administrator of the installation (or a user) then needs to adapt the properties as needed.

## Consequences

`PROPERTY FILE TEMPLATES` offer the following *advantages*:

- When properties are changed (e.g. as part of a system refactoring) and new binaries are deployed, these changes are deployed in all updated environments at once without further manual effort.
- The property files can be stored in the same repository as the source code so that the history of changes to the application is synchronized with the history of changes to the property files.

The solution also causes the following *liabilities*:

- For some properties, e.g. database passwords, there are no defaults available to put into a `PROPERTY FILE TEMPLATE` or it is not feasible to put secret values into a file that is available to all developers. With a `HIERARCHICAL PROPERTY LOADER` and `DEFAULT PROPERTIES BUNDLING` you can keep such information outside the bundled property files.
- When doing an upgrade, existing property files are overwritten. Therefore, manual effort and great care is necessary to restore all property values that had been modified before the upgrade.

## Known Uses

Unix packaging systems such as Debian [6] or RPM [7] bundle application binaries with default settings that can subsequently be modified by system administrators.

Mac OS X applications are distributed as folders having both binaries and property files at well-known locations. These bundle packages [8] are presented to the user as unique items.

Programs written for the Java platform usually distribute binary code in the form of JAR files (ZIP format) that can also bundle property files with application logic [9]. An example of this is the Maven build system which stores build meta information as properties in a JAR file [12].



# Hierarchical Property Loader

## Context

An application externalizes properties from the source code by using a `PROPERTY LOADER` that loads property values at start-up time and provides an API to retrieve the values at runtime.

## Problem

An application may need to be configured differently in different environments, for different users, or on different hosts. Maintaining separate property storages for every possible combination of these factors may be very expensive, in particular because, quite often, only a few property values differ between the environments.

**How can you provide an application with customized, i.e. environment-specific or user-specific, configuration settings without maintaining separate, mostly redundant property storages?**

## Forces

Several *constraints* and *requirements* make the problem difficult:

- *Maintainability*. Redundancies among the property storages should be avoided. If a default value exists for a property, that value should be declared only once. If the value of a property needs to be changed for a specific host or user, only that property should be redefined.
- *Administrability*: It should be easy for a system administrator to recognize which properties still have default values and which property values have already been changed locally.
- *Compatibility*. It should be possible to introduce new configuration properties without the need to manually modify existing local property storages when an installation is updated.
- *Security*: In most enterprise setups, developers should not have access to some properties such as production server passwords. Yet, it should still be possible for developers to provide defaults for non-critical properties.

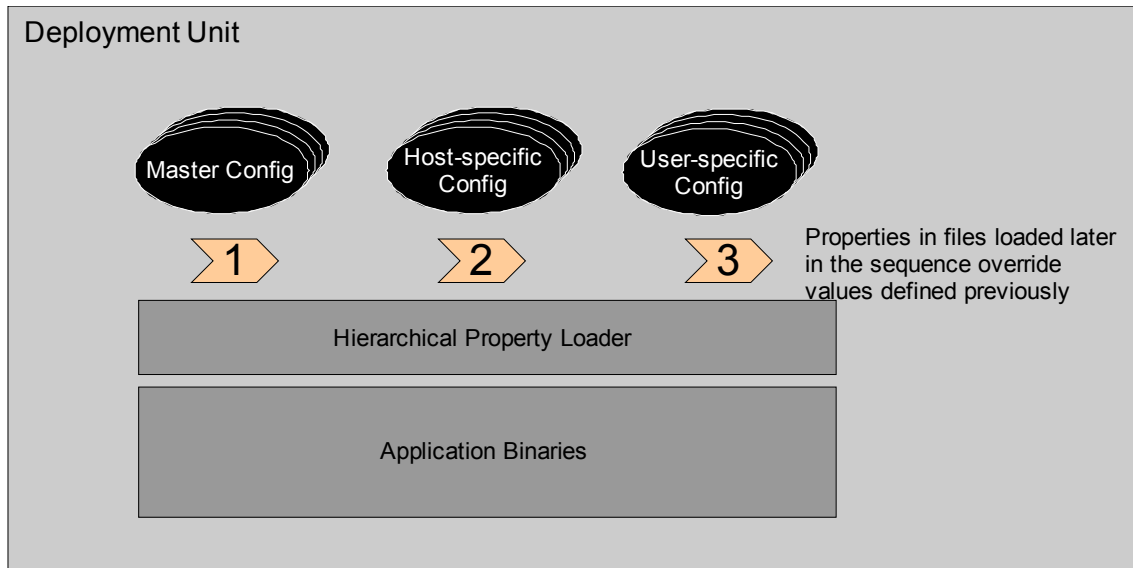
## Solution

**Extend the `PROPERTY LOADER` to handle not only a single property storage but multiple storages and let it process the application properties from these storages in a way so that property values from a more specific storage overwrite the values from a less specific storage.**

Create a default property storage with all properties that the application relies on and assign a default value to each property. For each supported environment, host, or user create an additional property storage that contains only those properties whose values differ from the defaults.

When the application starts up, let the `HIERARCHICAL PROPERTY LOADER` determine the current environment, host, and user and check for the respective property storages. Let the loader read first all default properties and then all properties from the more specific

property storages: every property value of a more specific property storage overwrites the values of the less specific property storages (see illustration 2).



*Illustration 2: Structure of deployment unit with a HIERARCHICAL PROPERTY LOADER*

Security-critical properties are stored in property storages that are only available locally whereas non-critical properties may be stored in default property storages. Security is enforced by means of the local platform. It is possible, for example, to set the permissions of a local properties file so that only the administrators and the application itself may read these properties.

In a complex scenario, a single property may be overwritten by values from several property storages. In order to make the actual value of a property at runtime comprehensible, the HIERARCHICAL PROPERTY LOADER internally stores information about the origin and overwritten values of each property.

If property files are used as property storages, you could implement the pattern by creating separate property files for each environment, host or user and apply a naming convention to make the HIERARCHICAL PROPERTY LOADER aware of the appropriate files. For example, you could call the master property file `config.properties` and environment, host or user specific property files `config.<environmentname>.properties`, `config.<hostname>.properties` or `config.<username>.properties`, respectively.

If a database table is used as property storage, consider adding a database column as qualifier that takes the name of the environment, host or user to which a property entry belongs. By filtering property entries by this qualifier you could retrieve the respective properties by executing several queries and handling the retrieved properties as described above.

An typical application is able to easily determine the name of the host it is deployed on and the name of the user that started the application. The name of the environment the application is currently deployed in (i.e. test, integration, production), is typically not available by standard means. Instead, quite often, a proprietary system variable is declared outside the scope of the application itself and queried from within the application. Such a system variable could be set, for example, in a system start-up script or in the application's start-up script.

## Consequences

The HIERARCHICAL PROPERTY LOADER offers several *advantages* over the normal PROPERTY LOADER:

- Redundancies among property storages are avoided because environment-specific property storages don't duplicate default values.
- You can bundle both default and environment-specific properties with the binaries and still allow a system administrator or user to change some property values in a local, not-bundled property storage.
- The unique properties of an environment are clearly documented by the content of environment-specific property storages because these storages contain only changes to the default settings.
- The HIERARCHICAL PROPERTY LOADER is responsible to determine the current environment and to provide access to the configuration properties. This encapsulation reduces the possibility of errors when retrieving configuration properties at runtime.
- Developers can change properties in property storages of their own that only affect their development environments without the danger of unintentionally checking in any changed default properties to version control systems.

The HIERARCHICAL PROPERTY LOADER also causes *liabilities* of its own:

- The actual complete configuration settings of a system in a specific environment cannot be determined any more from a single source.
- The more hierarchies the loader has to consider, the more complex the property handling becomes, in particular for a system administrator that needs to know how to determine and change properties quickly.
- The application needs a robust mechanism to identify the current environment, which might be difficult if the application is deployed on a cluster or in a virtualized environment. If, for example, the name of host on which the system is deployed is used as qualifier, the system cannot be easily moved to another host any more.
- If several applications are deployed and run on a single machine or cluster, you must take care that the applications' property storages are not mixed up.
- Because every property needs a default value, you cannot enforce that some properties for which there are no valid defaults need to be defined in environment specific property storages. PROPERTY ENFORCEMENT helps in this case.

## Variants

You can change the number of hierarchy levels, which allows for more fine-grained control of environments. For example, users might have different database account names in different environments. In such a setting, the mechanism described above is not sufficient. Extending the hierarchy levels is straightforward – in case of property files, adding a new property file `config.<hostname>.<username>.properties` that overrides all other files does the job.

Besides a distinction in environment, host, and user, other distinctions are possible as well, for example for language and country specific settings.

## **Known Uses**

In standard UNIX systems, hierarchical properties are a common pattern. For many programs, first a system-wide configuration is read, then properties are replaced by user-specific values or even from arguments given on the command line. A well-known application using this pattern is OpenSSH. It allows for global configuration using `/etc/ssh_config`, user-specific configuration with file `~/.ssh/config` and for command-line options via the parameter `-o <option>` [2].

Microsoft Group Policy applies a similar approach to setting policies for Windows machines. Group policies define defaults, user policies get into more specific details [3].

# Default Properties Bundling

## Context

A HIERARCHICAL PROPERTY LOADER uses property files as property storage.

## Problem

Once configuration information is externalized and distributed over multiple configuration files, the source code and the actual properties available in a deployment environment may not be consistent any more.

**How do you keep externalized properties that are hierarchically evaluated consistent with the application code?**

## Forces

The following *constraints* need to be resolved:

- *Robustness*: An application may depend on the availability of specific property values.
- *Updates*: When an application update is installed, new property values must be made available in the local installation.
- *Administration*: If an application is installed only in a few different environments (e.g. one test, one integration, and one production system), the respective property settings for each environment may be known at development-time and should be maintained at a single location.

## Solution

**Include the property files for all supported environments into the build process so that they are bundled with the application binaries. Do not unbundle them as part of the installation process so that they stay unchanged after the installation.**

First, include the property file that contains global default properties so that for all properties that the application relies on property values are available. Also include property files with properties for specific environments whose values are well-known at development time.

By using the HIERARCHICAL PROPERTY LOADER, all property values can still be overwritten by values from local property files.

## Consequences

DEFAULT PROPERTIES BUNDLING offers the following *advantages*:

- When properties are changed (e.g. as part of a system refactoring) and a new version of the application is deployed, the changed properties affect all updated environments at once without further manual effort.
- Default values for specific environments can be changed and new properties can be introduced without affecting an installation's manually set properties.

- Default property files can be stored in the same repository as the source code so that the history of changes to the application is synchronized with the history of changes to the property files.

The solution also causes the following *liabilities*:

- When application developers change the values of default properties and these properties are not overwritten locally, the behavior of an updated application might change without being noticed by the administrators.
- Although default values can be set and deployed for every property, it is not possible to enforce setting properties locally that can not or must not be deployed (e.g. passwords). Consider using PROPERTY ENFORCEMENT then.

## **Known Uses**

Many Unix applications deliver default properties in their packaged application binaries and allow the creation of a user-specific property file where all properties can be changed.

The Windows Registry [5] allows applications to write properties into the system part during its installation and to let users modify properties in the user part of the registry.

# Property Enforcement

## Context

A `HIERARCHICAL PROPERTY LOADER` reads from property storages for specific environments, hosts, or users and provides an API to retrieve the values at runtime.

## Problem

When a property may be set and refined in multiple property storages, it may happen that a property value is not set at all for a specific environment.

**How can you ensure that a mandatory property for which no default value exists is set in an environment, host, or user specific property storage?**

## Forces

The following *constraints* apply:

- *Fail early*: If a property is mandatory but no default value exists, the application should fail early, i.e. before the property is requested for the first time.
- *Notify user*: If the application fails due to a missing property, the application user should be told how to fix the problem.

## Solution

**Add all mandatory properties for which no default values exist to the default property storage and assign a dedicated value to them so that the `HIERARCHICAL PROPERTY LOADER` can check whether this value has been changed in a more specific storage.**

In the default property storage, set the property value of each mandatory property that does not have a default value to a value such as `"TO_BE_DEFINED"`. After the `HIERARCHICAL PROPERTY LOADER` has loaded the properties from all property storages, let the loader check whether any property still has the `"TO_BE_DEFINED"` value. In that case, let it clearly state the misconfiguration and, if no other option is suitable, let it terminate the application.

## Consequences

The given solution has the following *advantages*:

- By marking all properties in the default property storage as described, mandatory properties without defaults are clearly documented.
- If a software upgrade introduces a new mandatory property, there is no risk that the system administrators might miss setting the appropriate value.

The solution also has a *disadvantages*:

- Care has to be taken to spell the default value correctly – a slight misspelling can cause the system to not notice missing properties.

## Known Uses

The Confluence Wiki requires administrators to set the `confluence.home` property [14].

# Recursive Property Resolution

## Context

A (HIERARCHICAL) PROPERTY LOADER reads from property storages and provides an API to retrieve the values at runtime.

## Problem

Sometimes, a set of properties needs to be changed at once because the properties are closely semantically associated and it would be wrong to change only individual properties. For example, there are several databases to connect to and a couple of properties such as the host name, user name, and password need to be changed to connect to each one.

**How do you create a connection between individual properties so that if their values need to be changed all of them are changed at once?**

## Forces

The problem shows the following *forces*:

- *Consistency*: If some properties must always be changed together, it is important to make these changes consistently.
- *Maintainability*: It must be clear which properties are connected.

## Solution

Extend the PROPERTY LOADER by adding the ability to resolve indirections between properties so that by changing the value of one property, the values of a set of properties that refer to the changed property are changed as well.

Extend the PROPERTY LOADER so that properties can not only be mapped to concrete values but also to other properties which are resolved subsequently. Consider the following example:

```
environment=dev
dev.dataSource.username=devuser
dev.dataSource.password=devpw
ci.dataSource.username=ciuser
ci.dataSource.password=cipw
dataSource.username=${${environment}.dataSource.username}
dataSource.password=${${environment}.dataSource.password}
```

In this example the property `dataSource.username` is first resolved to `${dev.dataSource.username}` and then to `devuser` and, accordingly, `dataSource.password` is resolved to `devpw`. If the value of the property `environment` would be changed to `ci`, the value of `dataSource.username` would therefore be resolved to `ciuser` and the value of `dataSource.password` to `cipw`.

## Consequences

The solution has the following *advantages*:



- By changing a single property value, you can switch between sets of property values.
- If a HIERARCHICAL PROPERTY LOADER is in use, all foreseeable possible values of property sets may be added to the default property storage. Only the value of the “switching” property needs to be declared in a more specific property storage.

The solution also has some *disadvantages*:

- Recursive property resolving adds another layer of indirection that makes it more difficult to understand which properties are actually in use by a system, in particular if the affected properties are spread across multiple property storages.

## Known Uses

The Apache Commons Configuration framework allows for recursive property resolution [15] very similar to the pattern described above. The Rails programming language supports different property settings for three testing environment settings that effectively work as a recursive property resolution – based on a single switch property (RAILS\_ENV), other properties are adjusted accordingly [16]. The Spring Framework [17] contains a class `PropertyPlaceholderConfigurer` that, since version 2.5 of the framework, is able to recursively resolve properties.

# Dynamic Property Loader

## Context

A (HIERARCHICAL) PROPERTY LOADER reads from property storages and provides an API to retrieve the values at runtime.

## Problem

The PROPERTY LOADER loads properties at start-up time and stores them internally. Properties are not checked every time they are accessed for performance reasons: Some storages may impose relatively long waiting times when being queried. Still, many applications have very limited time slots for maintenance and restarting.

In other cases, it is desirable to change settings back and forth while the application runs, for example to temporarily change the logging level at a production system.

**How can the values of some properties be changed without restarting the application?**

## Forces

The following *forces* apply:

- *Uptime*: Restarting an application to reload the properties from their property storages might not be feasible for some applications to comply with quality of service agreements. In particular, server applications might take a considerably long time to start-up.
- *Performance*: Switching property values at runtime needs to be fast to avoid performance problems, in particular in large sites under load.

## Solution

**Extend the (HIERARCHICAL) PROPERTY LOADER so that it reloads the properties from all property storages on request.**

Special care must be taken to prevent the access to properties while they are reloaded and to remove references to properties values that might not be valid any more after the reload.

Implement any of the following three strategies to trigger the reload: (1) The PROPERTY LOADER may check for changed properties on every access to property values, (2) may periodically scan for changes to local property storages and/or (3) may provide an API to be explicitly notified when property storages change and a reload must occur.

From a performance point of view it is usually not feasible to execute full hierarchical property loading each time a property is accessed. Some cache invalidation strategy is necessary. With implicit invalidation, a periodic watchdog scans all property storage candidates for changes and reloads properties if necessary. With explicit invalidation, the re-evaluation of the hierarchical property loading is triggered via some signal (e.g. a JMX command [10]).

The choice between implicit and explicit cache invalidation is a tradeoff between quick round-trip times and prevention of accidental reconfiguration. For example, hierarchical

property loading for logging configuration benefits a lot if any changes are loaded automatically – the potential damage of a misconfiguration is usually low.

## Consequences

Dynamic property loading has the following *advantages*:

- Properties can be changed without restarting an application.
- It is also possible to trigger the reload of property values in a cluster of application servers nearly at the same time if the servers share the same base property storage, e.g. property files from a shared directory via NFS.

The solution also has some *disadvantages*:

- Changing properties may cause inconsistent system behavior if some operations assume constant property values over a period of time.
- Administrators must be aware that they must not change property storages while the application is running if the change should not have an immediate effect.
- Implementing a strategy to detect that properties need to be reloaded may harm the application's performance.

## Known Uses

The Apache Web Server can be told via a control command (`apachectl graceful`) to re-scan its configuration files without effectively restarting [18].

The Jetty servlet engine constantly monitors its context directory and reconfigures itself by restarting only specific web applications if any changes occur [19].

Most database systems maintain parts of their own configuration as internal database tables; as such, the configuration properties can be changed at runtime using standard SQL commands. An example is MySQL [20].

The Log4J Java Logging framework implements a periodic check of changing configuration properties via its `PropertyConfigurer#configureAndWatch()` method [11].

## Pattern Language Example

The patterns described in this paper build upon each other and are usually combined. In this section, we give an example combining the PROPERTY LOADER, HIERARCHICAL PROPERTY LOADER, RECURSIVE PROPERTY RESOLUTION and PROPERTY ENFORCEMENT patterns.

The Java platform offers a standard API to load properties [4]. As such, it serves as the basis of our example. Yet, we have also implemented the pseudo code described below in the Ant build system. Adoptions to other platforms are straightforward.

We assume that the default configuration is declared in a file called `config.properties`. It is stored with bundled with the application binary, as well as the host-specific configuration file `config.earth.properties` and the user-specific file `config.joe.properties`. On the deployment environment, a file `config.dev.properties` overrides specific parameters.

```
// Property Loader pattern
Properties props = read("config.properties")           // default properties

// Hierarchical Property Loader pattern
String env = System.getenv()                          // assume "dev"
String host = System.gethostname()                   // assume "earth"
String developer = System.getCurrentUser()           // assume "joe"

File[] overrides =
    ["config." + env + ".properties",                // config.dev.props
     "config." + host + ".properties",              // config.earth.props
     "config." + developer + ".properties"]          // config.joe.props

foreach file in overrides do
    Properties newProps = read(file)
    foreach newProp in newProps do
        props.setValue(newProp.name, newProp.value)
    done
done

// Recursive Property Resolution pattern
while (props.containsPropValueContainingOtherProp)
    String oldValue = propWithValueContainingOtherProp.value
    String newValue =
        oldValue.replace("${" + otherProp.name + "}", otherprop.value)
    propWithValueContainingOtherProp.setValue(newValue)
done

// Property Enforcement pattern
String UNDEFINED = "<HAS_TO_BE_DEFINED>"
if props.containsValue(UNDEFINED)
    throw Exception ("Some required properties are not set")
```

## Acknowledgements

The authors are very thankful to Andreas Rüping who provided invaluable feedback during the shepherding process. We'd also like to thank the participants of the writers' workshop at EuroPLoP 2009 for their valuable comments and suggestions.

## References

All WWW links are valid as of January 10<sup>th</sup>, 2010.

- [1] Apache Ant Project - <http://ant.apache.org/>
- [2] OpenSSH manual - <http://www.openssh.com/manual.html>
- [3] Jakob H. Heidelberg, Managing Windows Vista Group Policy (Part 2) - <http://www.windowsecurity.com/articles/Managing-Windows-Vista-Group-Policy-Part2.html>
- [4] The Java Tutorials: Properties – <http://java.sun.com/docs/books/tutorial/essential/environment/properties.html>
- [5] MSDN – Windows Registry – [http://msdn.microsoft.com/en-us/library/ms724871\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms724871(VS.85).aspx)
- [6] The Debian GNU/Linux FAQ; Chapter 7. Basics of the Debian package management system – [http://www.debian.org/doc/FAQ/ch-pkg\\_basics.en.html](http://www.debian.org/doc/FAQ/ch-pkg_basics.en.html)
- [7] Maximum RPM: Taking the Red Hat Package Manager to the Limit; Chapter 13. Inside the Spec File – <http://www.rpm.org/max-rpm/s1-rpm-inside-files-list-directives.html>
- [8] Apple Developer Connection: Bundle Programming Guide – <http://developer.apple.com/mac/library/documentation/CoreFoundation/Conceptual/CF-Bundles/index.html>
- [9] The Java Tutorials: Packaging Programs in JAR files – <http://java.sun.com/docs/books/tutorial/deployment/jar/index.html>
- [10] The Java Tutorials: Java Management Extensions (JMX) – <http://java.sun.com/docs/books/tutorial/jmx/index.html>
- [11] Log4J Manual - <http://logging.apache.org/log4j/1.2/manual.html>
- [12] MaestroDev, Better Builds with Maven, Chapter 5.4.1 - <http://www.maestrodev.com/better-build-maven>
- [13] Configuring Logging in Atlassian Confluence - <http://confluence.atlassian.com/display/DOC/Configuring+Logging>
- [14] Installing Confluence EAR-WAR on Tomcat - <http://confluence.atlassian.com/display/DOC/Installing+Confluence+EAR-WAR+on+Tomcat>
- [15] Apache Commons Configuration, PropertiesConfiguration class - <http://commons.apache.org/configuration/apidocs/org/apache/commons/configuration/PropertiesConfiguration.html>
- [16] Rails testing environments - <http://guides.rubyonrails.org/testing.html>

- [17] Spring Framework - <http://www.springsource.org/about>
- [18] Apache Web Server apachectl documentation - <http://httpd.apache.org/docs/2.2/programs/apachectl.html>
- [19] Jetty Servlet Engine Context Deployer - <http://docs.codehaus.org/display/JETTY/ContextDeployer>
- [20] MySQL Server Administration Documentation - <http://dev.mysql.com/doc/refman/5.5/en/server-administration.html>

# Software Architecture Patterns for Distributed Machine Control Systems

Veli-Pekka Eloranta, Johannes Koskinen, Marko Leppänen, and Ville Reijonen \*  
{firstname.lastname}@tut.fi

Department of Software Systems  
Tampere University of Technology  
Finland

**Abstract.** In distributed machine control system the software architecture is typically a weak spot because developers lack good design practices. Software architecture design patterns have been found useful for improving the software design. However, there is no comprehensive collection of patterns for distributed machine control systems even though many patterns and pattern languages can be applied to this domain. We carried out architecture assessments in Finnish machine industry and this gave us a possibility to collect recurring solutions as patterns for this domain. The resulting pattern language constitutes a comprehensive collection of solutions for distributed machine control systems. In this paper, we suggest a pattern language for embedded distributed control systems and introduce seven representative patterns from this language.

## 1 Introduction

A distributed machine control system consists of multiple embedded controllers which communicate with each other. These embedded controllers are devised to control, monitor or assist the operation of equipment, machinery or plant [1]. In this context, a distributed machine control system is a software system that controls large machines such as harvesters and mining trucks. Environments of such systems impose special requirements for the software architecture such as distribution, real time, and fault tolerance. For example, harvester head software architecture has to meet certain requirements because productivity demands fast operation and high measurement precision. As machine control systems have become larger and complex, the software architecture of these systems plays a crucial role in the overall quality of the products. Yet, there is little systematic support for designing such architectures. We present a pattern language specifically targeted for this domain. This pattern language assists software architects in designing high-quality systems.

Although there is a plethora of design patterns for software architectures, there are fewer patterns for high level distributed embedded control system design. The identification of patterns in this domain is hard due to several reasons. Typically, the software in many embedded systems has been poorly documented, and the hardware architecture design tends to dominate development of the software architecture. Designers of such systems have often different area of expertise than software system design and are more familiar with the hardware technologies than with modern software engineering. However, there are proven solutions, which are

---

\* Copyright retain by authors. Permission granted to Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website.

informally communicated among the architects. The purpose of this paper is to document this folklore as patterns.

During years 2008 and 2009, we have visited four sites of Finnish machine industry to identify design patterns specific to this domain. The target companies are global manufacturers of large machines and highly specialized vehicles intended for different branches of industry. During the visits, patterns were identified in the context of an architectural assessment of machine control systems provided by the companies. The process for architectural assessment was derived from ATAM [2]. Because the fundamental goal was to create a comprehensive pattern language for this domain, essential solutions were captured and documented as patterns regardless of their prior existence in other pattern collections. The collection process of pattern language is documented in more detail in [3].

In this paper, we introduce a pattern language consisting of found patterns for embedded distributed control systems. In addition, we describe in more detail seven patterns typical for the domain that we consider to be the most interesting. The full set is available online [4].

The patterns were formulated and written down by the members of the assessment team. As the pattern mining process was associated with an architectural assessment, it was easy to find the quality attributes the patterns are related to. This is visible in pattern descriptions in the *Forces* section. Each force is prefixed with the related quality attribute. Quality attributes make it easy to see which aspects of the system design the pattern affects.

A figure is sketched to describe the idea of the pattern. These figures are intended to give a quick intuitive idea rather than a technically sound solution model. An example application of a pattern, abstracted from the actual occurrences, is presented in the pattern description as *Known Usage* section. In addition, *Known Usage* may incorporate a technical diagram.

## 2 The Pattern Language

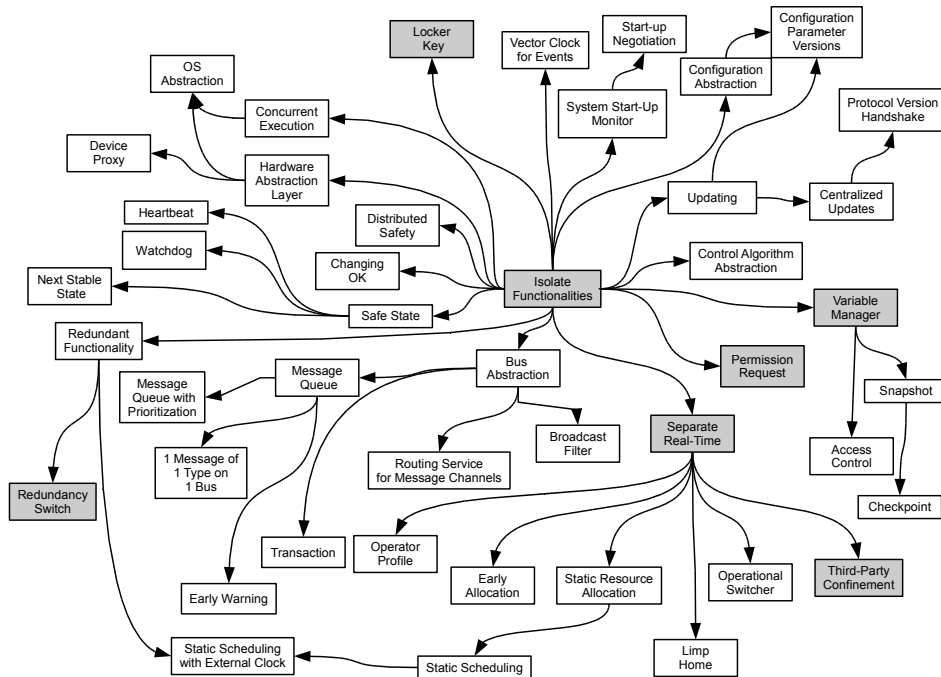
Our pattern language consists of 45 patterns in its present form (Fig. 1). The patterns with gray background color are presented in this paper. When constructing the language, the patterns were at first grouped loosely together based on their area of effect. Patterns which were affecting the design of larger parts of the system were grouped together and patterns with smaller impact were put together depending on which part of the system they improve. This gave an overall view of the pattern relationships.

In the second phase we connected the isolated groups with arrows. The semantics of an arrow pointing from pattern A to pattern B in our language is "pattern B refines pattern A". This means that if the architect has solved some design problems with pattern A, the design context is now compatible with the required context of pattern B. The designer might look at all refining patterns if there are still some unsolved problems in the context.

The connecting of the patterns simulated the thinking process of an imaginary architect, who is tackling the problems in the design one by one and advancing from more general problems towards more detailed design. The key question was to try to understand which is the problem the architect tries to solve and which patterns are related to this problem. As the patterns change the system design, the design problems will change. New patterns can be applied to solve, if necessary, the new design problems.

The simulated use of the pattern language helped to see some "blind spots", solutions that emerged in the software architecture assessments but were not yet identified as patterns. Often there were separate groups of patterns that were difficult to link with the rest of the pattern





**Fig. 1.** Pattern language for machine control systems. Grayed patterns are presented in this paper.

language. After adding a pattern solving a problem in a more general context, the orphaned pattern group could become children of this pattern. For example, there was a group of system updating related patterns that was not linked with the language. This was solved by adding UPDATING pattern that describes the basic mechanisms needed for updating. Therefore, the pattern connecting is very efficient approach to recognize patterns.

### 3 Patterns

In this section, we present the selected patterns that are most interesting and typical for the domain. The selected patterns are grayed in Fig. 1.

We use the following template to describe our patterns. First, we present *Context* where the pattern can be applied. The next section is the description of *Problem* that the pattern will solve in the given context. In *Forces* section we describe the motivation or goals that one may want to solve by applying this pattern. Furthermore, we give *Solution*, argument it in *Consequences* section and give *Resulting Context* that will be the new context after applying the pattern. Finally, *Related Patterns* are presented and in *Known Usage* one known case of usage is given.

### 3.1 Isolate Functionalities

#### Context

An embedded control system is needed to control a large machine which consists of different kinds of sensors, actuators and controlling devices.

#### Problem

What is a reasonable way to design embedded control system for a large machine?

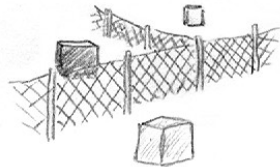
#### Forces

- *Resource utilization*: Available resources should be used efficiently.
- *Analyzability*: It is easier to understand smaller and less complex entities.
- *Extendability*: It should be easy to add new functionalities.
- *Variability*: It should be possible to create different products from the same base system.
- *Testability*: Smaller entities are easier to test.
- *Reusability*: Same components can be reused in different products.
- *Subsetability*: The implementation of different kinds of functionalities may need different kinds of special expertises.
- *Cost efficiency*: Extensive wiring is expensive.
- *Cost efficiency*: High end components cost significantly more than low end components.
- *Fault tolerance*: Extensive wiring is more likely to break.
- *Fault tolerance*: Monolithic software in single device creates a single point of failure.

#### Solution

The system is divided into subsystems according to functionality. These subsystems can be placed on separate devices. For example, there can be separate controller for drive engine, frame, hydraulic pressure, boom, etc. As a controller takes care only its functionality, the hardware requirements are not so high. Therefore, a suitable hardware can be also chosen from low end components. In this way every functionality has just enough resources without wasting them. The controlling device should be located close to the actuators and sensors it is using. This results in less extensive, cheaper and less error prone wiring.

The devices are interconnected and they function as nodes on the bus, abstracting the physical location of the devices. The bus allows easy extendability as new nodes can be added without additional wiring. One of the most commonly used bus standard in large machines is CAN [5]. The communication on the bus takes place using a common message format. For example, CANopen [6] and J1939 [7] are commonly used standards for messaging on top of CAN bus. The throughput of the bus limits the amount of nodes and messages that can be on the bus at the same time. Standard bus components are cost-effective and well available. A bus is usually implemented with single well shielded wire that is less probable to break than a set of wires.



It is possible to test the device as a standalone subsystem, as it implements only certain functionality. This also makes the system easier to understand for developers. Implementation of some functionalities such as boom kinematics needs special expertise. The experts do not need to worry about the whole system as they can concentrate only on the problem area. When the same functionality is needed in some other product, the same device and software can be reused.

### **Consequences**

- + Functional division makes system more understandable and manageable.
- + New nodes can be added easily within the limits of the message bus capacity.
- + Nodes do not depend statically on each other, but only on the message format.
- The capacity of the message bus limits the amount of nodes.
- Throughput may be compromised due to heavy message traffic.
- It may be difficult to know where functionality resides because the location is abstracted by the bus.
- Changing the bus implementation may require changes in several devices.
- A single bus wire can still break, even if the possibility is smaller than with a set of wires.

### **Resulting Context**

Distributed and scalable embedded machine control system where nodes communicate with each other via a bus using a common message format. This pattern forms the base for a distributed embedded control system for a large machine.

### **Related Patterns**

MESSAGE CHANNEL [8] and MESSAGE BUS [9] describe similar mechanisms for decoupling nodes but in different domains.

### **Known Usage**

Programmable Logic Controllers (PLC) are used as controlling units in a drilling machine. These controlling units are connected to sensors and actuators required for the functionality such as measuring drill hole depth. Every unit is a node on the bus that interconnects them. In the drilling machine CAN bus is used as it is common bus solution for this domain. High-level communication protocol is provided by CANopen, removing the need to design it in-house. In addition, COTS (commercial off-the-shelf) components supporting CANopen are readily available. The system is divided so that each functionality has its own controller. For example, there are separate controllers for boom and drill.

## 3.2 Permission Request

### Context

A distributed embedded control system has been divided into nodes with ISOLATE FUNCTIONALITIES. The nodes make independent decisions about their functionality. However, there may be situations where the autonomous functionality can not be executed as other nodes may have conflicting needs preventing the action. For example, the cabin controller has knowledge that parking brake is engaged, in this case the transmission controller should not allow driving. As there are multiple dependencies, albeit simple as such, but as a whole they form complex combinations which are difficult to synchronize.

### Problem

How to ensure that an independent action of a node is not in conflict in some other nodes goals?

### Forces

- *Distributability*: Information required for making a decision for execution may reside on some other node.
- *Resource utilization*: A node may require information from multiple nodes to make a decision. This may cause a lot of bus traffic.
- *Performance*: All information can not be shared to all nodes, as it causes latency and compromises bus throughput.
- *Safety*: Individual node may not take action autonomously as it may compromise safety. For example, the machine should not move if the cabin door is open.
- *Scalability*: When the number of nodes increases, the dependencies between nodes grow exponentially.
- *Decoupling*: A node does not need to mind of functionalities out of their responsibilities. For example, the transmission controller does not need know that harvester head even exists. However, harvester head operation may prevent the driving.

### Solution

A node is responsible for some functionality and it has relevant information for its own responsibility area. However, different functionalities in different nodes may interfere with each other. Additional information, which is not locally available, may be needed for decision whether to execute functionality. Typically actions which need permission are irreversible. There has to be a way to either get information for decision making or externalize the decision.

As the system may consist of different set of nodes, it is difficult to know beforehand what functionalities might interfere with each other. Therefore, it is simpler to centralize the decision making to separate permit authority. It needs information from nodes to make a decision. The permit authority may collect the information on demand or it has collected them from the announcements of different nodes. Usually the permit authority is located in high-end node, as decision making may require processing power.

A device does not need to ask permission for internal actions separately, because it has permission to execute the whole functionality. For example, if harvester head option for prevention for fungal disease is activated, after cutting the tree down the root will be sprayed

automatically. However, for the cutting functionality a permission has to be asked. If something occurs that interferes with the cutting functionality during operation, the action has to be aborted with separate abort message.



### **Consequences**

- + As the decision making is externalized, less processing capacity is required in the node.
- + The permission is always asked from the same entity. This makes the system more understandable.
- + The details do not need to be transferred, because permit authority processes the information to a single decision.
- + Decreases possibility of single node to cause harm.
- + External dependencies of single node are reduced.
- Permit authority may be a single point of failure.
- Designing the permit authority may be challenging as it depends on all the information and therefore from all the nodes.
- Different combinations of devices present different constraints for decision making.

### **Resulting Context**

A system where a node can check from the permit authority if it can execute its functionality.

### **Related patterns**

DEVICE PROXY and HARDWARE ABSTRACTION LAYER can be used to abstract devices, so that if the hardware implementation changes, the permit authority is not affected. SOMEONE IN CHARGE [10] presents a paradigm that there has to be always someone responsible for decision. PROCESS IMAGE or VARIABLE MANAGER can be used to provide a place to store the information required for decision making.

### **Known Usage**

One node of the system is a dedicated authority node. This node uses VARIABLE MANAGER to store system state. For example, the drive controller receives a command from the bus to move the machine. Before the controller executes the command, it checks from the authority node if the command can be executed. For example, if the machine operator has pressed emergency stop, the command can not be executed.

### 3.3 Third-Party Confinement

#### Context

A distributed embedded control system where SEPARATE REAL TIME has been applied. Real time part is separated from high-end functionality. There are third-party applications, such as fleet management and navigation which require a lot of processing power. The third-party applications should not compromise the system functionality.

#### Problem

How to cost-efficiently provide a generic and safe platform to run third-party applications?

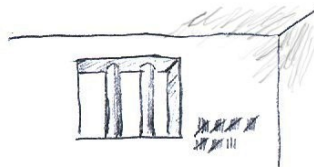
#### Forces

- *Efficiency*: Third-party services may need a lot of processing power.
- *Safety*: There is a need to run third-party applications that can not be trusted as they may interfere with the machine functionalities.
- *Maintainability*: Development and maintenance of third-party services should be possible on COTS hardware.
- *Extendability*: It should be simple task to add new third-party applications to the system.
- *Reusability*: It should be possible to use same third-party applications in different products.
- *Installability*: The machine operator should be able to install preferred third-party applications.
- *Maturity*: There should be a place to run immature applications without risk of interfering with the well-tested machine functionalities. These applications may be third-party or in-house developed.

#### Solution

There is usually a need to offer third-party applications, such as mapping, to improve service quality for the machine operator. These applications rarely have real time demands but may require a lot of CPU time. In many cases such third-party applications are readily available and therefore make rapid development possible. Often their behavior in all situations can not be guaranteed and they should be isolated. Thus, a component providing a platform for the third-party application is added to the system. This component is placed at a high-end node to isolate it from the machine control functionality. Third-party applications reside inside this component in order to isolate them from machine specific high-end functionality.

The third-party applications communicate with the rest of the system via machine vendor provided interfaces. These interfaces should be mature and reliable. The application platform should have enough processing power to support resource intense third-party applications. The other option is to limit the available resources. A common solution to provide this service is to use PC as the high-end node.



### **Consequences**

- + Third-party applications improve user experience for the machine operator.
- + New third-party applications can be easily installed.
- + It is easier to create third-party applications for COTS hardware as COTS operating systems, application development tools, and libraries can be used.
- + Third-party applications do not interfere with other machine functionalities.
- The high-end node creates a single point of failure.
- Confinement may cause latency that may not be acceptable for some third-party applications.
- Interface between machine and third-party application is challenging to design. It may cause maintainability issues and it may unnecessarily restrict third-party functionalities.

### **Resulting Context**

A system with a capability to provide a restricted environment for third-party applications.

### **Related patterns**

QUARANTINE [10] presents an error confinement system.

### **Known Usage**

Machine vendor provides a subcontractor an interface that is used to implement remote diagnostics application. Remote diagnostics application uses the interface to acquire diagnostics data, such as oil pressure and diesel consumption. Subcontractor creates an user interface that is implemented on PC using QT. This user interface can provide different sensor values and production information for the machine operator. Remote access on PC is used to provide work plans in the beginning of the shift and send production reports to organization's database after the operator's shift.

### 3.4 Variable Manager

#### Context

ISOLATE FUNCTIONALITIES has been applied, resulting in a distributed embedded control system with consists of several autonomous nodes. These nodes must share system state information to take proper care of their responsibilities. The node does not need to know the source and location of the information, and all data should be accessible independently of the provider.

#### Problem

How can you efficiently share systemwide information in the distributed embedded system?

#### Forces

- *Accuracy*: Data is volatile, so the local caches need to be flushed frequently.
- *Efficiency*: Data updating causes message traffic that should be minimized.
- *Scalability*: System must be scalable in terms of nodes.
- *Extendability*: It should be possible to add new units and they should have easy access to the system state information without changes to the rest of the nodes.
- *Adaptability*: It should be easy to change the location of the state information source.
- *Usability*: It should be easy to find the desired information about the system state.

#### Solution

A common state information module is added to every node. It contains the information that is relevant to operation of the corresponding node. This information is presented as separate state variables. For example, the current hydraulic pressure in kilopascals may be stored as one integer variable in the component. The values are received from and sent to other nodes using the bus. The local value of the remote variable can be updated when a message containing the changed value is noticed on the bus. All variable changes are sent as broadcast messages so the actual location of the information is not needed. The values of the variables can be sent and updated using different strategies: by-request, periodically, or as a side-effect, for example, when another variable is updated.

A variable can have an associated status or age. This information can be used to check whether the stored value of the variable is valid. If the value is not valid, the node should send a request for newer data to the bus. In this way, the node does not need to know about the origins of the data. The system may present even its internal state information as state variables. Its responsibility is then to send the notifications about the changes in the relevant variables to the bus. As there is a lot of information, which is partially not crucial for the whole system functionality, a care should be taken on what variables are updated via the bus. Otherwise, the bus capacity might be exceeded.





### **Consequences**

- + Assuming that the updating frequency of the shared variables is high enough, each node gets sufficiently accurate state information concerning the other parts of the system.
- + Nodes can change information location transparently.
- This solution does not prevent the nodes from modifying the common system state so that the information becomes inconsistent.
- The solution may result in a large number of variable names that is difficult to manage.
- Variable broadcasting may cause a lot of bus traffic.

### **Resulting Context**

A distributed embedded control system where common state information is shared between the nodes.

### **Related patterns**

The pattern can be thought of as a special kind of PROXY [11] pattern. BLACKBOARD [12] uses similar kind of data sharing. If the state information sharing mechanism is distributed, it can be seen as a BUS ABSTRACTION. On the other hand, the implementation may use BUS ABSTRACTION internally.

### **Known Usage**

A heavy machinery system such as forestry machine uses multiple controllers to steer the machine. The controllers must share information in order to co-operate. This is achieved by every node keeping track of the needed parts of the system state information. The nodes update their system state information by receiving messages from the bus. Some information-carrying messages such as messages containing sensor data are sent periodically, some are sent whenever information is available.

### 3.5 Redundancy Switch

#### Context

A distributed embedded control system where hardware is replicated using the REDUNDANT FUNCTIONALITY pattern to meet availability requirements. Redundant units are relatively error-prone. The system needs to recognize that the active unit's output is within acceptable limits. If the limits are not met, the unit is considered faulty and the system needs to choose a new replicated unit as active. The mechanism described by HEARTBEAT can not be used to detect a failure of a unit as it generates extraneous message traffic between units and may be too slow to detect the failure of an active unit.

#### Problem

How to respond rapidly to a failure in a unit and choose the active one from redundant units?

#### Forces

- *Availability*: A backup unit should take over in predetermined strict response time when the active unit malfunctions.
- *Reliability*: A unit can not make sanity checks for itself as self-tests may not be reliable enough or consume too much resources.
- *Reliability*: In a system with even numbered amount of units, voting is not an option as it may result in draw.
- *Resource utilization*: A unit may not make sanity checks for other units, because it may consume too much resources.
- *Scalability*: It should be possible to add new backup units to the system and use the same fault detection mechanism.
- *Resource utilization*: Fault detection can not be implemented using mechanism such as HEARTBEAT as it is not fast enough and the heartbeat messages from multiple units may congest the bus.
- *Fault tolerance*: Malfunction of monitoring mechanism should not interfere with the output of the active unit.
- *Response time*: Switching the active unit should be as fast as possible.

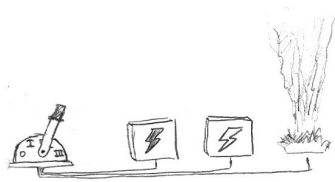
#### Solution

The system has redundant units that communicate with each other to keep their state consistent. One is used as active unit to control the process and others are hot standby units that can take over control when needed. A special monitoring unit or component is added to the system. The monitoring component examines the output of replicated control units. A monitoring component is configured so that it knows the acceptable limits of the output of controlling units. It takes outputs of all controlling units as inputs and examines the output and chooses which unit is used as active unit. The monitoring component then switches the output of active unit as system's control output.

By default monitoring component uses the active unit for output. When it detects that output is not within the acceptable limits, but outputs of redundant units are, it chooses one of them to produce output. The output can be changed using a hardware switch. After the control output is switched it can boot the unit which malfunctioned, in order to get it back

online as a backup unit. If the rebooted unit does not produce output which is in acceptable limits, the monitoring mechanism takes the unit offline and may notify the system operator or administrator.

The monitoring component does not communicate directly with the active unit. In this way, it does not affect the operation of active unit. However, the monitoring component creates a single point of failure. Therefore the monitoring component should be robust and simple enough in order to be reliable. The failure of the monitoring component should not prevent the active unit from functioning.



### Consequences

- + Monitoring component makes the fault detection more reliable.
- + Monitoring component makes the fault detection and unit switching fast enough to meet high availability requirements.
- + System operator or administrator can be notified of unit malfunction.
- Monitoring component creates a single point of failure. In addition, the mechanism limits the possibilities to make a redundant monitoring component.
- If the active units output is in acceptable limits, the monitoring component does not recognize a malfunction.

### Resulting Context

A system where units have redundant units and the controlling which one of them is active unit is placed on separate component. Active unit can be switched within predetermined response time.

### Related patterns

WATCHDOG can be used to detect malfunctioning units and restart them.

### Known Usage

In an embedded control system, a redundancy control unit (RCU) is connected to two redundant control systems. The redundancy control unit is an intelligent switch, which chooses the currently active controller according to the operating statuses of the control systems. Alarm from the sanity checking system signals a malfunction and RCU is notified. RCU tells to backup unit to go active.

### 3.6 Locker Key

#### Context

A distributed embedded control system where ISOLATE FUNCTIONALITIES has been applied. A controller has shared memory that multiple processes can access. Processes communicate with each other using some kind of messaging scheme. Intense communication between processes may occur. Direct interprocess communication scheme involves memory allocation for messages, copying large memory blocks from address space to another and requires active participation of both, the sender and the receiver. Therefore, this kind of scheme should be avoided.

#### Problem

How to efficiently communicate between processes avoiding dynamic memory allocation?

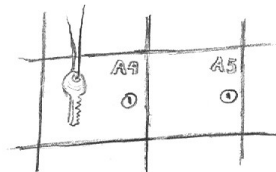
#### Forces

- *Efficiency*: Dynamic memory allocation consumes processing time. It takes time to find and reserve free memory and copy information between memory areas.
- *Throughput*: Direct interprocess communication should be minimized.
- *Safety*: Dynamic memory allocation may fail and such a situation may cause drastic consequences in a real time system.
- *Interoperability*: Interprocess communication should be carried out in a uniform way regardless of programming language.

#### Solution

An embedded controller has memory that can be shared among the controller's processes. All the processes can access this shared memory space and it has enough free space to accommodate all the communication needs of the system. This memory can be used for communication by allocating part of it as message lockers. A locker is a predefined sized memory area that can be used to store message content. The locker size is usually determined from the largest message size. By pre-allocating the lockers, dynamic memory allocation is not required later on. Therefore, the interprocess communication may not run out of memory and is faster.

Every locker is has a key which can be used to access the locker content. The key can be index, i.e. locker number. However, different key types such as Universally Unique ID (UUID) or result from a hash function can be used. Message sender requests a locker (i.e. a space in the shared memory) by using a reservation function. The function returns a key to the caller. The message sender uses a function with the key to insert data in the acquired locker. The key is sent to the receiver with direct interprocess communication scheme. The receiver uses the key to access the message from the shared space. The locker is freed after the access so it can be reused for other messages.



### **Consequences**

- + No dynamic memory allocation is needed for messages. This is fast and prevents memory exhaustion.
- + The transferred data is minimal as only a key is delivered.
- + Solution offers a possibility for the receiver of the key to fetch the message data at suitable moment.
- + Common messaging mechanism makes the system more understandable.
- Normal indexes do not provide any kind of memory protection and therefore the locker integrity is compromised. By using UUID or hash, the key protection level is higher but also more processing is needed.
- In the situation where there is memory management provided by the operating system it may be costly and error-prone to map a single physical memory block to different logical addresses on different processes.

### **Resulting Context**

An embedded controller with fast interprocess messaging capabilities.

### **Related Patterns**

EARLY ALLOCATION and STATIC RESOURCE ALLOCATION can be used to allocate lockers. EARLY WARNING can be used to find out the optimal amount of lockers and also during runtime to warn that critical amount of lockers are already in use.

### **Known Usage**

In an elevator controller a part of the shared memory is reserved for an array that is used as lockers. The array element size is determined from the largest possible message payload. The lockers are used through an interface. The message sender can store the message payload to the locker using an interface function. The interface then returns the key for the particular locker. The sender delivers the key to the message receiver. The receiver uses the key to retrieve the message payload through the interface. When the key is used the locker space is freed. In other words, the same key can be used only once.

## 4 Conclusions

In this paper, we have introduced a pattern language that was collected during architectural evaluations in Finnish machine industry. We have illustrated this language with seven example patterns. These patterns reflect the some characteristics of evaluated embedded systems such as distribution, real time and fault-tolerance.

Although we have a pattern language the pattern collection work is expected to continue. Therefore, pattern language is constantly evolving, leading to a more comprehensive and more systematically organized pattern language of embedded machine control systems. Even though there could be some gaps, the pattern language covers the most important discovered solutions.

## 5 Acknowledgements

We like to thank our shepherd Jorge L. Ortega Arjona for his insight. Also we would like to thank for valuable feedback our EuroPLoP 2009 workshop members Anjali Das, Arto Juhola, Farah Lakhani, Martin Wagner, Stefan Sobernig and Tim Wellhausen. Special thanks to our professor Kai Koskimies and colleagues who gave us valuable input. Additionally, our appreciation goes to Jim Coplien who guided us to the path of patterns.

## References

1. IEE: Embedded systems and the year 2000 problem: Guidance notes. IEE Technical Guidelines 9:1997 (1997)
2. Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley Professional (2002)
3. Leppänen, M., Koskinen, J., Mikkonen, T.: Discovering a Pattern Language for Embedded Machine Control Systems Using Architecture Evaluation Methods. In: 11th Symposium on Programming Languages and Software Tools (SPLST'09). (2009)
4. Eloranta, V.P., Koskinen, J., Leppänen, M., Reijonen, V.: A Pattern Language for Distributed Machine Control Systems. Technical report, Tampere University of Technology (2010) <http://practise.cs.tut.fi/publications.php?project=sulake>.
5. ISO 11898: Road vehicles – Controller Area Network (CAN). ISO, Geneva, Switzerland. (2003) <http://www.iso.org/>.
6. CiA: CANopen specification. CiA, Nürnberg, Germany. <http://www.can-cia.org/>.
7. SAE: J1939 standard. SAE International, Warrendale, USA. <http://www.sae.org/>.
8. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing. Wiley (May 2007)
9. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
10. Hanmer, R.: Patterns for Fault Tolerant Software. John Wiley & Sons (2007)
11. Vlissides, J.M., Coplien, J.O., Kerth, N.L.: Pattern Languages of Program Design 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1996)
12. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. John Wiley & Sons (August 1996)

# Summary of the open session on pattern selection and pattern repositories

*Edited by Aliaksandr Birukou, using comments by Alexander Ernst, Alexandros Karagkasidis, Till Schümmer*

During this open session we summarized existing repositories and solutions for pattern selection and then brainstormed on issues in pattern selection and how one can build a new pattern repository to overcome those issues. This summary document first lists the participants of the session, then summarizes existing solutions and issues, and, finally, lists architecture components of the proposed repository and the benefits such repository will provide to pattern authors and users.

## *Participants*

1. Paris Avgeriou
2. Aliaksandr Birukou
3. Alexander Ernst
4. Ed Fernandez
5. Uwe van Heesch
6. Jim Hensman
7. Alexandros Karagkasidis
8. Stefan Sobernig
9. Uwe Zdun
10. Till Schümmer (post-session discussion)

## *Existing solutions*

### **Pattern repositories and catalogs.**

Some recent examples:

- PatternForge [http://www.patternforge.net/wiki/index.php?title=Main\\_Page](http://www.patternforge.net/wiki/index.php?title=Main_Page)
- CMI patterns <http://www.cmi-patterns.org/>
- J2EE patterns <http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html>
- Portland Pattern Repository <http://c2.com/ppr/>
- EAM Patterns (<http://eampc-wiki.systemcartography.info>)
- PatternSeer (not available anymore, hosted by Microsoft till 2007) - <http://www.patternseer.org/>

### **Formal approaches for pattern selection**

See works by Paris Avgeriou, Uwe van Heesch, Uwe Zdun and others

### **Pattern search engines and similar tools**

- work by Aliaksandr Birukou and Michael Weiss
- Google Design Pattern custom search <http://www.google.com/coop/cse?cx=000531763273211731096%3Ab-lv61obcte>.

## *Raised issues*

- Community is important

- How existing pattern repositories are used?
  - Jim told that people contribute and use patterns to their repository called Pattern Language Network ([PLANET](#))
- In some cases, catalogs are not useful when you design an application
  - however, if you have a network of patterns, it can be much easier to explore electronically than in a book
  - more general issue can be raised - “patterns are not useful when you design an application”. May be catalogs/patterns are not useful because people do not use/believe in/understand/have unsuccessful experience with them?
- There is a need for context-aware guidelines
  - the system should know which part of the project you are working on (e.g., security) and helps you to browse only related patterns (e.g., security patterns)
  - another example – the system knows you have applied A, so A is not recommended anymore, but B is, because it usually follows A
  - also links between patterns are guidelines
- Why would people provide feedback?
  - Reasons for providing feedback include but not limited to these:
    - Because they are proud of their way applying the pattern
    - Because they want to thank the author for the advice
    - Because they want to correct the authors
    - Because they want to get rewarded somehow
    - Organizing a focus group at EuroPLoP was mentioned as one of the options. Participants could tag pattern and specify links between them
  - How many people who are really interested in patterns have enough time to provide feedback? Is it a realistic assumption that a common developer under constant pressure has enough time to contribute, to provide valuable feedback? This also relates to the issue what is the community.
- What is the target community of the repository?
  - Pattern contributors: Search for the related patterns to avoid reinventing the wheel.
  - Pattern freaks or hobbyists, and students: people just interested in patterns who want to know what’s up in the area. The repository could substitute the search on Amazon or Google or running through all the \*PLoPs proceedings as a newcomer.
  - Constant pattern practitioners – the developers who systematically apply the patterns and want to learn more about new patterns.
- Is format a problem?
  - There was a common agreement that you should not provide a uniform format, several formats might and should be supported as long as we know how to search and display patterns using each of the formats.
  - Uploading a single PDF per pattern + author + title would not take much time
  - We should be careful when dealing with patterns that appeared in books
- Possible problems with naming – same name for different pattern, similar (the same?) patterns with different names



- this is a minor issue
- Different levels of abstractions of patterns
- If possible, the repository should trace pattern sequences
- How to convince practitioners to apply patterns?
  - If we knew, the existing pattern repositories would have been more successful?
- If we include links between forces, developers (or more general: users) can explore the problem space and thereby reach an understanding of available patterns that inform their design of a solution
- We need to reach the understanding of how the process of pattern application looks (or could look) like, i.e. concrete Use Cases. Then we can think about how the repository could support this process.
- Think not only about the design phase
- It is also about the “software engineering culture” and about developers – repository should take into account their skills and practices
- The repository is a tool, and the developers must be protected against wrong application of this tool.

### *Architectural components*

- Repository
- Indexing (of pattern description)
- Search (text-based)
- Metadata + semantics

To the repository you contribute patterns in .pdf or .wiki formats. It would be nice to have there all patterns from Wiley, Addison-Wesley, but just indexing all \*PloP patterns would be a good start.

Contributions to the repository should have: name, authors, .pdf for each pattern

Optionally, authors can provide the categorization of their patterns as well. The categorization must include many independent criteria (or axes) like layer of abstraction, development phase, type of application (distributed systems, data bases,...), relation to the non-functional requirements, etc. Probably, the POSA I is a good starting point.

Wiki is there to gather metadata from the community:

- Links to related patterns
  - links are forged by users, e.g. “these two patterns are related”
  - some links can be mined from the information about usage of patterns, if we have it
- Sequences
- Associations between patterns (in a graphical way), a pattern map (like in POSA 1 book or J2EE pattern map)
  - we should be aware that it can become extremely unhandy – too large to make any sense. The art is to provide meaningful subgraphs
  - pattern maps should be optional
- Topics (what pattern is about), etc.
- Resolving ambiguities with names

To build the repository we should use Hillside material

### ***Benefits for researchers and writers***

- Comments about (using) patterns
- Patterns applicable to my problems
- Patterns related to my research (e.g., CHI patterns)
- ...

### ***Building new repository***

- Who is going to use the repository (target audience)?
- We should not do automatic implementation of patterns, instead support the user of patterns in making the “right” design decisions.
- Have concrete use cases about how the process of pattern application looks like.
- The repository should “fit” existing practices/culture of developers.

## Appendix



Figure 1 - Demo workshop



Figure 2 - Workshop in the garden



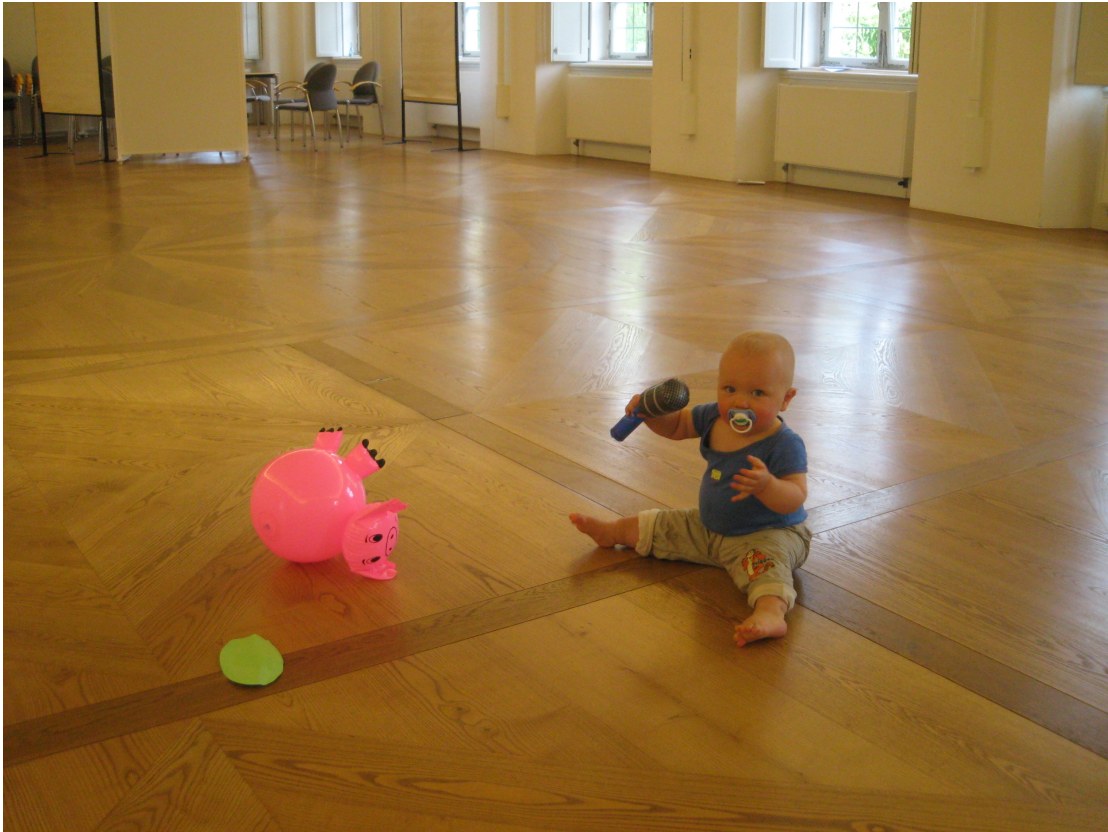


Figure 3 - Grisha



Figure 4 - All work and no play / is not the EuroPloP way





Figure 5 - Iresee from the garden



Figure 6 - Inside the chapel



Figure 7 - Closing swim

All photographs (c) Allan Kelly