Generative Programming of Graphical User Interfaces

Max Schlee
Software Engineer, DFA
Thomson grass valley
Brunnenweg 9, D-64331 Weiterstadt (Germany)
+49 (0) 6150 104 0
Max.Schlee@thomson.net

ABSTRACT

Generative Programming (GP) is a new paradigm that allows automatic creation of entire software family, using the configuration of elementary and reusable components. GP can be projected on different technologies, e.g. C++-templates, JavaBeans, Aspect-Oriented Programming (AOP), or Frame technology. This paper focuses on Frame Technology, which aids the possible implementation and completion of software components. The purpose of this paper is to introduce the GP paradigm in the area of GUI application generation. It also demonstrates how automatically customized executable applications with GUI parts can be generated from an abstract specification.

Keywords

ABA, AI, ANGIE, DSL, FODA, GP, JANUS, OO, OOA/D, Qt , XML, UML

INTRODUCTION

The evaluation of such a technical projection was done using an application as an example. For this purpose, a family of image processing program was chosen, as it provides enough variability, which makes it an excellent base for generative programming. The purpose is to use the GP paradigm in the field of GUI generation.

Unlike the well-known GUI generators such as JANUS, the ANGIE-Based GUI generAtor (ABA) is domain-specific. This means that the GUI part of the system that needs to be generated also undergoes a feature modeling process. This permits the creation of the systems containing only those GUI parts and functions that were specified by the user, both in their generated source code and in the executable binary file. Furthermore, it is possible to use the components created by the user as well as those created by a software designer. Though ABA is a (feature) model-based GUI generator, it does not require an extensive knowledge of the base model because it is completely projected on the GUI elements and the specificator is connected to the generator. This is the reason why the generator can be used intuitively and can be easily understood [6].

INTRODUCTION OF GENERATIVE PROGRAMMING (GP)

"Most OOA/D methods focus on developing single systems rather than families of systems. OO implementation mechanisms for implementing intra-application variability (e.g., dynamic polymorphism) are also used for inter-application variability. This results in "fat" components or frameworks ending up in "fat" application."[3]

Generative Programming (GP) is a software engineering paradigm based on modeling software system families. When given a particular requirement specification, it can use configuration knowledge to automatically manufacture highly customized and optimized intermediate and end products from elementary reusable implementation components [2]. It does not compete with the existing paradigms but supplements them. GP supports reusability and adjustability much better than object-oriented programming, frameworks and design patterns [5].

The purpose of software development automation is not only to speed up the development process and reduce development costs, but also to improve software quality and error resistance. Besides, it helps reduce the cost of maintenance and similar necessities [10].

Generative Domain Model

Generative Programming represents an approach permitting the creation of whole product families. It consists of three elements:

1. The left oval represents the methods used for the family member specification. It is made for users and computer experts. They use a specific language that has specific features and terms. This language is implemented as a *domain-specific language* (DSL). The purpose of a DSL is to give the user the opportunity to describe a particular system in a most suitable way. This helps to "order" a particular system. In order to do this, a text, a form dialog, or a graphical-interactive environment can be used [7].
2. The arrow represents the configuration generator. The configuration generator automates the product assembly. It accepts a DSL specification and analyses it. Then, if necessary, it carries out a buildability check and assembles a software product from the implementation components [6].
3. The right oval describes the world of the software developer. It contains developed elementary components the system can be assembled from. They

must have maximum combinability with minimum redundancy. The use of a feature diagram which graphically represents the elementary components in the form of a tree-like structure is helpful.

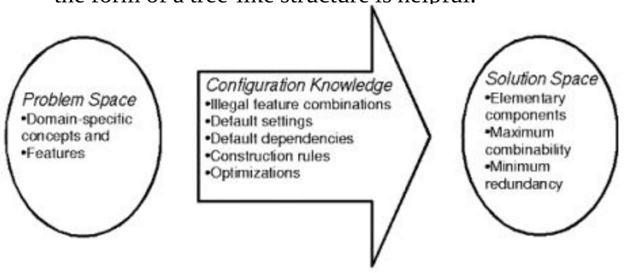


Figure 1. Generative domain model

The GP principles assume that the members of a system family can be generated based on the common model of this system family, the generative domain model [10]. The modeling of software system families allows the production of a large number of system variants based on specific requirements. Most of this process can be automated, which significantly reduces the development time and cost as well as improves the software quality [6].

Feature Models

"OO technology does not support reuse and configurability in an effective way. One of weaknesses of current OO Analysis and Design (OOA/D) is the inadequate support for variability modelling Feature modelling allows capturing the variability of domain concepts. Concrete concept instances can then be synthesized from abstract specification" [3]

"*Current OO notations do not support variability modelling in an implementationindependent way, e.g., the moment you draw a UML class diagram, you have to decide whether to use inheritance, aggregation, class parametrisation, or some other implementation mechanism to represent a given variation point"* [3]

Feature modeling is the central activity of domain engineering. It was introduced by the *Feature-Oriented Domain Analysis* (FODA). Feature modeling is the process of analyzing and modeling of common and variable features of concepts and their interdependencies, as well as describing their arrangement in a coherent model, the so-called feature model. Feature models serve as documentation help.

With feature models, common and variable features within a system family can be modeled. The main component of the feature models are features. Apart from the name, a large amount of additional information can belong to a feature. This additional information comes mostly in the form of tables, lists, or free text. It can also be documented in diagrams or with the help of a suitable tool, for example, the feature model editor *AmiEddi[12]*. Feature diagrams with this additional information make up a feature model.

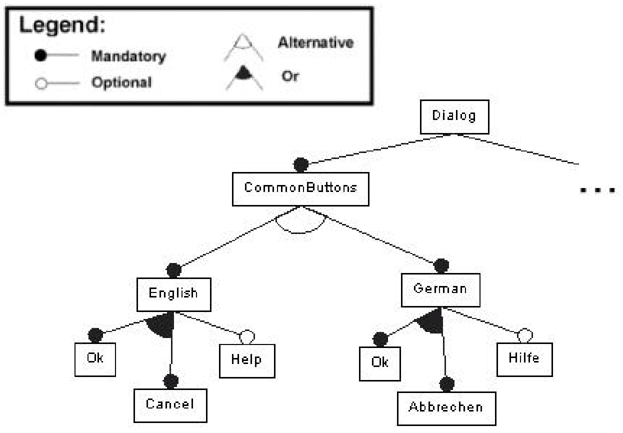


Figure 2. Example for a feature diagram

An example of a feature diagram is shown in a Figure 2. It describes a part of a dialog window. Its root represents the dialog concept. The other nodes of the features are:

- Mandatory features: Every dialog window has the common buttons.
- Alternative features: A dialog window may support either English or German languages.
- Or-features: A dialog window may have an Ok-Button, a Cancel-Button, or both.

Optional feature: A dialog window may or may not have a Help-Button.

Dialog-based graphical-interactive DSL

The use of a dialog-based graphical-interactive DSL permits the user to automate the whole specification development process. It is no longer necessary to see the logic or the declarations in the specifications. The important part is that the user needs no knowledge of the language used in the specification (e.g. XML). GUI elements are easier for the human perception than text specifications. When editing a text specification, it is highly possible to make mistakes, such as typing errors. Besides, the semantic rules can be violated because the user has to create and run the logic of the system that is being created in his head, which is absolutely impossible if he has to create complex systems. The introduction of a dialog-based graphical-interactive DSL makes it easier for the user and takes over this task. It is in charge of both dependencies appearing in the system and invalid input. The generator is becoming more and more user-friendly, the user can easily learn how to use it by trying out the available options. The user can not do anything wrong, as the system accompanies him at every step assuring an error-free creation of a specification that does not need to be verified by the generator. The generator can be used intuitively and is easy to understand. The whole process of the specification creation runs in the background, visible for the user.

When creating a dialog-based graphical-interactive DSL, it is necessary to transform the feature model into GUI elements. This transformation has the advantage that the user does not need an advanced knowledge of the feature models used in the DSL. The transformation of the DSL must meet the following criteria:

The mandatory features do not appear in the dialog because they are available anyway. If such a feature does appear in the dialog, then it appears only as a group boxes title.
The logic of the optional features is completely covered by the checkboxes. An additional logic verification is unnecessary.
The Radio buttons are suitable for the group of alternative features. In this case, an additional logic verification is also unnecessary because the logic coincides with the widgets (RadioButtons).
A group of *or*-features can be represented with the help of check box. In this case, an additional verification is necessary because the user must make sure that at least one feature is selected. Or, if the superior feature is an optional feature, the user is not allowed to deactivate it, e.g. if the user does not select one of its sub-features that are gathered in an *or*-group).

FRAME TECHNOLOGY

Frame technology is a new generator technology suited for use with the generative paradigms like generative programming (GP), MDA and others. Frame technology deals with the concept of frames and slots. In 1974, Marvin Minsky's article "A Framework for Knowledge Presentation" [9] was published in the book "The Psychology of Computer Vision" (ed. Patrick Winston) [7]. The frame/slot approach originated in *Artificial Intelligence* (AI) and was then introduced to the area of picture identification. Later, it turned out that it was also possible to use this approach in the analysis and synthesis of languages [10]. A frame defines set values, the so-called slots. The slots of a frame can be filled with frame instances. In this way, complex hierarchies can be created. The purpose of the frames is to classify the scene descriptions or texts based on their patterns [7]. This concept is very powerful for the representation of text.
Frame technology most not be viewed as a paradigm of its own. However, it can be used while working with the generative paradigm [7]. This technology proved effective and showed decent results in industrial use [1].
Frame technology is well suited for generative programming. Feature models [2] possess all necessary information to build a frame hierarchy

ANGIE-BASED GUI-GENERATOR (ABA)

In ABA, the generative domain model is divided so that the problem area is projected on the GUI of the specificator that is connected to the GUI generator. Most of the configuration knowledge such as the default settings, dependent features, illegal feature combinations as well as optimizations is taken over by the specificator. The construction rules are carried out with the help of ANGIE (developed by the Delta Software Technology GmbH)[4]. script functions. The solution area is projected on ANGIE frames. The frame contents include such files as C++, make files, XPM and Developer Studio workspace.

The generation process in ABA

The whole development process runs in the background and is visible for the user. This is the reason why the generator can be used intuitively and is easily understandable. The base of the process is the ABA user interface. It controls the whole generation process.

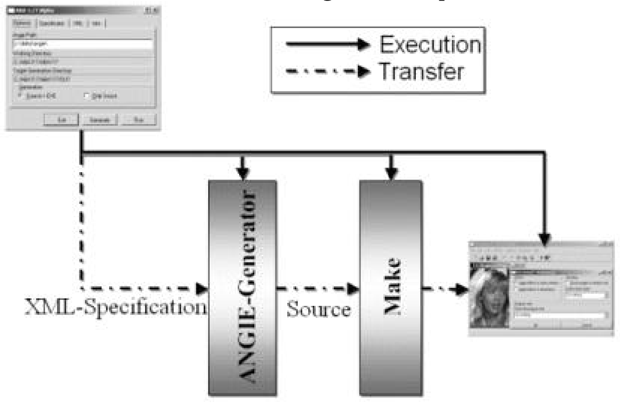
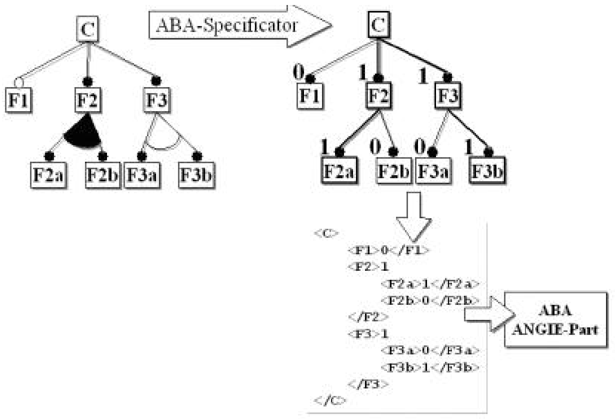


Figure 3. The generation process in ABA

One of the most important tasks of the ABA user interface is the creation of the XML specification that is then transferred to the ANGIE-part of the generator. There, the system creates the source code according to the selected specification. If the user wishes, the "Make" process is also carried out. When it is completed, the ABA user interface can start the created application.

The Process of Creating a XML-Specification

The main purpose of the specificator is to create a specification of a system. In order to accomplish this, it is necessary to remove all the variable parts from the feature diagram.



```
<C>
    <F1>0</F1>
    <F2>1
        <F2a>1</F2a>
        <F2b>0</F2b>
    </F2>
    <F3>1
        <F3a>0</F3a>
        <F3b>1</F3b>
    </F3>
</C>
```

Figure 4. The creation process of the XML-specification

The first part of the figure shows an example of a feature model that undergoes a specialization process with the help

system. Alternately, the features marked with 1 are required by the system.
This tree-like structure is described with XML. Based on the XML presentation, the user can see the whole structure of the system that is being generated and can imagine the components the system consists of. Based on this presentation, the user sees the non-existing components and the corresponding parts of the system where these components will not be situated.

Frame Creation

The user can see the tree-like structure created by the specificator as a frame hierarchy tree. The logic of the component construction rules is divided over the whole frame hierarchy. Every frame is a world of its own. First, the concept frame is created. It contains the knowledge of the frames that only have to be "called to life". With the creation of the corresponding frame the purpose of the specificator is fulfilled. The created frames create further subordinate frames.
Parts of components appear in different parts of the code. This is why every component includes parts from its father node, as they contain the information about the correct positioning of the component parts. First, the concept frame is created. It receives the information about what components are necessary from the XML specificator and creates them. The created components receive further information about the components they can contain from the XML-specification. If required by the specification, these components are created.

CONCLUSION

The feature diagram of the entire system of possible GUI prototypes consist of over 200 features. The generator permits the creation of a great number of prototype variants $V \approx 5 * 10^{17}$. In comparison with that, using the Microsoft AppWizard $V \approx 6 * 10^{7}$ prototype variants can be created[11]. The frame borders are labelled in the comments, in the source of the generated prototypes. The frames can be updated in the generator after the editing of source. This makes Roundtrip-Engineering possible and allows to expand the generator when necessary.
During the MiniABA-Project was successfully tested the option of generating GUI not as C++-Source but as Resources. For further information, see [8].

ACKNOWLEDGMENTS

This research was carried out as part of the PoLITe project. PoLITe stands for "The Development and Testing of the Manual for the Software Product Line Implementation Technologies". This project was supported by the Foundation for Innovation in Rhineland-Palatinate, Germany. The development of the software lines is documented at the Fraunhofer Institute for the Experimental Software Engineering (IESE) and at the University of Applied Sciences in Kaiserslatern (its branch in Zweibrücken). I would like to thank all members of the PoLITe project.

REFERENCES

1. Basset, P., *Framing software reuse: lessons from the real world*, Yourdon Press, Prentice Hall, 1997.
2. Czarnecki, K., Eisenecker, U.W., *Generative Programming*. *Methods, Tools, and Applications*, Addison-Wesley, 2000, see http://www.generative-programming.org .
3. Czarnecki, K., Eisenecker, U.W., *Synthesizing Objects*, An Extended Version of a Paper Presented at ECOOP'99 , 1999.
4. Delta Software Technology GmbH, *Angie Online Help*, provided with ANGIE, version 2.1.2002 see http://angie.d-s-t-g.com .
5. Eisenecker, U.W., Czarnecki, K., *Generative Programmierung: wie man Komponenten baut und nutzt*, In: iX 2/1999, S. 126-132.
6. Eisenecker, U.W., Henss, M., Lang, M., Schlee, M., *Generative Programmierung*, Slides, Net.ObjectDays 2002.
7. Eisenecker, U.W., Schilling, R., *Generative Programmierung mit einem Frameprozessor*, In: iX 10/2002, S. 114-121.
8. Emrich, M., Eisenecker, U.W., Endler, Ch., Schlee, M., *Emerging Product Line Implementation Technologies: C++, Frames, and Generating Graphical User Interfaces*, not published jet see www.polite-project.org.
9. Minsky, M., *A Framework for Representing Knowledge*, Massachusetts Institute of Technology, A.I. Laboratory, Artificial Intelligence Memo No. 306, June 1974, ftp://publications.ai.mit.edu/ai-publications/0-499/AIM-306.ps.
10. Schilling, R., *Generative Programming – Von der Theorie zur Praxis*, Delta Software Technology GmbH, 2001, see www.delta-software-technology.com.
11. Schlee, M., *Generative Programming of Graphical User Interfaces*, Diploma Thesis, University of Applied Sciences of Kaiserslautern, 2002
12. Selbig, M., A Feature Diagram-Editor – Analysis, Design and Implementation of its Core Functionality, Diploma thesis, University of Applied Sciences Kaiserslautern, Zweibrücken 2001.