Proceedings of the

1st International Workshop on Semantic Matchmaking and Resource Retrieval

SMR 2006

September 11, 2006

Co-located with 32nd International Conference on Very Large Data Bases (VLDB 2006)

The Convention and Exhibition Center (COEX)
Seoul, Korea

Edited by
Tommaso Di Noia, *Eugenio Di Sciascio* and *Francesco M. Donini*

Preface

Matchmaking is basically the process of discovering, based on a given request, promising partners for some kind of purpose. It is a process common to several scenarios in the Internet era, spanning from e-commerce to web-services, to grid computing, to human resource management, to actual dating services, to Peer-to-Peer computing. Both requests and partner descriptions are usually, but not always, given some kind of structure. When such a structure refers to a shared ontology, semantic-based matchmaking can be obtained. Obviously, main issues emerge when the search is not limited to identity matches but, as in real life, when the objective is finding partners suitable at least to some extent, or – when a single partner cannot fulfill the request – find a pool of cooperating partners able to accomplish it. As this process may lead to various possible matches, the notion of ranking becomes central, to provide a list of potential partners ordered according to some criteria.

Due to the diversity of frameworks of application, matchmaking has been studied by several communities and it has been faced under various perspectives and techniques, including text-based information retrieval, vague query answering, planning, constraint programming, fuzzy logic, frame logic, and description logics reasoning. Also, different definitions and terminologies have been given to common matchmaking issues. Recently, the Semantic Web initiative has provided a spin towards logic-based approaches, which are still well far from being a "panacea" or a real "silver bullet", as several problems remain unsolved, such as the need for shared or at least integrated schemas, and the need for inferences smarter than standard ones. We received a broad spectrum of submissions and we are confident that the papers that were finally selected for publication and presentation will contribute to a better understanding of Semantic Matchmaking issues and possibilities. All papers were reviewed by at least three reviewers, either members of the Program Committee or external experts in the field.

We are grateful to all authors for their submissions, the members of the Program Committee and the external reviewers for their time and efforts. Finally, we would like to acknowledge that all the management of the workshop was greatly simplified by using the EasyChair conference management system (www.easychair.org) developed by Andrei Voronkov.

Tommaso Di Noia, Eugenio Di Sciascio and Francesco M. Donini
SMR 2006 PC chairs and organizers

Workshop Organization

Programme Chairs

Tommaso Di Noia — Politecnico di Bari, SisInf Lab - Italy
Eugenio Di Sciascio — Politecnico di Bari, SisInf Lab - Italy
Francesco M. Donini — Università della Tuscia, SisInf Lab - Italy

Programme Committee

Boualem Benatallah — The University of New South Wales - Australia
Fabio Casati — HP Labs - USA
Lalana Kagal — Massachusetts Institute of Technology - USA
Mick Kerrigan — DERI - Austria
Werner Kieling — Lehrstuhl fur Informatik II, Universitat Augsburg - Germany
Matthias Klusch — DFKI - Germany
Ruben Lara — Tecnologa, Informacin y Finanzas - Spain
Alain Leger — France Telecom - France
Massimo Mecella — University of Rome ”La Sapienza” - Italy
Massimo Paolucci — DoCoMo Laboratories - Germany
Yun Peng — University of Maryland, Baltimore County - USA
Euripides Petrakis — Technical University of Crete - Greece
Axel Polleres — Universidad Rey Juan Carlos - Spain
Marta Sabou — The Open University - UK
Ulrike Sattler — University of Manchester - UK
Pavel Shvaiko — University of Trento - Italy
Robert Stevens — University of Manchester - UK

External Reviewers

Khalid Belhajjame — University of Manchester - UK
Simona Colucci — Politecnico di Bari - Italy
Alfons Huhn — Lehrstuhl fur Informatik II, Universitat Augsburg - Germany
Uwe Keller — DERI - Austria
Massimiliano de Leoni — University of Rome ”La Sapienza” - Italy
Freddy Lecue — France Telecom - France
Azzurra Ragone — Politecnico di Bari - Italy
Ruggero Russo — University of Rome ”La Sapienza” - Italy
Eufemia Tinelli — Politecnico di Bari - Italy
Evgeny Zolin — University of Manchester - UK

Contents

Keynote Talk

Industry Adoption of Semantic Web.
Guo Tong Xie **1**

Regular Papers

MOD - A Multi-Ontology Discovery System.
Le Duy Ngan, Tran Minh Hang and Angela Eck Soong Goh **3**

On the Topological Landscape of Web Services Matchmaking.
Hyunyoung Kil, Seog-Chan Oh and Dongwon Lee **19**

SPARQL-based OWL-S Service Matchmaking
Said Mirza Pahlevi, Akiyoshi Matono and Isao Koijima **35**

A flexible model forWeb service discovery.
Rubén Lara, Miguel Ángel Corella and Pablo Castells **51**

Recovery of Faulty Web Applications through Service Discovery.
M.G. Fugini, E. Mussi **67**

A Non-Monotonic Approach to Semantic Matchmaking and Request Refinement in E-Marketplaces.
Tommaso Di Noia, Eugenio Di Sciascio and Francesco M. Donini **81**

Short Papers

Semantic Matchmaking using Ranked Instance Retrieval.
Matthias Beck and Burkhard Freitag **97**

Semantic-based Taxonomic Categorization of Web Services.
Miguel Ángel Corella and Pablo Castells **105**

Helping Architects In Retrieving Architecture Documents: A Semantic Based Approach.
Rambabu Duddukuri, Prabhakar T.V **113**

Keynote Talk (Abstract)

Industry Adoption of Semantic Web

Guo Tong Xie

Semantic Integration Department
IBM China Research Lab
xieguot@cn.ibm.com

Data Semantics is becoming a more and more important topic for Information Management. Generating semantic metadata and publishing to its existing and potential consumers are viewed as a key step to information integration and information delivery. Further, if the semantic metadata and the data it described are made accessible and identifiable on the Web — that is the vision of Semantic Web — the way people and program access data will be deeply changed.

This presentation will talk about the trend of industry adoption of semantic web, and several usage scenarios of using semantic web technologies in metadata management, master data management and information integration areas. Finally, some challenges of applying semantic web technologies to real industry solutions will be discussed.

MOD - A Multi-Ontology Discovery System

Le Duy Ngan, Tran Minh Hang, and Angela Eck Soong Goh

School of Computer Engineering, Nanyang Technological University (NTU), Singapore
{ledu0001, A0261836A, ASESGOH}@ntu.edu.sg
http://www.ntu.edu.sg

Abstract. The rapid development of semantic web services has led to a demand for a semantic web service discovery mechanism. Though such discovery systems have been developed, there is a lack of support for matching semantic web services which use different ontologies. This paper introduces a general framework for the discovery of web services that use both the same ontology as well as different ontologies and describe experiments to test the algorithm. Contributions include a discovery algorithm and a novel concept similarity matching algorithm to eliminate mismatches and to increase the accuracy by using syntactic, properties, domain, and neighborhood similarity matching. The experimental results confirm the viability of the discovery system.

1 Introduction

Web services discovery is the most important task in the web services model. Discovery is the process of finding web services which satisfy certain requirements. Researchers have developed discovery systems [5-8,13,16-19] to match web service requesters against web service providers. Current discovery systems are adequate when the web service requester and provider use the same ontology. Unfortunately, research has not focused much on the situation where a web service requester and provider use different ontologies. Since the web service requesters and providers operate independently, they usually define their own ontologies to describe their services. In the real world, a web service provider can provide an exact service to the requester even though both services use different ontologies. Therefore, a discovery system that supports web services using different ontologies is extremely important.

Previous work on web services discovery includes LARKS (Language for Advertisement and Request for Knowledge Sharing) [17-19], a project based on a collaboration between Toshiba and Carnegie Mellon University [6,7], a Matchmaker from TU-Berlin [5], and systems by Li and Horrocks [8], Paolucci [13] etc. These web service discovery systems only support matching web services using the same ontology. This assumption implies that if different ontologies are used, matching cannot be carried out. As mentioned, this is a major limitation.

Cardoso and Sheth [3] have addressed this problem. However, their method of matching web services based on concepts, syntax, and their properties is inadequate, and has been improved upon by Oundhakar et al [12]. The discovery technique is based on METEOR-S [9] which is a web service discovery infrastructure. This infrastructure provides a facility to access registries that are based on business domains and grouped into federations. The matching is divided into syntactic and semantic matching. In semantic matching, Oundhakar's algorithm has improved on Cardoso's work by considering the coverage and the context information of concepts, thereby improving on the matches and eliminating false matches. However, the system does not allow users to intervene during the matching process to restrict the matching results. This will be a significant problem since the numbers of web services from matching result is huge. Furthermore, the system does not consider the domain of two ontologies, from which the two concepts belong. The domain of the two ontologies is important in the matching process.

This paper introduces a general algorithm which supports matching web services using not only the same ontology but also different ontologies. Computing semantic similarity between concepts from different ontologies is the core component of the algorithm. The component will be introduced with a novel algorithm to eliminate mismatches and improve matching by using syntactic, properties, domain, and neighborhood concept similarity matching. The system is named Multi-Ontology Discovery (termed MOD) system. In the discussion to follow, we assume that the building and maintaining of the ontologies are outside the scope of the paper.

Our work is based on OWL-S [4] which is a semantic web service description language. OWL-S uses OWL [20] ontology to describe the service. A semantic web service defined in OWL-S includes four basic classes, namely *Service*, *ServiceProfile*, *ServiceModel*, and *ServiceGrounding*. For matching, only *ServiceProfile* is used because it contains the description about the service to be discovered. The *ServiceProfile* describes three basic types of information, namely, contact information, functional description of the service, and additional properties. The contact information contains a textual description providing a brief description of the service. It describes what the service offers, or what service is requested. The functional description of the service is the most important declaration used for matching. It specifies the parameters of the inputs, outputs, preconditions, and effects of the service. The Profile also declares operations of the web service.

The rest of the paper is organized as follows. Section 2 introduces the MOD matching algorithm. Section 3 presents main components which are used in MOD. Section 4 describes experiments and results, followed by the conclusions in section 5.

2 MOD Algorithm

The matching algorithm is divided into four stages namely input, output, operation and user-defined matching. The four-stage set up is similar to TUB [5] but MOD supports matching web services using different ontologies.

2.1 Definition of Semantic Similarity

In order to introduce concept similarity, we define similarity distances. The definition is based on the experience in [12]. Assume that A, B, C, and D are concepts of an ontology. Figure 1 describes the possible relationships between concepts. The similarity distances between two concepts are defined as follows:

- Exact similarity (A, A): is the most accurate match. It happens when the two descriptions are semantically equivalent. The similarity degree for this match is given a value 1 and this is the highest similarity.
- Subsumes similarity (A, B): B is subsumed by A. As A is a direct superclass of B, it is more general than B. This match is less accurate than exact match. The similarity degree for this match is 0.75.
- General similarity (A, D): A is more general than D but A is not a direct superclass of D. We assume that there are x levels (with $x \geq 1$) between A and D, then the similarity degree for this match is $0.75 - 0.01*2^{x-1}$.
- Invert-Subsumes similarity (B, A): B is more specific than A and B is a direct subclass of A. The similarity degree for this match is 0.3.
- Specific similarity (D, A): this is the inverse of the general match. A is more general than D but A is not a direct super class of D. We assume that there are x levels (with $x \geq 1$) between A and D, then the similarity degree for this match is $0.3 - 0.05*x$.
- Fail similarity (A, C): A and C are not related in the ontology. With reference to figure 1, we also have Fail (B, C).

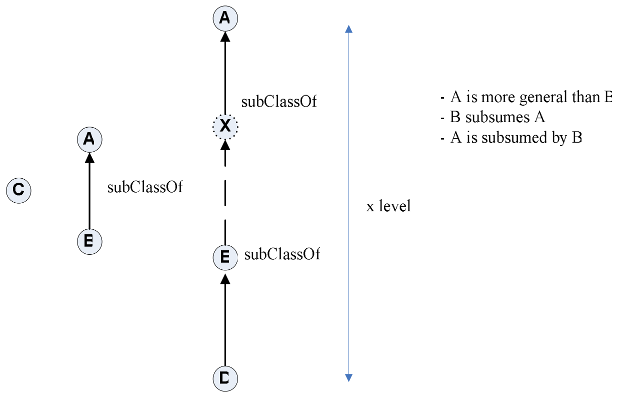


Figure 1. The relation of concepts

Similar to [12], we use levels of two concepts to compute general and specific similarity. We reduce the similarity by 0.01 for parents, and 0.02 for grandparent, and so on for super-concepts. With sub-concepts, we reduce by multiples of 0.05 for each level. The maximum number of levels to be considered is 6.

2.2 MOD Matching Algorithm

The MOD algorithm divides the matching process into four-stage and each stage is independent of the other and the final result is based on the sum of the individual stages.

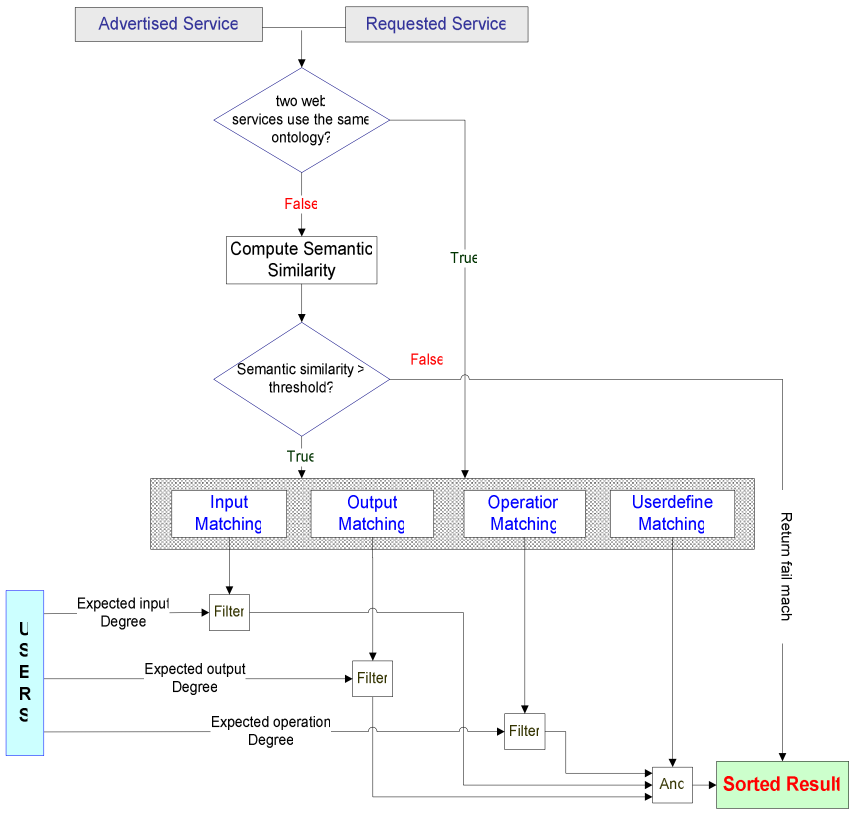


Figure 2. Matching algorithm

The matching algorithm is described in figure 2. *reqService* and *advService* denote the requested and advertised service respectively. First, the two web services are checked if they use the same ontologies. If they do, the four-stage matching algorithm [5] is used. If they do not, the “compute semantic similarity” component computes the concept similarity between concepts. If the semantic similarity is less then a threshold which was defined by the user, then the matching fails. Otherwise, the four-stage matching algorithm is used with input, output, or operation matching. The algorithm is described in more detail in section 3. The four-stage algorithm first checks the user-defined matching. If the matching fails, the result of matching two services also fails and the matching process finishes. Otherwise, the matching process continues with input, output, and operation matching. The final matching result will be composed of the input, output, and operation matching.

2.2.2 Input Matching

In the input matching, we determine how inputs of the advertised services are satisfied by the inputs of the requested services. The input matching operates as follows: for each input of the advertised service, the algorithm tries to find an input of the requested service that is the most similar (the highest degree match). If none of the advertised input can match with the requested input, the input matching returns fail. Thus, the matching of the two web services will fail. The input matching algorithm is elaborated as follows:

```
inputMatch (inputAdv, inputReq){
 if (inputAdv = empty_list){
 return 1; //exact match.
 }
 degreeInput = 0;
 numInput = 0; //number of input of advertised service
 for all inputAdvElement in inputAdv do{
  numInput = numInput +1;
  degreeMax =0;
  for all inputReqElement in inputReq do{
   if(checkOntRelation(inputAdvElement,inputReqElement)){
    degreeMatch =
    computDegree(inputAdvElement,inputReqElement);
   }
   else {
    degreeMatch =
    computSimilarity(inputAdvElement,inputReqElement)
   }
   if (degreeMatch >degreeMax){
    degreeMax= degreeMatch;
   }
  if degreeMax = 0 then return fail;
   degreeInput = degreeInput + degreeMax;
  }
 degreeInput = degreeInput/numInput;
 if degreeInput < threshold then return fail;
 else return degreeInput;
}
```

The number of input of advertised and requested service may be none or many. If the advertised service does not contain any input, then the input matching returns an exact match. *inputAdvElement* and *inputReqElement* denote elements of the input list of advertised and requested service respectively. *degreeMatch* is the degree of the matching between an input pair (an input of advertised service and an input of the requested service). *degreeMax* is the maximum degree of an input of advertised service with every input of the requested service. In other words, *degreeMax* is the maximum of *degreeMatch* for every input of advertised service. *degreeInput* is the final result of input matching. *degreeInput* will be compared against a threshold whose value depends on the specific domain. If it is less than the threshold, then the matching fails. Otherwise, the matching returns the *degreeInput*. *checkOntRelation*() is the function that checks if the two concepts have a hierarchical relationship.

computDegree() is the function that computes the semantic degree of two concepts when they are related hierarchically. *computSimilarity*() is the function that compute the semantic similarity between two concepts from different ontologies. The algorithms for the *computSimilarity*() functions will be introduced in section 3.

2.2.3 Output, Operation, User-defined, and Final Result Matching

Both output and operation processing are carried out in the same manner as input. But in output matching, all output from the requested service will be matched with each output in the advertised service. In operation matching, service category operation of the requested and service category operation of the advertised service are matched. The results of output matching and operation matching are the degree of output matching and operation matching.

In user-defined matching, the users can declare some more constraints or rules to restrict the matching and increase the accuracy of the results of matching. For example, a requester wishing to buy a computer describes a web service with price as input and configuration of the computer as output. They may also declare more constraints relating to the computer manufacturer. These constraints or rules will be matched against the advertisement. Users may decide not to define any constraints or they may define many constraints. We assume that the result of a constraint matching is either "match" or "not match". In short, the result of user-defined matching is as follows: Match if every constraint is satisfied; Fail if at least one constraint fails. Of course, it is possible to define the distance of this matching but it is outside the scope of work.

The final matching result will be composed of the input, output, and operation matching.

$$S = \frac{(w_1 * inputMatch + w_2 * outputMatch + w_3 * operationMatch) * userMatch}{w_1 + w_2 + w_3}$$

w_1, w_2, and w_3 are defined by users. After matching with all advertisements from the database, we will sort by the degree of similarity and return the matched list to the requester.

3 Semantic Similarity between Concepts from Different Ontologies

This section introduces the main component of MOD which was mentioned in section 2, namely, computing semantic similarity of two concepts from different ontologies. Determining the semantic similarity of concepts from ontologies is necessary in information retrieval and information integration fields related to ontologies. MOD focuses on matching inputs, outputs, and operations because in the semantic web, these parameters are in fact ontology concepts. The concept similarity algorithm includes four main components: syntactic similarity, properties similarity, domain similarity (or context similarity), and neighborhood similarity.

$$CS = \frac{w_s * synSim + w_p * proSim + w_c * conSim + w_n * neiSim}{w_s + w_p + w_c + w_n} \qquad (1)$$

where w_s, w_p, w_c and w_n are weights defined by users depending on application. The following sections provide details of the five main matching components.

3.1 Syntactic Similarity

Each concept in an ontology is labeled (concept name) and has a short text (concept description) which describes it. The syntactic similarity computes the similarity between the concept names and concept descriptions of the two concepts, respectively. The concept names in OWL are defined as a word or a set of words. There are some techniques to compute word similarity such as n-gram [2,15,21] and token Matcher [14] but most of these only consider a word as a string of characters. The semantic relationship of the words, which is very important in computing word similarity, has been ignored. To overcome this drawback, WordNet [10] is used.

WordNet can only be used when the words which we would like to compute similarity are stored in its database. The WordNet database only stores root and single words. Therefore, before using WordNet, we must have a preprocessing process to convert words to their roots. If both concept names contain only one word, we can use directly WordNet to compute similarity words. In cases when there is more than one word in the concept name, we will find the most similar terms between synonym sets. WordNet is also used to compute similarity between two concept descriptions in the same way as computing concept names similarity when the latter contains more than one word. The syntactic similarity is a weighted average of concept names and concept descriptions similarity.

3.2 Property Similarity

A concept may have one or more properties. Similar to concept, a property also has a name and description. In addition, it contains range and cardinality. To compute the property similarity, all this information of the two properties should be matched. The method to compute property name and description similarity is syntactic similarity which was introduced in section 3.1.

Range of a property is either a primitive data type or another concept. If both ranges are primitive data types then the similarity between two ranges is as shown in table 1. If one range property has a primitive type and the other has a concept, the ranges are incompatible; therefore the range similarity is 0. If two range properties are concepts, the matching is carried out recursively as with the computing of two concepts.

Table 1. Similarity when range is primitive data type

	Range of Property 1					
Range of Property 2		Integer	Long	Float	Decimal	String
	Integer	1	0.9	0.8	0.7	0.3
	Long	0.9	1	0.8	0.7	0.3
	Float	0.8	0.8	1	0.7	0.3
	Decimal	0.7	0.7	0.7	1	0.3
	String	0.3	0.3	0.3	0.3	1

Cardinality of property permits the specification of the exact number of elements in a relation. For OWL, the cardinality values are limited to 0 or 1. We simply define the Cardinality similarity as follows: Cardinality similarity =1 if the cardinality of two properties is equal; Cardinality similarity =0 if the cardinality of two properties are different

The final property similarity is the combination of the three components: syntactic, range, and cardinality similarity.

3.3 Domain Similarity

Domain which is also known as a context of the ontology is important to the computation of concept similarity. For example, assume there is an Animal ontology which has concept Fish. Another Food ontology also has concept Fish. We assume that the properties of the two concepts Fish are the same. If we do not consider the domain of the two ontologies, the concept similarity of two concepts from the two ontologies will be 1. However, the Fish concept has different meaning in the two ontologies: one refers to an animal; the other refers to food. Therefore, the similarity distance of the Fish concept from the two ontologies should be low. In short, domain must be considered in computing concept similarity.

To compute the domain similarity of the two concepts, we compute the similarity of the two roots from the two ontologies since the root represents the domain of the ontologies. The root is a special concept in an ontology, which does not have super concepts. In the real world, most OWL ontologies have one root, but there are ontologies with more then one root. In this situation, we must find the root which the concept belong to. The paper only solves the simple and common situation when the ontology has one root.

3.4 Neighborhood Similarity

Neighborhood similarity computes the two concepts with respect to their neighborhood, namely, their super concepts and sub concepts. Figure 3 presents a matching of two concepts Network Node and Network Element. Both concepts have super concepts Equipment; their sub concepts are Computer, Switch Equipment, and Computer, Central Hub, respectively.

The neighborhood similarity is computed using the following equation:

$$\text{neighborSim} = \sqrt{\text{supSim} * \text{subSim}}$$

where supSim and subSim are super concept similarity and sub concept similarity, respectively.

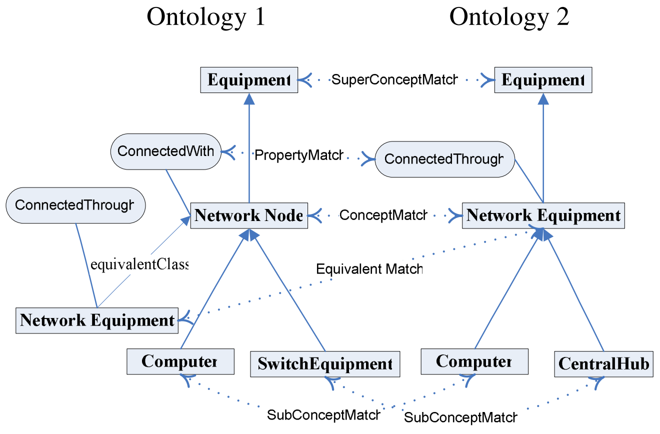


Figure 3. Comparing two NetworkNode concepts

A concept may have one or several super concepts. The super concept similarity is computed as follow:

$$\text{supSim}(C_P, C_R) = \frac{\text{allSubSim}(C_P, C_R)}{n}$$

where n is the number of pairs of matched super concepts. C_P and C_R are concepts from the two ontologies used by provider and requester, respectively. allSupSim is similarity of the matched super concepts. As shown in equation (2), the super concepts are matched one to one such that the average super concept match is maximized. This is done by using a recursive formula as follows:

$$\text{allSupSim(CP, CR)} = \text{Max(allSupSim(CP -SupP, CR -SupR))} + \text{conceptSimilarity(SupP, SupR))} \quad (2)$$

A concept may have one or more sub concepts. subSim is computed in the same way as supSim.

4 Experiment and Discussion

In this section, an experiment to determine concept similarity using two real world ontologies is introduced. This is followed by a description of how two web services which use the two ontologies are matched.

4.1 Experiments with Concept Similarity

For testing the concept similarity algorithm, we employ two ontologies from the real world taken from I3CON [11], the Information Interpretation and Integration Conference. Figure 4 shows two ontologies from two different OWL files: networkA.owl and networkB.owl. These two ontologies describe the nodes and connections in a local area network. networkA focuses more on the nodes themselves, while networkB encompasses the connections. Concept pairs were selected for testing arbitrarily but focused on pairs which have matching potential. The concept similarities are computed using equation (1). In most cases, w_i is set at 0.25. In some special cases, w_i is set differently. Table 2 shows the results of testing twelve concept pairs from the above two ontologies. The "expected results" column shows the "Ground Truth" result which is defined manually by human intelligence to determine closest fit. The "MOD Results" column shows the actual results which were obtained by using the proposed algorithm.

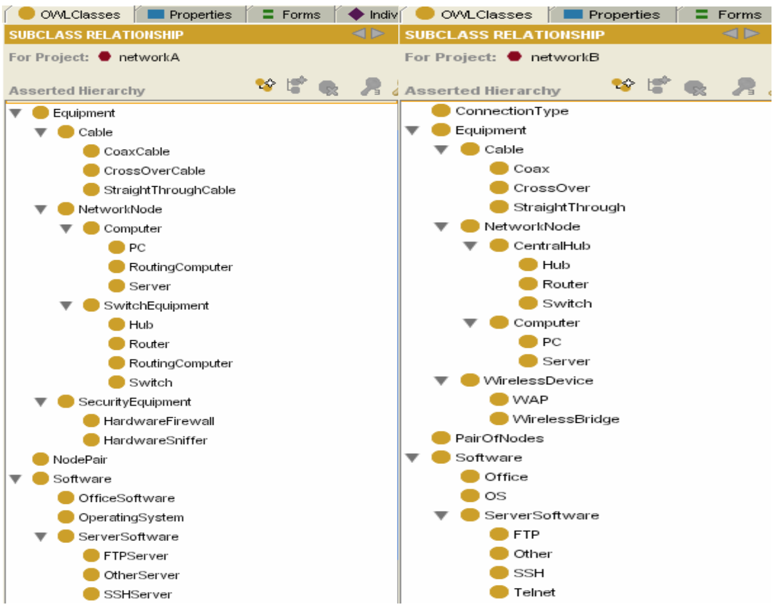


Figure 4. Structures of networkA.owl and networkB.owl in Protégé [1]

The details of matching are shown in figure 5. Each column pair in the figure shows the "expected" and MOD concept similarity. MOD results are computed based on the four similarity components: syntactic, properties, neighborhood, and domain similarity.

Table 2. Similarity between concepts

Case	Concepts from NetworkA.owl	Concepts from NetworkB.owl	Expected Results	MOD Results
1	NetworkNode	NetworkNode	0.80	0.78
2	Equipment	Equipment	0.75	0.65
3	PC	Router	0.60	0.58
4	NodePair	PairOfNode	0.57	0.63
5	Hub	NetWorkNode	0.25	0.25
6	Computer	Computer	0.80	0.82
7	CrossOverCable	Cable	0.70	0.55
8	Router	WAP	0.35	0.59
9	Server	Router	0.45	0.67
10	CoaxCable	Coax	0.72	0.7
11	CrossOverCable	CrossOver	0.72	0.75
12	StraightThroughCable	StraightThrough	0.72	0.78

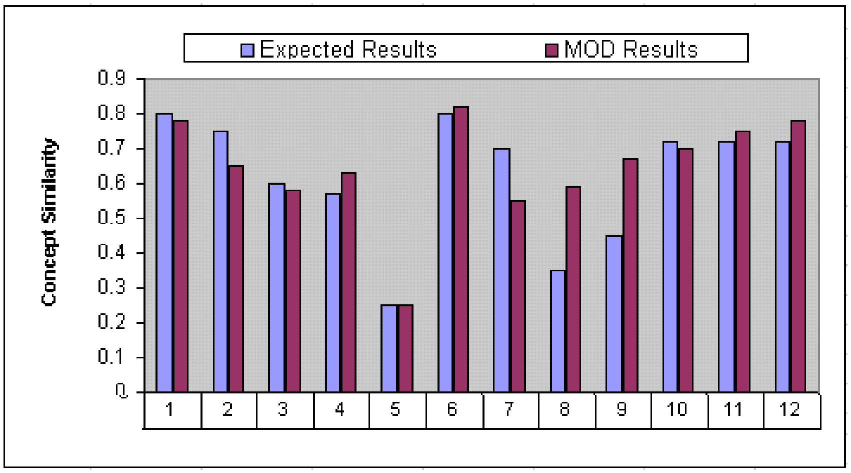


Figure 5. Graph represents concept similarities from Network A and Network B ontology

The syntactic matching results show that cases 1, 2, and 6 are 1 since their concept names are identical and the concept descriptions are nearly the same. The lowest syntactic similarity is in case 5 which involves two concepts: Hub and NetWorkNode. With the two concept names "Hub" and "NetWorkNode", WordNet returns a similarity is 0. But the similarity of the two concept descriptions is 0.2, so the syntactic similarity is 0.1. When no property is defined in one of the two concepts, their property similarity is 0. In this case, the algorithm will ignore the property dimension. In other words, the w_p in equation (1) is 0.

In most cases, the neighborhood similarity is high when their superclass and subclass similarity are high. This similarity is low in case 7 since CrossOverCable concept in ontology networkA matched Cable concept in ontology networkB. However, in ontology NetworkA, the Cable concept is the superclass of CrossOverCable concept. Therefore, the neighborhood similarity of the two concepts is the similarity of the superclass pair (Cable, Equipment). The subclass similarity is ignored because CrossOverCable does not have a subclass. The pair (Cable, Equipment) has super/sub class relationship, but in neighborhood similarity matching, the algorithm only considers the syntactic, property, and domain similarity. Therefore, the neighborhood similarity between CrossOverCable and Cable is low. Domain similarity is the similarity of two root concepts of the two ontologies. Therefore, the domain similarities in the twelve cases are the same.

The similarity of two concepts is computed as the average of the four components: syntactic, property, neighborhood, and domain similarity. Therefore, the value of each component will affect the final result. The contribution of the four components is equal in most cases. However, there are some cases where one component carries more value than the other. The neighborhood similarity for the pair (CrossOverCable, Cable) is low and has 'negative' effect on the final result because the matching algorithm does not recognize that they have a superclass/subclass relationship. It is therefore up to the user to select a suitable weight w_i to place on each of the components.

4.2 Matching Web Services

We assume that there are two companies: Cisco is a network company which sells network components. BCS is a network company which buys network components. Cisco and BCS develop web services to sell and buy network components, respectively. The inputs of the web services are price; the outputs of the web services are network components. To develop web services, companies must develop price, e-business, and equipment ontologies. We assume that the price and e-business ontology are popular, so the two companies use the same price and e-business ontology. But the equipment ontology is not available. So, they develop different ontologies for the equipment as shown in figure 4. The web services of the two companies are shown in figure 6. In this experiment, we assume that the threshold of concept similarity is very small, so all the concept similarities are larger then the threshold.

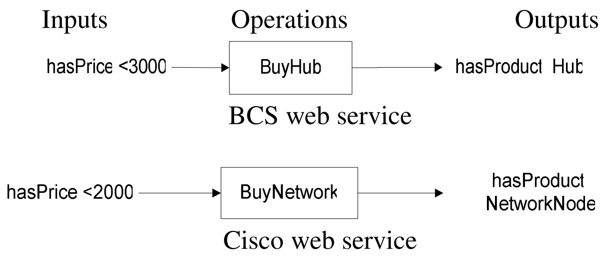


Figure 6. Web service requester and advertisement

Inputs of the web services are concepts from price ontology which is described in figure 7; We introduce concepts <3000, <2000, and <1000 that refer to prices that are less than 3000 units, 2000 units, and 1000 units, respectively.

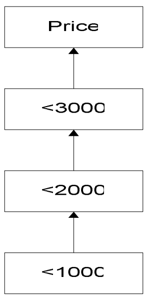


Figure 7. Ontology for price domain

Operations of web services are concepts from eBusiness ontology which is described in figure 8. The eBusiness provides Sell and Buy services. The Buy concept provides BuyComputer, BuyEquipment, and BuySoftware. BuyEquiment provides BuyHub, BuyRounter and BuySwitch.

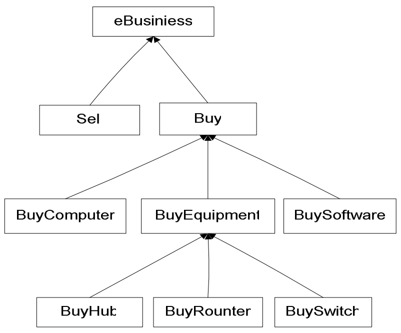


Figure 8. Ontology in eBusiness domain

With descriptions of the two web services and above ontologies, matching results are as follows:

- Input matching: This case is invert-subsumes since the input of the BSC service is a superclass of the input of Cisco service. Based on the definition in section 2, the input matching value is 0.3.
- Operation matching: This case is subsumes since the operation of the BSC service is the sub concept of the input of Cisco service. The operation matching value is 0.75.

- Output matching: In this case, the two web services use two concepts from different ontologies. Based on the results stated on section 4.1, the matching of two concepts Hub and NetworkNode is 0.25.
- User-defined: The user can define more rules or constraints to restrict the results. In this example, for simplicity, we assume that the user-defined matching is satisfied.

Based on the MOD algorithm, the similarity of the two web services is computed as follows:

$$S = \frac{(inpMatch + OutpMatch + OperMatch) * UserDefMatch}{3} = \frac{0.3 + 0.75 + 0.25}{3} = 0.43$$

The matching result of the two web services indicates that this is "subsume matching", that is, provider Cisco can satisfy the requester BCS but Cisco service is more general than BCS service.

5 Conclusions and Future Work

This paper has introduced an algorithm termed MOD which supports matching of web services using different ontologies. The algorithm is divided into four stages, namely, input, output, operation, and user-defined matching. The computing concept similarity which plays a key role in MOD has four main components, namely, syntactic, properties, domain, and neighborhood similarity. The experimental results confirm the viability of the discovery system. In the experiment, we used OWL-S ontologies to test the algorithm, but the MOD algorithm is general; therefore it can be used for web services using different semantic web service description languages such as DAML-S, and RDFS. However, the similarities defined in section 2 should not be the crisp sets; it should be defined as fuzzy sets. The domain similarity only considers the simple case when the ontology has one root. In the real word, the ontology may have more then one root. These problems will be considered in the future.

References

[1] Protégé - A tool for OWL editor.http://protege.stanford.edu/overview/

[2] Angell, R.; Freund, G., Automatic spelling correction using a trigram similarity measure, *Information Processing and Management*, (1983) **19**, 4, 255,

[3] Cardoso, J.; Sheth, A., Semantic e-Workflow Composition, *Journal of Intelligent Information Systems*, (2003) **21**, 3, 191 0925-9902, Kluwer Academic Publishers.

[4] DAML, Services Home Page, OWL-S markup of services.www.daml.org/services/owl-s/

[5] Jaeger, M. C.; Tang, S.; Liebetruth, C., The TUB OWL-S Matcher.Available at: http://ivs.tu-berlin.de/Projekte/owlsmatcher/index.html.

[6] Kawamura, T.; Blasio, J. D.; Hasegawa, T., et al., (2003), Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Register, *1st International Conference on Service Oriented Computing (ICSOC 2003)*, Trento, Italy, 208,

[7] Kawamura, T.; Blasio, J. D.; Hasegawa, T., et al., (2004), Public Deployment of Semantic Service Matchmaker with UDDI Business Registry, *3rd International Semantic Web Conference (ISWC 2004)*, **LNCS 3298**, 752,

[8] Li, L.; Horrocks, I., (2003), A software framework for matchmaking based on semantic web technology, *12th International World Wide Web Conference*, Budapest, Hungary, ACM Press, 331, 1-58113-680-3,

[9] LSDIS, METEOR-S: Semantic Web Services and Processes.http://lsdis.cs.uga.edu/projects/meteor-s/

[10] Miller, G., WordNet: An Electronic Lexical Database, The MIT Press: (May 15, 1998).

[11] NIST, I3CON Information Interpretation and Integration Conference.http://www.isd.mel.nist.gov/PerMIS_2004/index.htm

[12] Oundhankar, S.; K. Verma; Sivashanugam, K., et al., Discovery of web serivces in a Muti-Ontologies and Federated Registry Environment, *International Journal of Web Services Research*, (2005) **1**, 3,

[13] Paolucci, M.; Kawamura, T.; Payne, T. R., et al., (2002), Semantic Matching of Web services Capabilities, *1st International Semantic Web Conference (ISWC 2002)*, Sardinia, Italy, 333,

[14] Porter, M. F., An algorithm for Suffix Stripping, In *Morgan Kaufmann Multimedia Information And Systems Series*, Morgan Kaufmann Publishers: (1997); pp 313

[15] Salton, G., Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer, Addison-Wesley Ed. Massachusetts: (1988).

[16] Srinivasan, N.; Paolucci, M.; Sycara, K., (2004), Adding OWL-S to UDDI, implementation and throughput, *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA,

[17] Sycara, K.; J. Lu, M. K.; Widoff, S., Dynamic service matchmaking among agents in open information environments, *ACM SIGMOD Record (Special Issue on Semantic Interoperability in Global Information Systems)*,, (1999) **28**, No.1, 47,

[18] Sycara, K.; J. Lu, M. K.; Widoff, S., (March, 1999), Matchmaking among heterogeneous agents on the internet, *AAAI Spring Symposium on Intelligent Agents in Cyberspace*,

[19] Sycara, K.; Widoff, S.; M. Klusch, J. L., (2002), LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace, *Autonomous Agents and Multi- Agent Systems*, Vol.5, 173,

[20] W3C, Organization, OWL - Web Ontology Language Overview.http://www.w3.org/TR/owl-features/

[21] Zamora, E.; Pollock, J.; al, e., The Use of Trigram Analysis for Spelling Error Detection, *Information Processing and Management*, (1981) **6**, 17, 305,

On the Topological Landscape of Web Services Matchmaking

Hyunyoung Kil, Seog-Chan Oh, and Dongwon Lee[⋆]

The Pennsylvania State University
University Park, PA, USA
{hkil,seogchan,dongwon}@psu.edu

Abstract. Despite the rapid development in web services technologies, there has been little study on the public web services from network analysis perspective. To address this, in this paper, we have performed a topological study on various web service networks using real-world data sets. We first propose a flexible framework by which we can study semantic web services network "matchmaking" in a unified manner. Under the matching framework, then, we have conducted extensive experiments to study the characteristics of various web services networks. By and large, publicly available web services networks exhibit the typical characteristics of small world networks well and power law like distributions to some extent.

1 Introduction

The recent realization—the core web services specifications of WSDL [7], SOAP [17], and UDDI [8] are not sufficient to realize the goal of the *Semantic Web*—has developed a plethora of new means such as OWL [9], OWL–S [14], WSDL–S [16], and WSML [5] to annotate rich semantics to web services (ontology) so that truly automatic data exchange on the web is possible. Therefore, it is expected that, in the near future, service vendors will publish their offerings as "semantic web services" — semantically enriched web services.

However, in practice, there has been little study as to how useful current public web services are. In particular, since web services that are specified in WSDL and published in UDDI form a network, one can use network analysis methods to study the characteristics of the web services network. In general, the topological structure of a network affects the processes and behaviors occurring in the network. For instance, the topology of a social acquaintance network may affect the spread rate of gossip or disease. Similarly, the topology of the Internet is known to be correlated with the robustness of communication therein. Accordingly, understanding the structural properties of networks often help gain better insights and develop better algorithms. Therefore, in this paper, we aim at studying the web services network using network analysis techniques.

[⋆] The author is partially supported by Microsoft Scientific Data Intensive Computing (SciData) Award, 2005.

The focus of our study is on the web services specified in WSDL since currently there are not enough “semantic web services” specified in OWL–S or WSDL–S yet.Nevertheless, our matchmaking between web services is beyond syntactic matchmaking. By incorporating flexible matchmaking framework, we study the effect of semantic web services over real-world web services in WSDL. Our contributions of this paper are as follows:

– We propose a flexible web services matchmaking framework that allows to incorporate different “matching” in a unified manner. By using different matching functions, the framework can cover from exact to approximate to semantic matching.
– We propose three dimensions of granularity to form web services networks from parameter to operation to web services. By looking into web services network using different glasses, we can understand the characteristics of the network better.
– We have conducted extensive experimentation under the proposed framework, and applied network analysis techniques to study the distribution, average distance, diameter, and clustering coefficient of the network.

1.1 Background

For the efficient exposition of the paper, in this section, we present a simplified abstraction of web services specified in WSDL.

A *web service* in a WSDL file can be viewed as a collection of *operations*, each of which in turn consists of input and output *parameters*. When an operation op has input parameters $IN = \{p_1, ..., p_n\}$ and output parameters $OUT = \{q_1, ..., q_m\}$, we denote the operation by $op(IN, OUT)$. Furthermore, each parameter is a pair of *(name, type)*. We denote the name and type of a parameter p by $p.name$ and $p.type$, respectively. The WSDL specification lists four types of operations [7]: (1) *one-way*: the operation can receive a message but will not return a response; (2) *request-response*: the operation can receive a request and will return a response; (3) *solicit-response*: the operation can send a request and will wait for a response; and (4) *notification*: the operation can send a message but will not wait for a response. In order to decide if an operation op_1 can invoke an operation op_2, one needs to check if the types of op_1 and op_2 are compatible or not. Since this can be done trivially, from here forward, we assume that types of operations are all compatible, and focus on other issues instead.

For instance, consider the following web service w:

```
<message name='findRestaurant_Request'>
  <part name='zip' type='xs:integer'>
  <part name='foodPref' type='xs:string'>
</message>
<message name='findRestaurant_Response'>
  <part name='name' type='xs:string'>
  <part name='phone' type='xs:integer'>
</message>
```

```
<portType name="allRestaurant">
  <operation name="findRestaurant">
    <input message="findRestaurant_Request"/>
    <output message="findRestaurant_Response"/>
  </operation>
</portType>
```

The web service, w, is a request-response type, and consists of a single operation, `findRestaurant`, that takes two input parameters, p_1=(zip, integer) and p_2=(foodPref, string), and returns two output parameters, q_1=(name, string) and q_2=(phone, integer). Therefore, a client program wishing to get the name and phone number of a restaurant may invoke `findRestaurant`(16801,"Thai") to w.

1.2 Related Work

Many empirical networks are well modeled by complex networks including the scale-free and small-world networks. The small-world networks are generated by the classical Watts-Strogatz network model [23]. Albert, Barabasi and Jeong [1] have proposed a set of different models for generating scale-free networks, based on the growing process of the Internet and other empirical complex networks. Denning [10] surveyed various network laws with a focus on the power law distribution and the scale-free networks. In this paper, we apply the developed techniques to examine the semantic web services networks.

The problem of matching descriptions of two services is related with the classical problem of the *schema matching* in Database. For instance, [20] presents an array of latest techniques on the schema matching, and some of them could be used in our study. Note that we do not propose a particular matchmaking method in this paper. Instead, we advocate a flexible framework that can accommodate a plethora of matchmaking schemes in a unified manner. Then, based on the framework, we studied various topological properties of web services network. Due to the availability, for instance, we happen to use Cosine and WordNet based matching schemes. However, it is also possible to incorporate the aforementioned matching approaches such as [24, 22] or schema matching techniques in [20] in our framework.

Current web service matchmaking solutions are based on the keyword matching supported by the category-browsing of UDDI. However, the keyword-based matching ignores the real functions of web services. To address this limitation, researchers have developed a set of methods which assess the similarity of web services to achieve matchmaking. Wu [24] suggested a matchmaking process based on a lightweight semantic comparison of signature specifications in WSDL by means of several assessment methods. Wang and Stroulia [22] assessed the similarity of the requirement description of the desired service with the available services via the semantic information-retrieval (IR) method and a structure-matching approach. Maedche and Staab [13] provide multiple-phase cross-evaluation to assess the similarity between two different ontology. An excellent survey of modern matchmaking algorithms and their applications to the web service matching field is available in [24]. Finally, some of the recent ontology-based service matchmaking works such as [2, 15, 3] are relevant to our research too. We leave the extension of our framework to accommodate those recent developments as future work.

2 The Matchmaking Framework

In this Section, we formalize the notion of matchmaking on web services.

2.1 Flexible Matching

Network consists of nodes and edges. In our framework, different entities can be used as nodes and edges in a unified manner. First, as nodes, we consider three kinds – parameters, operations, and web services – from finer to coarser granularity. Second, as edges, we use the notion of parameter matching and operation invocation.

When the meanings of two parameters, p_1 and p_2, are interchangeable, in general, two parameters are said to be matching each other. The simplest way to check this is if two parameters have the *same* name and type: $(p_1.name = p_2.name) \wedge (p_1.type = p_2.type)$. Since web services are designed and created in isolation, however, this naive matching is often too rigid and thus misses cases like p_1=("password", string) and p_2=("passwd", string). On the other hand, if web services are annotated with rich semantics (e.g., using RDF [4] or WSDL-S [16]), then the so-called "semantic" matching can be easily reasoned out. However, in practice, majority of public web services do not have annotated semantics yet.

In order to cover all the spectrum of matching, therefore, we propose a generic boolean function, `match(`p_1`, `p_2`)`, that determines if two parameters p_1 and p_2 are matching or not. Formally,

Definition 1 (type-match) *A boolean function,* `type-match`*($p_1.type$, $p_2.type$), returns True if: (1) $p_1.type = p_2.type$, or (2) $p_1.type$ is derived from $p_2.type$ in a type hierarchy.* □

Definition 2 (name-match) *A boolean function,* `name-match`*($p_1.name$, $p_2.name$, $\mathcal{D}$, θ), returns True if the distance between $p_1.name$ and $p_2.name$ by a distance metric $\mathcal{D}$ is below the given threshold θ: i.e., $\mathcal{D}(p_1.name, p_2.name) \leq \theta$.* □

For instance, `name-match`("password", "passwd", =, 1) represents the exact matching, and would return False since "password" $\neq$ "passwd"[1]. Similarly, by using string matching functions such as Edit distance [25] or cosine metric, one can express the approximate name matching. For instance, `name-match`("password", "passwd", edit-distance, 3) would return True since two names of parameters have the edit distance of 2 (< 3). In the experimentation, we use three metric functions: = (i.e., literal equality), cosine distance with

[1] For the exact matching, the threshold value other than 1 is not useful.

TF/IDF weights [21], and WordNet [12] based semantic distance. However, note that it is also possible to incorporate the aforementioned matching approaches such as [22] and [24] in our framework, if the implementation is available. Based on two boolean functions, now, we define a generic parameter match function as follows:

Definition 3 (match) *A boolean function,* `match`*(p_1, p_2, $\mathcal{D}$, θ), returns True if: (1)* `name-match`*(p_1.name, p_2.name, $\mathcal{D}$, θ) = True, and (2)* `type-match`*(p_1.type, p_2.type) = True.* $\Box$

Definition 4 (Parameter Matching) *When a boolean function,* `match`*(p_1, p_2, $\mathcal{D}$, θ), returns True, it is said that a parameter p_1* **matches** *a parameter p_2 (i.e., p_1 can be used in the place of p_2), and written as "$p_1 \sim p_2$".* $\Box$

In order to invoke an operation op_1 with input parameters IN=$\{p_1, ..., p_n\}$, one may need to provide values for all required (i.e., mandatory) input parameters. When one can provide all required input parameters, op_1 is said "fully invocable". When one can provide at least one required input parameter, op_1 is said "partially invocable", so partially invocable operations includes a set of fully invocable operations. For instance, Google API has a search operation with several input parameters, but only part of them are mandatory. Similarly, consider a client program wishing to invoke an operation op_1 first and then have op_1 invoke another operation op_2 directly (i.e., a case of web services composition). In this case, if the output parameters of op_1 can satisfy all required input parameters of op_2, then op_1 "fully" invokes op_2, and if there exists any output parameter of op_1 satisfying any input parameter of op_2, then op_1 "partially" invokes op_2. Formally,

Definition 5 (Operation Invocation) *For two operations, $op_1(IN_1, OUT_1)$ and $op_2(IN_2, OUT_2)$,*

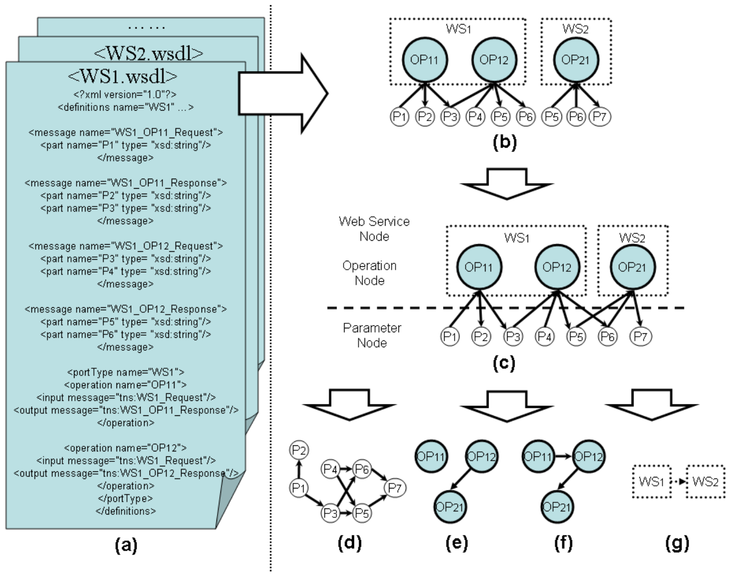
- *op_1 fully invokes op_2, denoted by "$op_1 \rightarrowtail op_2$", if for every mandatory input parameter $p \in IN_2$, there exists an output parameter $q \in OUT_1$ such that $q \sim p$.*
- *op_1 partially invokes op_2, denoted by "$op_1 \dashrightarrow op_2$", if there exists a mandatory input parameter $p \in IN_2$ and an output parameter $q \in OUT_1$ such that $q \sim p$.*

We denote each invocation by **full-** *and* **partial-invocation***, respectively.* $\Box$

2.2 Web Service Network Model

In this Section, using the notions of *nodes* and *edges* defined in Section 2.1, we propose a flexible web service network model as follows.

Definition 6 (Web Service Network Model) *A web service network is generated by a 4-tuple model $\mathcal{M}$=($\mathcal{T}$, $\mathcal{D}$, θ, $\mathcal{I}$), where:*



```
<WS1.wsdl>
<?xml version="1.0"?>
<definitions name="WS1" ...>

<message name="WS1_OP11_Request">
<part name="P1" type= "xsd:string"/>
</message>

<message name="WS1_OP11_Response">
<part name="P2" type= "xsd:string"/>
<part name="P3" type= "xsd:string"/>
</message>

<message name="WS1_OP12_Request">
<part name="P3" type= "xsd:string"/>
<part name="P4" type= "xsd:string"/>
</message>

<message name="WS1_OP12_Response">
<part name="P5" type= "xsd:string"/>
<part name="P6" type= "xsd:string"/>
</message>

<portType name="WS1">
<operation name="OP11">
<input message="tns:WS1_Request"/>
<output message="tns:WS1_OP11_Response"/>
</operation>

<operation name="OP12">
<input message="tns:WS1_Request"/>
<output message="tns:WS1_OP12_Response"/>
</operation>
</portType>
</definitions>
```

Fig. 1. Web service networks: (a) WSDLs, (b) Conceptual web service network, (c) Web service networks from diverse models, (d) Parameter node network, $\mathcal{M}_p$, (e) Fully invocable operation node network, $\mathcal{M}^f_{op}$, (f) Partially invocable operation node network, $\mathcal{M}^p_{op}$, and (g) Web service node network, $\mathcal{M}_{ws}$

- *$\mathcal{T}$ is the type of node and can be either* `p` *for parameter,* `op` *for operation, or* `ws` *for web service*
- *$\mathcal{D}$ (and θ resp.) is the distance metric to be used in parameter matching (and its threshold resp.)*
- *$\mathcal{I}$ is the type of operation invocation and can be either* `FI` *for full invocation or* `PI` *for partial invocation* □

Example 1. A model $\mathcal{M}_1$=(op, cosine, 0.75, FI) generates an operation node network where parameter matching is done using cosine metric with a threshold 0.75. Furthermore, an edge from an operation op_1 to op_2 is added only when $op_1 \rightarrowtail op_2$. □

Example 2. A model $\mathcal{M}_2$=(ws, word-net, 0.9, PI) generates a web service node network where intra-parameter matching is done using WordNet based semantic distance metric with a threshold 0.9. Furthermore, an edge from a web service ws_1 to ws_2 is added if $op_1 \dashrightarrow op_2$ for op_1 ($\in ws_1$) and op_2 ($\in ws_2$) – that is, partial invocation among operations. □

Consider Figure 1 as an example. Here, a web service network is formed by a set of web services, each of which consists of a set of operations. An operation is invoked with a set of input parameters and produces a set of output parameters.

There are three kinds of nodes representing web services (e.g., ws_1 and ws_2), operations (e.g., op_{11}, op_{12} and op_{21}) or parameters (e.g., $p_1, \cdots, p_7$). Each web service node containing a set of operation nodes is connected to parameter nodes with "directed" edges. These edges show the flow of the parameters as inputs or outputs of operations. An edge from a parameter node p to an operation node op indicates that the parameter p is used as one of inputs of the operation op. On the other hand, an edge from an operation node op to a parameter node p indicates that the operation op produces the parameter p as an output. For example, p_1 is one of inputs of op_{11} in ws_1. Similarly, p_5 is not only an output of op_{12} but also an input of op_{21} in ws_2.

In order to study subtle difference among node types, one can project out the aforementioned web service network into three kinds as follows:

- A *parameter node network*, i.e., $\mathcal{M}_p$=($\texttt{p}$, $\mathcal{D}$, θ, $\mathcal{I}$), consists of parameter nodes and edges representing operations that has the source node as an input parameter and the target node as an output parameter. For instance, if $op_1(IN_1, OUT_1)$ exists, then we create edges from each parameter p of IN_1 to every output parameter q of OUT_1. When approximate matching or semantic matching is used for intra-parameter matching, each node in $\mathcal{M}_p$ is in fact a set of similar nodes. That is, suppose there is an edge from p_1(password, string) to p_3(firstName, string) in $\mathcal{M}_p$. This indicates that one can retrieve "firstName" information by providing "password" information. Now, if an approximate matching method (e.g., edit distance) or semantic matching method determines that a parameter p_1(password, string) matches a parameter p_2(passwd, string), then p_2.name is merged with p_1.name to form an edge: {password, passwd} $\rightarrow$ firstName. That is, the source node is a set of two nodes.
- An *operation node network*, i.e., $\mathcal{M}_{op}$=($\texttt{op}$, $\mathcal{D}$, θ, $\mathcal{I}$), consists of nodes representing operations and edges representing the invocation in-between. That is, if an operation op_1 can (either partially or fully based on $\mathcal{I}$) invoke an operation op_2, there is an edge $op_1 \rightarrow op_2$ in $\mathcal{M}_{op}$.
- A *web service node network*, i.e., $\mathcal{M}_{ws}$=($\texttt{ws}$, $\mathcal{D}$, θ, $\mathcal{I}$), consists of all web service nodes and directed edges representing the existence of invocable operations between web services.

The Figure 1(d) describes a parameter node network $\mathcal{M}_p$ based on Figure 1(b). For instance, p_1 can be transformed to p_2 by op_{11}. For operation node example, op_{21} can be invoked with two input parameters both of which are produced by op_{12} in Figure 1(c). Therefore, there exists an edge $op_{12} \rightarrow op_{21}$ in both operation node networks (Figure 1(e) and Figure 1(f)). On the other hand, p_3, one of outputs of op_{11}, is used as one of input of op_{12}. However, op_{12} requires another input parameter data p_4 not produced by op_{11}. In the partial invocation mode, therefore, an edge $op_{11} \rightarrow op_{12}$ is added. On the contrary, in the full invocation mode, no edge is added. Note that in the example of Figure 1, we get the same web service node network whether we use partial or full invocation mode. However, in general, different networks will be formed depending on the choice of invocation mode.

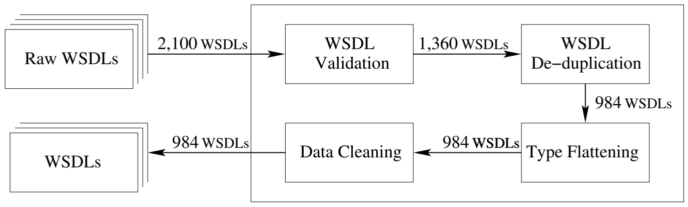


Fig. 2. Overview of pre-processing.

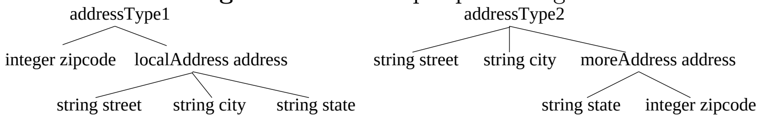


Fig. 3. Two compatible types with different structures.

3 Topological Landscape

3.1 Set-Up

The pre-processing for the experimentation consisted of five steps:

1. *Data Gathering*: A total of 2,100 real-world WSDL files were gathered from two sources. First, 1,554 files were taken from Fan et al. [11] who downloaded the files from public repositories such as `XMethods.org` or `BindingPoint.com`. Second, out of top-1000 ranked WSDL files from Google for the query string "`wsdl filetype:wsdl`", 546 were downloaded using Google API (the rest were un-downloadable).
2. *WSDL Validation*: According to WSDL standard, 740 invalid WSDL files were removed, and 1,360 files are left out.
3. *WSDL De-duplication*: 376 duplicate WSDL files at operation level were removed, yielding 984 valid WSDL files.
4. *Type Flattening*: In matching parameters, we use both name and type of parameters. However, since WSDL files are designed by different people in isolation, using only atomic types of XML Schema can be too rigid. For instance, consider two parameters: p_1(address, `addressType1`) and p_2(MyAddress, `addressType2`) of Figure 3. Although names of two parameters are similar, their types are user-defined and different. However, two types are in fact "compatible". Therefore, if we *flatten* two types into p_1.type={`integer` zipcode, `string` street, `string` city, `string` state} and p_2.type={`string` street, `string` city, `string` state, `integer` zipcode}, then two parameters can be matched. We call this process as type flattening[2]. After type flattening, then each atomic type and name are compared using type hierarchy of XML Schema and flexible matching scheme (e.g., exact or approximate), respectively. The resulting statistics of type flattening is shown in Table 1.

[2] Alternative to our type flattening is to use more expensive metric such as tree edit distance. For its simplicity, we used type flattening in this paper.

Table 1. Type distribution.

Type	Before Flattening		After Flattening	
	input(num)	output(num)	input(num)	output(num)
`anyType`	6.79% (7)	9.03% (5)	0.2% (35)	0.2% (51)
`simpleType`	65.52% (6,751)	32.37% (1,792)	92.43% (19,809)	90.63% (22,946)
`string`	53.75% (5,538)	22.00% (1,218)	65.06% (13,943)	54.74% (13,859)
`number`	7.79% (803)	7.12% (394)	17.69% (3,791)	22.89% (5,796)
`time`	0.90% (93)	0.22% (12)	1.85% (396)	2.93%(743)
`boolean`	3.74% (385)	2.46%(136)	7.16% (1,535)	9.14% (2,314)
`complexType`	34.41% (3,545)	67.54% (3,739)	7.41% (1,588)	9.16% (2,320)
Total Number	10,303	5,536	21,432	25,317

5. *Data Cleaning*: The final step is cleaning data to improve the quality of parameters. For instance, substantial number of output parameters (16%) were named “return”, “result”, or “response” which is too ambiguous for clients. However, often, their more precise underline meaning can be derived from contexts. For instance, if the output parameter named “result” belongs to the operation named “getAddress”, then the “result” is in fact “Address”. In addition, often, naming follows apparent pattern such as `getFooFromBar` or `searchFooByBar`. Therefore, to replace names of parameters or operations by more meaningful ones, we removed spam tokens like “get” or “by” as much as we could.

For flexible matching, we used: (1) *Exact Matching*; (2) *Cosine Distance*. For instance, when we compare two parameters p_1(“BirthDay”, string) and p_2(“B-day”, string), names are segmented to a set of tokens with TF/IDF weights like p_1({Birth, Day}, string) and p_2({B, Day}, string). Then, by converting both sets into vectors of v and w, their distance is measured as $\cos(\theta) = \frac{v \cdot w}{\|v\| \cdot \|w\|}$; (3) *WordNet-based Distance*. Since WordNet is a network of English words labeled with semantic classes (e.g., synonyms), it carries various semantic relations among words. People have proposed various ways to measure the “semantic distance” of two words in WordNet [6]. We use the one by Lin [12] that scales the information content of the least common subsumer by the sum of the information content of two vocabularies. Both cosine and WordNet metrics used threshold to determine the matchmaking (i.e., if the calculated distance is above threshold, new edge is added into the network).

At the end, we have generated a total of 25 (= 5 network types × 5 distance metrics) web services networks: (1) three kinds of networks – $\mathcal{M}_p$, $\mathcal{M}_{op}^f$, and $\mathcal{M}_{ws}^f$, and two of their counterparts of “partial invocation” – $\mathcal{M}_{op}^p$ and $\mathcal{M}_{ws}^p$; (2) five distance variations – Exact, Cosine (0.75), Cosine (0.95), WordNet(0.75), and WordNet(0.95).

Table 2. Small world properties of giant components in public web services.

Matching Scheme	Network	L_{actual}	L_{random}	C_{actual}	C_{random}
Exact matching	$\mathcal{M}_p$	4.31852	3.42441	0.2229	0.0021
	$\mathcal{M}_{op}^p$	2.8590	1.9830	0.3056	0.0362
	$\mathcal{M}_{op}^f$	3.7605	2.5628	0.2147	0.0180
	$\mathcal{M}_{ws}^p$	2.2710	1.9222	0.4809	0.0874
	$\mathcal{M}_{ws}^f$	2.9659	2.4250	0.2610	0.0405
Cosine (0.95)	$\mathcal{M}_p$	4.1760	3.4442	0.2324	0.0022
	$\mathcal{M}_{op}^p$	2.8651	1.9881	0.3125	0.0340
	$\mathcal{M}_{op}^f$	3.7538	2.5787	0.2001	0.0173
	$\mathcal{M}_{ws}^p$	2.2847	1.9254	0.4925	0.0833
	$\mathcal{M}_{ws}^f$	3.0046	2.4803	0.2499	0.0359
Cosine (0.75)	$\mathcal{M}_p$	4.1981	3.4730	0.2397	0.0020
	$\mathcal{M}_{op}^p$	2.8671	1.9925	0.3190	0.0326
	$\mathcal{M}_{op}^f$	3.7392	2.5910	0.1990	0.0172
	$\mathcal{M}_{ws}^p$	2.2923	1.9307	0.4822	0.0801
	$\mathcal{M}_{ws}^f$	2.9990	2.4394	0.2392	0.0375
WordNet (0.95)	$\mathcal{M}_p$	3.6088	3.3282	0.2612	0.0027
	$\mathcal{M}_{op}^p$	2.4234	1.9425	0.3251	0.0574
	$\mathcal{M}_{op}^f$	3.4165	2.4440	0.1493	0.0190
	$\mathcal{M}_{ws}^p$	2.1222	1.8865	0.5290	0.1138
	$\mathcal{M}_{ws}^f$	2.6215	2.1839	0.2527	0.0510
WordNet (0.75)	$\mathcal{M}_p$	3.2656	3.3665	0.3118	0.0030
	$\mathcal{M}_{op}^p$	2.0546	1.8743	0.4818	0.1256
	$\mathcal{M}_{op}^f$	2.6506	1.9657	0.1842	0.0429
	$\mathcal{M}_{ws}^p$	1.8484	1.7790	0.6697	0.2214
	$\mathcal{M}_{ws}^f$	2.2226	1.9023	0.3487	0.0993

3.2 Small World

First, we examine if web services network exhibit small world properties. In general, networks or graphs are called *small world* networks if they show the properties of both *random* and *regular* networks.

Definition 7 (Random Network) *$G_{(N,p)}$ is a random network of N nodes, if each pair of nodes is connected with the probability p. As a result, edges are randomly placed among a fixed set of nodes.* □

Definition 1 (Regular Network). *$Rg_{(N,k)}$ is defined as the regular network on N nodes, if node i is adjacent to nodes $[(i+j) \text{ mod } N]$ and $[(i-j) \text{ mod } N]$ for $1 \leq j \leq k$, where k is the number of valid edge of each node. If $k = N-1$, $R_{(N,k)}$ becomes the complete N-nodes graph, where every node is adjacent to all the other $N-1$ nodes.* *None*

Random networks are characterized by their short average distances among reachable nodes. On the other hand, in regular networks, each node has highly clustered neighbor nodes, such that the connectivity between neighboring nodes are very high. Consequently, small world networks show both (1) highly clustered structure and (2) small shortest distance.

- L: The average shortest distance (i.e., number of hops) between reachable pairs of vertices. $L(p)$ is the L of Watts-Strogatz graph [23] with probability p. L_{random} is identical to $L(1)$.
- C: The average clustering coefficient. For a node i with v_i neighboring nodes, $C_i = \frac{2E_i}{v_i(v_i - 1)}$, where E_i is the number of edges between v_i neighbors of i. C is the average clustering coefficient C_i for a network. Again, $C(p)$ is defined as C of the Watts-Strogatz graph with the probability p. C_{random} is identical to $C(1)$.
- $Index_{SN}$: The small world network index is defined as:

$$Index_{SN} = \frac{|C_{actual} - C_{random}|}{|L_{actual} - L_{random}|} \tag{1}$$

  where C_{actual} and L_{actual} represent C and L of the measured network, respectively, and both C_{random} and L_{random} represent C and L of the random graph with the same number of nodes and average number of edges per node as the measured network.

If a network has the small world properties, then its L and C shows: $C \gg C_{random}$ and $L \gtrsim L_{random}$. Therefore, the more distinct the small world properties of a network are, the bigger $Index_{SN}$ of the network becomes.

Table 2 shows the average shortest path, L and clustering coefficient, C for giant components extracted from each of the 25 web service networks, compared to random graphs with the same number of nodes and average number of edges per node. Note that we treat all edges of each network as undirected and unweighted[3]. It is interesting that all web services networks are the small world network by showing $L \gtrsim L_{random}$ and $C \gg C_{random}$. This suggests that the real web services have *short-cuts* that connect nodes of networks. Otherwise, nodes of networks would be much farther apart than L_{random}.

There are distinct changes between different matching schemes in terms of the small world properties. For example, in the parameter node network $\mathcal{M}_p$, L(=3.2656) of the WordNet(0.75) is much smaller than L(=4.318) of the exact matching, while C(=0.3118) of the WordNet(0.75) is much bigger than C(=0.222) of the exact matching. For better understanding, Figure 4 illustrates the changes of $Index_{SN}$ for different matching schemes. Recall that $Index_{SN}$ tends to increase as C increases or L decreases. In the Figure, we can see that the usage of semantic matching scheme, WordNet(0.75) generates more distinct small world properties than the exact and cosine distance matching schemes. In the case of $\mathcal{M}_p$ with the WordNet(0.75) matching applied, $Index_{SN}$ is ten

[3] This crude approximation is often acceptable since our goal is to study the network topology, not the flow of materials on edges (e.g., information, electricity). Similar assumption was made by Watts and Strogatz in [23]. In Section 3.3, directed network cases are analyzed further.

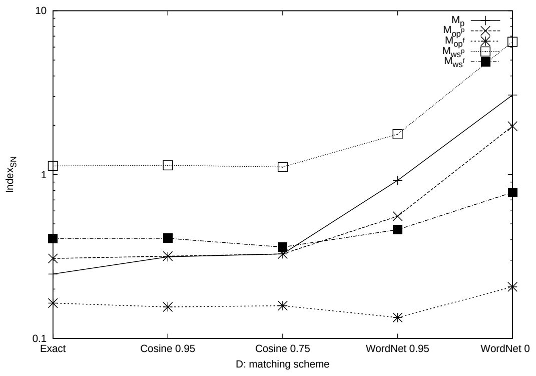


Fig. 4. $Index_{SN}$ changes.

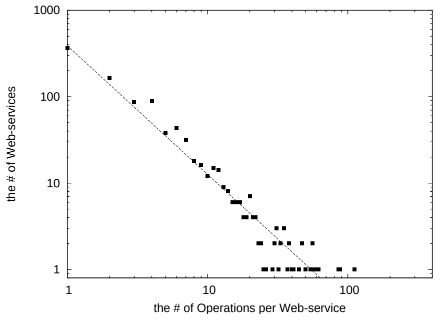
times bigger than that of the exact matching case. This result suggests that the semantic matching scheme makes the network more dense with the increased number of edges, so that L decreases while C increases. From the web service composition perspective, this result suggests that semantic matching scheme can facilitate more productive and effective service compositions than when only exact syntactic matching is used.

3.3 Power Laws

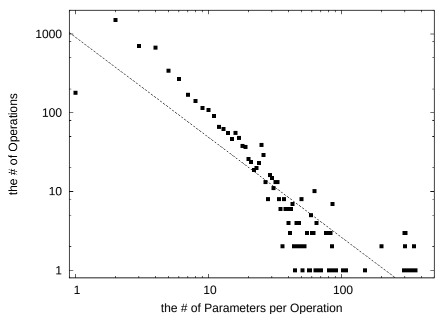
A power law distribution often occurs in complex systems where a majority of nodes have very few connections, while a few nodes have a high degrees. The existence of power law distribution has been observed in many real and artificial networks such as power grid, WWW, or collaboration network, and believed to be one of signs of mature and robust networks [10, 19]. In this section, we examine if semantic web services network follows power law distribution or not.

First, we examine how *complex* web services are in general. One way to measure the complexity of web services is to measure how many operations (parameters resp.) are involved in each web service (operation resp.) [11]. These results are shown in Figure 5, fitted to a power-law function $y = Cx^{-\alpha}$ (in log-log plot). They show *near*[4] power law distribution with the exponent of 1.49 and 1.27, respectively. In Figure 5(a), 37% of the web services have just one operations and 71% of the web services have less than 5 operations. On the other hand, the largest web services have over 110 operations. Similarly, in Figure 5(b), 181 operations (among 5,180 operations) have less than two input/output parameters. In addition, 194 operations have no input parameter and 255 operations have no output parameter. However, the whole distribution also shows a power law like property. After type flattening, around 65% of operations have less than

[4] A recent study [18] reported that most of real-world power law distributions exhibited an exponent of $2 \leq \alpha \leq 3$.

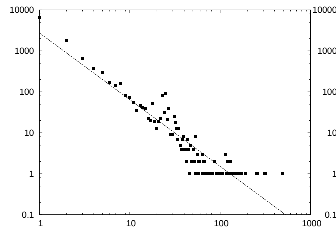


(a) # of operations per web service ($\alpha = 1.49$)

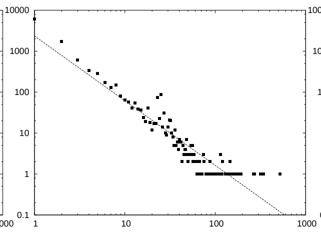


(b) # of parameters per operation ($\alpha = 1.27$)

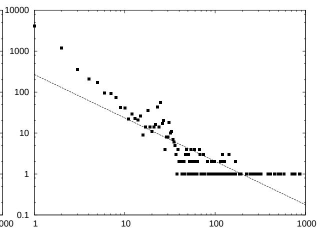
Fig. 5. The complexity of web services and operations.



(a) Exact Matching α=1.63



(b) Cosine (0.75) α=1.58



(c) WordNet (0.75) α=1.06

Fig. 6. The popularity of parameter names. X-axis is the frequency of parameter names while Y-axis is the number of samples.

6 parameters while the maximum number of the parameters in an operation is 360.

Second, we examine the *popularity* of parameter names – some parameter names occur more often than others. X-axis is the frequency of parameter names while Y-axis is the number of samples. That is, a coordinate (10, 100) indicates that there are 100 number of parameter names that occur 10 times. Again, all of them show near power law distributions. From these distributions, we can observe a very small number of popular "hub" parameter names. For instance, a parameter p_1 ("*name*", *string*) appears most frequently in our collection. In addition, as the matching method gets more flexible (from left to right in Figure 6), more parameters get to match each other. Since the matched parameters are considered as a single parameter, the number of distinct parameters gets smaller but the frequency of parameter names gets bigger. For example, while p_1 appears 490 times in Exact matching with 11,301 distinct parameter names, p_1 appears 3,278 times in WordNet (0.75) matching with 7,042 distinct parameter names, where similar parameters like p_2("*name*", *string*) or p_3("*identification*", *string*)

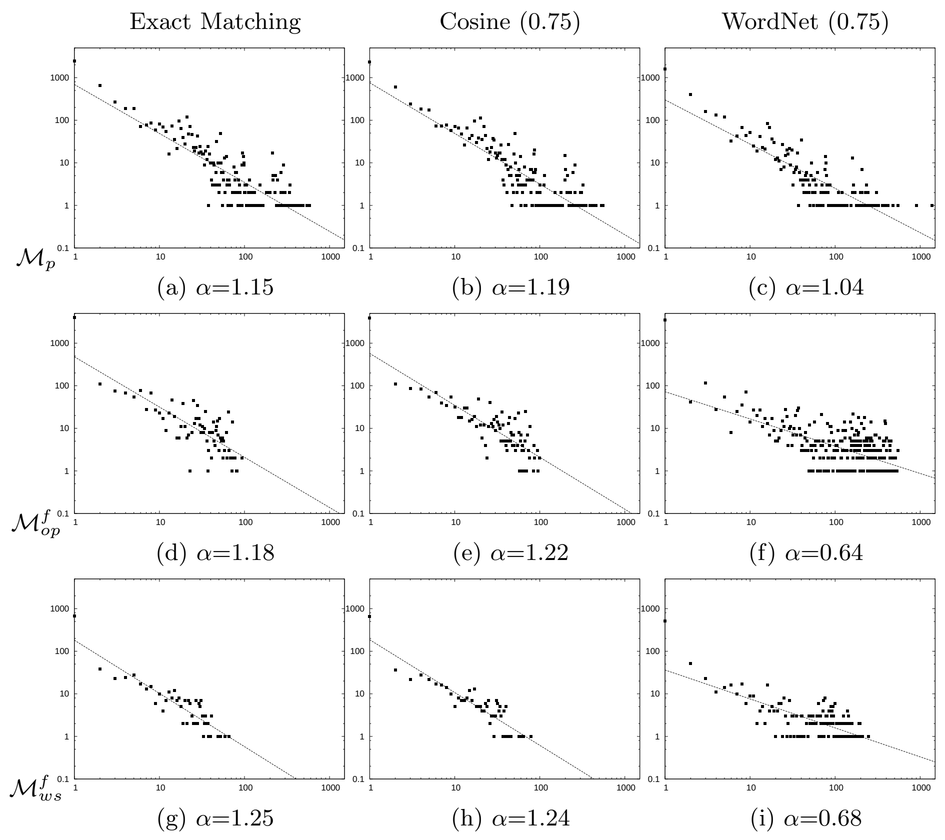


Fig. 7. Out-degree distribution of three web service networks (rows) of three matching schemes (columns). X-axis is out-degree, and Y-axis of the first, second, and third rows is the number of parameter, operation, and web service nodes, respectively.

are considered to be a "match" and consolidated. Therefore, Figure 6(c) has the the smallest α and the largest tail among three.

In Figure 7, next, we examine the out-degree distributions of three web service networks of three matching schemes. Unlike previous figures, by and large, out-degree distributions no longer follow power law distributions well (although we still fit them with power law functions). Nevertheless, it is evident to see the existence of hub parameters, operations, or web services with huge number of out-degrees while majority has only a few. In the parameter node networks of $\mathcal{M}_p$ (first row of Figure 7), due to flexible matching, the number of nodes decrease from left to right: 11,301 for Exact matching, 10,568 for Cosine (0.75), and 7,042 for WordNet (0.75). However, even though Figure 7(c) has the smallest number of nodes of 7,042, it has the largest number of edges and biggest out-degree. For example, a parameter node including ("*name*", *string*) has 317 out-degrees (to 2.7% of nodes) in Exact matching, 306 out-degrees (to 2.9% of nodes) in

Consine (0.75), and 1,378 out-degrees (to 19.57% of nodes) in WordNet (0.75). The distributions between Figure 7(a) and Figure 7(b) show little difference with respect to the number of nodes.

Unlike parameter node networks of $\mathcal{M}_p$, all operation node networks of $\mathcal{M}^f_{op}$ (second row of Figure 7) have 5,180 fixed operation nodes, but, from left to right, the total number of edges becomes bigger from 22,253 to 25,174 to 184,429. The fact that approximate matching scheme such as Cosine or semantic matching scheme such as WordNet have more out-degrees indicates that information can flow more flexibly among web services operations through many edges. In the last row of Figure 7, the web services node networks of $\mathcal{M}^f_{ws}$ also have a fixed number of 984 nodes, and out-degree increases from left to right. While some edges in operation node networks include intra-connections within the same web service, the edges of $\mathcal{M}^f_{ws}$ are amount to pure inter-connection among different web services. Consequently, more cooperation with different web services can be expected as we have more flexible semantic matching. Since one can invoke more number of operations, in general, it gets easier to combine and compose multiple web services [19].

Finally, we also examined the difference between partial vs. full invocation in invoking operations. Due to space constraint, we do not present the results. The finding is consistent with previous cases in that partial invocation based network gets more edges for its more flexible matching that full invocation based one.

4 Conclusion

In this paper, we have studied the topology of web services network using real-world data sets. To find out topological properties of the networks, we performed extensive experiments on various node networks under a flexible matchmaking framework that covers exact, approximate, and semantic matching. According to our experiments, we have learned that: (1) Public web service networks show small world network property well and follow a power law like distribution pattern to some degree. However they do not fully show the maturity seen in other complex networks yet; and (2) Semantic matching among web services generates networks with higher degrees than exact or approximate matching does. This suggests that more flexible web services composition can be achieved with the increased semantics in web services matchmaking.

References

[1] R. Albert and A.-L. Barabasi. "Topology of Evolving Networks: Local Events and University". *Phys. Rev. Lett.*, 85:5234–5237, 2000.

[2] V. De Antonellis, M. Melchiori, L. De Santis, M. Mecella, E. Mussi, B. Pernici, and P. Plebani. "A Layered Architecture for Flexible Web Service Invocation". *Software: Practice and Experience*, 36(2):191–223, 2005.

[3] D. Bianchini, V. De Antonellis, B. Pernici, and P. Plebani. "Ontology-based Methodology for E-service Discovery". *Information Systems*, 31(4):361–380, 2006.

[4] D. Brickley and R. V. Guha (Eds). "Resource Description Framework (RDF) Schema Specification 1.0". W3C Recommendation, Mar. 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[5] J. Bruijn (Ed.). "The Web Service Modeling Language WSML", 2005. http://www.wsmo.org/wsml/wsml-syntax.

[6] A. Budanitsky and G. Hirst. "Evaluating WordNet-based Measures of Semantic Distance". *Computational Linguistics*, 32(1):13–47, 2006.

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. "Web Services Description Language (WSDL) 1.1". W3C Recommendation, 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[8] L. Clement et al. (Ed.). "UDDI Version 3.0.2", 2004. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm.

[9] M. Dean and G. Schreiber (Ed.). "OWL Web Ontology Language Reference", 2004. http://www.w3.org/TR/owl-ref/.

[10] P. J. Denning. "Network Laws". *Comm. ACM*, 47(11):15–20, 2004.

[11] J. Fan and S. Kambhampati. "A Snapshot of Public Web Services". *SIGMOD Record*, 34(1):24–32, 2005.

[12] C. Fellbaum (Eds). *"WordNet: An Electronic Lexical Database"*. The MIT Press, 1998.

[13] A. Maedche and S. Staab. "Measuring Similarity between Ontologies". In *European Conf. on Knowledge Acquisition and Management (EKAW)*, Siguenza, Spain, 2002.

[14] D. Martin (Ed.). "OWL-S 1.1 Release", 2005. http://www.daml.org/services/owl-s/1.1/.

[15] B. Medjahed and A. Bouguettaya. "A Multilevel Composability Model for Semantic Web Services". *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 17(7):954–968, 2005.

[16] J. Miller et al. "WSDL-S: Adding Semantics to WSDL - White Paper", 2004. http://lsdis.cs.uga.edu/library/download/wsdl-s.pdf.

[17] N. Mitra (Ed.). "SOAP Version 1.2 Part 0: Primer". W3C Recommendation, 2003. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.

[18] M. E. J. Newman. "Power laws, Pareto distributions and Zipf's law". *Contemporary Physics*, 46:323–351, 2005.

[19] S.-C. Oh, D. Lee, and S. Kumara. "A Comparative Illustration of AI Planning-based Web Services Composition". *ACM SIGecom Exchanges*, 5(5):1–10, Dec. 2005.

[20] E. Rahm and P. Bernstein. "A Survey of Approaches to Automatic Schema Matching". *VLDB J.*, 10(4):334–350, 2001.

[21] G. Salton and M.J. McGill. *"Introduction to Modern Information Retrieval"*. McGraw–Hill, 1983.

[22] Y. Wang and E. Stroulia. "Semantic Structure Matching for Assessing Web Service Similarity". In *Int'l Conf. on Service Oriented Computing*, Trento, Italy, 2003.

[23] D. J. Watts and S. H. Strogatz. "Collective Dynamics of 'Small-World' Networks". *Nature*, 393(4):440–442, 1998.

[24] J. Wu and Z. Wu. "Similarity-based Web Service Matchmaking". In *IEEE Int'l Conf. on Service Computing (SCC)*, Orlando, FL, USA, 2005.

[25] K. Zhang and D. Shasha. "Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems". *SIAM J. Comput.*, 18(6):1245–1262, Dec. 1989.

SPARQL-based OWL-S Service Matchmaking

Said Mirza Pahlevi, Akiyoshi Matono, and Isao Kojima

National Institute of Advanced Industrial Science and Technology (AIST)
Grid Technology Research Center, Tsukuba, Ibaraki 305-8568, Japan

Abstract. The increasing availability of Web services has made the accurate and efficient discovery and selection of target services an important issue. OWL-S is a language that semantically describes Web services to facilitate automated service discovery and selection. This paper proposes an OWL-S service matchmaking mechanism that utilizes the emerging standard SPARQL. In this mechanism, a service requestor queries service repositories via a matchmaker by using the SPARQL query language. The matchmaker performs service matching and ordering with the help of a SPARQL query engine. Using SPARQL as the query language offers several advantages that are not provided by existing matchmaking systems, allowing complex service matching and reducing the difficulty of query formulation. In addition, it makes the architecture of the matchmaking system loosely-coupled and that of the matchmaker lightweight.

1 Introduction

One of the primary goals of semantic Web services is to enable automatic discovery and selection of the services. Automatic discovery is an automated process for locating a Web service that provides a particular class of service capabilities while adhering to client-specific constraints. To discover a service, a software agent needs a computer-interpretable description of the service and a means by which to access it. OWL-S [1], WSMO [2] and WSDL-S [3] are examples of languages and models that semantically describe Web services.

In this paper, we focus on using OWL-S for describing Web services because we adopt a matching method used by many service matchmakers that is based on the OWL-S service description. OWL-S describes the Web services in terms of the capabilities offered based on the Web Ontology Language (OWL) [4].

In a typical scenario, a service provider describes advertised services using an OWL-S compliant ontology and submits the advertisements to a Web service repository. A service requester queries the repository by creating a service request using the ontology. The repository matches registered advertisements to the request by performing inferences on the concept hierarchy and orders the results based on the degree of match between the request and advertisements. Since queries are formulated using an ontology rather than a high-level query language, queries are quite difficult to compose and the repository usually only supports simple queries. Furthermore, the system needs to implement a proprietary query engine because each system uses a different matching method.

The recently approved standard WS-Resource Framework (WSRF) [5] defines conventions related to managing Web services, which improves several aspects of these services to make them more adequate for grid applications. It defines a *WS-Resource* that describes the relationship between a service and a resource. In this environment, resources are discovered by identifying the Web services that interact with them. Resource discovery in the grid is currently based on text matching [6]. Text matching-based resource discovery, however, is not a feasible solution for grids having a large number of resources with different capabilities distributed across different organizations.

This paper proposes a new OWL-S service matchmaking mechanism based on the SPARQL query language [7] that is able to support semantic resource matching within a grid. We envision service repositories with a SPARQL query processing capability[1] and a matchmaker that serves requester queries. A service provider uses the OWL-S ontology to describe a service, but a service requester formulates a query using SPARQL sent to a matchmaker.

On receiving a requester query, the matchmaker forwards it to a repository and receives query results. The matchmaker needs to order results because either they are inherently unordered or they are ordered in a way that is not based on class subsumption relationships. To this end, the matchmaker rewrites the requester query before sending it to a repository so that query results will contain all information necessary for result ordering. This paper proposes two mechanisms: *query rewriting* and *result ordering* that enable the matchmaker to order query results based on class subsumption relationships. The matchmaker also allows a requester to set an ordering preference based on service characteristics.

Using SPARQL to formulate a service request provides the following advantages.

- *Imposing no restrictions on the repository system.* Since SPARQL is a standard query language, the repository does not need to identify whether a query comes from a matchmaker. Furthermore, the repository can use existing SPARQL query engines [8].
- *Loosely-coupled architecture.* A matchmaker can easily switch from one repository to another or send a simultaneous query to multiple repositories.
- *Lightweight matchmaker.* The matchmaker must only perform lightweight tasks such as query rewriting and result ordering, while the heavyweight task of service matching is provided by a "standard" SPARQL query engine in a repository system.
- *Complex service matching.* A requester can use the rich SPARQL constructs (e.g., optional, filtering, and union) during service requests. Furthermore, SPARQL inherently supports some sorts of grid resource matching.
- Ability to use standard *SPARQL query results in XML format* [9] and *protocols* [10] in communications between a requester and the matchmaker and between the matchmaker and the repository.

[1] The SPARQL query engine could be attached to the matchmaker or a stand-alone module. However, we adopted this approach because we envision a future in which many RDF repositories are located behind a SPARQL query engine.

The rest of the paper is organized as follows. Section 2 briefly describes the semantic markup for Web services, OWL-S, and the SPARQL query language. Section 3 reviews related work. Section 4 describes the proposed matchmaking mechanism. Section 5 describes semantic resource matching in the Grid. The final section discusses advanced use of SPARQL for service and resource matching, and outlines future activities.

2 Background

2.1 OWL-S

OWL-S [1] is an ontology of service concepts for describing the properties and capabilities of web services in a machine interpretable form. It consists of three main parts/classes: the `ServiceProfile`, `ProcessModel`, and `ServiceGrounding`. The `ServiceProfile` is used for advertising and discovering services. It includes three basic types of information: what organization provides the service (e.g., contact information), what function the service computes (e.g., inputs and outputs) and what characteristics the service has (e.g., service category). The `ProcessModel` gives a detailed description of a service's operation and the `ServiceGrounding` specifies the details of how to access the service.

A class `Service` simply binds the three parts together into a unit via three object properties: `presents` (to the `ServiceProfile`), `describedby` (to the `ProcessModel`) and `supports` (to the `ServiceGrounding`).

Since `ServiceProfile` is used for service discovery, this paper mainly deals with the class or classes that extend the `ServiceProfile`.

2.2 SPARQL Query Language

SPARQL consists of a query language [7], a means of conveying a query to a query processor [10], and the XML format in which query results will be returned [9]. The specification is currently under discussion as a W3C Working Draft but the availability of several SPARQL query engines [8] means that the specification is getting stable for the practical use.

The SPARQL query language is based on matching graph patterns. The simplest pattern is the *triple pattern* consisting of subject, predicate, and object. Any or all of subject, predicate, and object values may be replaced by a variable prefixed with either "?" or "$". Combining triple gives a *basic graph pattern*, where an exact match to a graph is needed to fulfill a pattern.

The query below finds a service (bound to variable $?s$) that has a `serviceProfile` (bound to $?o$) whose types/classes of its two `pr:hasInput` values (bound to $?in_1$ and $?in_2$) are `:Input1` and `:Input2`, respectively.

```
PREFIX sr: <http://www.daml.org/services/owl-s/1.1/Service.owl#>
PREFIX pr: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
BASE <http://www.owl-ontologies.com/unnamed.owl#>
```

```
SELECT ?s WHERE ?s sr:presents ?o.
?o pr:hasInput ?in1. ?in1 rdf:type :Input1.
?o pr:hasInput ?in2. ?in2 rdf:type :Input2.
```

The `PREFIX` keyword associates a prefix label with a URI and `BASE` keyword defines the based URI to resolve relative URIs. The `SELECT` clause indentifies the variables to appear in the query results and the `WHERE` clause contains a graph pattern.

SPARQL provides other graph patterns for complex matching such as *optional graph pattern* for matching semi-structured RDF graphs, *alternative graph pattern* for combining graph patterns so that one of several alternative graph patterns may match, and *patterns on named graphs* where patterns are matched against named graphs. It also provides value constraints which restrict RDF terms in a solution. SPARQL includes a number of syntax shortcuts that simplify the writing of patterns. For example, triple patterns sharing the same subject and predicate can be written using the ",“ notation (e.g., triples with predicate `pr:hasInput` in the query above can be written as $?o$ `pr:hasInput` $?in_1$, $?in_2$.) and `rdf:type` can be replaced with keyword "a“.

3 Related Work

3.1 OWL-S Semantic Matchmaking

Over the last few years, researchers have proposed several semantic matching methods. [11] is the first method that provides an algorithm for matching service advertisements and requests based on service inputs and outputs. An advertisement matches a request when *all request outputs* are matched by advertisement outputs, and *all advertisement inputs* are matched by request inputs. This characteristic guarantees that the matched service provides all outputs requested by the requester, and that the requester provides all input required for correct operation to the matched service.

More specifically, an advertisement output outA exactly matches a request output outR when it is identical with outR or when outR is an immediate subclass of outA. If outA subsumes outR then this is considered a Plug-in match; conversely, if outR subsumes outA, then this is considered a Subsume match. If no subsumption relationship appears between outA and outR, then this is considered a Fail match. Matching of inputs involves a similar computation method but reverses the order of request and advertisement.

The following matching methods are based on [11]. [12] adds a new similarity metrics intersection to detect when an advertisement satisfies only some features of a request, whereas [13] further examines relationships among classes and their property classes. [14] and [15] add syntactic similarity and [16] adds service category matching and user-defined matching to semantic matching. [17] proposes a rather different matching approach. It computes the best combinations of Web services to provide the most satisfactory request outputs while requiring the least possible inputs that are not provided in the request.

All matchmaking systems, however, use ontology or description logic in service requests instead of a high-level query language such as SPARQL. Furthermore, because the systems have different matching methods, they use proprietary matching engines.

3.2 Resource Discovery in the Grid

The grid middleware, Globus Toolkit (GT) 4.0 [18], provides a service, called the Index Service [6], that facilitates grid resource monitoring and discovery. This service collects data from various sources and publishes them as XML documents. This service provides structured (XML) data but without semantic constraints. S-MDS [19] is a framework to support automatic service discovery and monitoring in the grid. It describes the metadata of WS-Resources using OWL-S ontology and provides an efficient mechanism to aggregate and maintain the ontology instances. S-MDS, however, is a kind of an index service rather than a matchmaking system. Our matchmaking system can work with S-MDS to provide semantic resource discovery and selection in the grid.

Classad [20] is a matchmaking framework to resource management in distributed environment with decentralized ownership of resources. In this framework, a service advertisement and request are formulated using a semi-structured data model called classified advertisements (classad) which consists of attribute-value pairs. A matchmaker matches the advertisement and the request based on constraints specified by attributes in the classads. This framework, however, only allows a *bilateral match*, that is, matching a single request with a single resource. [21] extends the work so that a single request can be matched with multiple (types of) resources (*gang match*).

Redline [22] is a grid matchmaking system that reinterprets matching as a constraint problem and exploits constraint-solving technologies to implement matching operations. It provides two new matching methods which are not supported by classad: *set match* and *congruous match*. The former matches between a request and a resource set with particular aggregated properties whereas the latter matches between multiple requests and a resource.

Ontology-based Matchmaker (OMM) [23] is an ontology-based resource selector for solving resource matching in the grid. Resources and requests are described by (different) ontologies and they are matched using matching rules. The matchmaker can be easily extended by adding vocabularies and inference rules to include new concepts about resources. Like classadd, OMM supports bilateral match and gang match.

Classad and Redline, however, are based on text matching rather than semantic matching. Furthermore, they do not use ontology for service advertisement and request. On the other hand, OMM supports semantic matching for resource discovery. It, however, uses proprietary ontology to describe resources. Similar to the OWL-S matchmakers, a resource request is formulated using ontology and a special matching engine is needed to process the request.

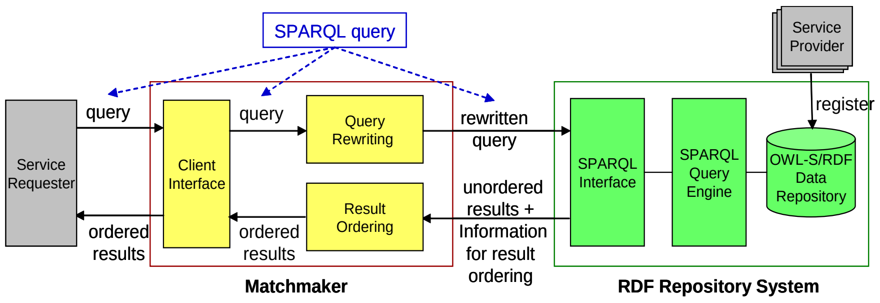


Fig. 1. System architecture

4 SPARQL-based OWL-S Service Matchmaking

Fig. 1 presents the basic architecture of the proposed matchmaking system. A requester sends a SPARQL query as a service request to a matchmaker through a client interface. The matchmaker rewrites the query and sends the rewritten query to a repository system. The query is rewritten such that matching results returned by the repository will contain the information necessary for result ordering. Matching results are ordered based on the degree of match between the query and its resulting services and the ordered results are returned to the requester.

A SPARQL query treats an RDF graph purely as data, i.e. it does not interpret RDF schema information or entailments. The query, however, can be issued to an inferred graph, thus allowing query over the entailments. In this paper, we assume that in addition to storing OWL-S service instances with their schema, an RDF repository provides an inferred graph of the instances. The former is called *uninferred Graph* (*uGraph*), whereas the latter is called *inferred Graph* (*iGraph*)[2]. *iGraph* is a query's default graph, and *uGraph* is accessed through the SPARQL named graph mechanism.

The rest of this paper will apply the following definitions. A graph pattern in the definitions is matched against *iGraph*. For simplicity, SPARQL prefix keywords and labels are always omitted from a query and graph pattern and syntax shortcuts are always used.

Definition 1. Type-matching query graph pattern (type-matching pattern) pt = $\{?o\ p\ ?var_1\ ,\ \ldots,\ ?var_k\ .\ ?var_1\ \ a\ c_1\ .\ \ \ldots\ ?var_k\ \ a\ c_k\ .\}$ *is defined as a graph pattern that matches* k *values of predicate* p *while restricting values to those from types/classes* $c_1, \ldots, c_k$. p *is called the* target property/predicate *and* $c_1, \ldots, c_k$ *is called the* target property value (tpv) classes[3].

Definition 2. Relative classes *of class* c *include all its superclasses and subclasses (including* c*).*

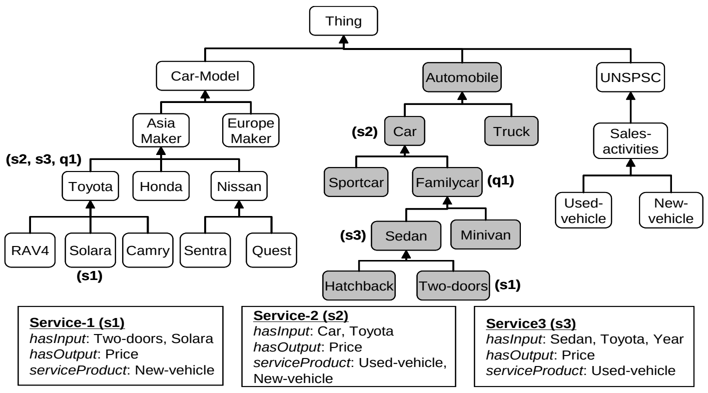
[2] When a resource is defined as an instance from class c in *uGraph*, it is inferred as instances from c's subclasses in *iGraph*.

[3] Henceforth, we will use *type* and *class* and *property* and *predicate* interchangeably.

Definition 3. *Given type-matching pattern pt: Instance s is said to be a* sub-class instance *of target property p if s matches pt. In this case, values of p (of s) val_1, ..., val_k (which are bound to var_1, ..., var_k of pt) are from a subclass of (tpv classes) c_1, ..., c_k, respectively.*

Definition 4. *Given type-matching pattern pt: Instance s is said to be a* super-class instance *of target property p if s does not match pt and values of p (of s) val_1, ..., val_k are from a relative class of (tpv classes) c_1, ..., c_k, respectively. In this case, one (or more) of the relative classes is a superclass of a tpv class (excluding the tpv class itself).*

Definition 5. *Given type-matching pattern pt: Property p of instance s is said to be* an inclusive property *(with respect to pt) if s is a subclass or superclass instance of target property p and p has (exacly) k values. Otherwise p is said to be* a non-inclusive property.



Service-1 (s1)
hasInput: Two-doors, Solara
hasOutput: Price
serviceProduct: New-vehicle

Service-2 (s2)
hasInput: Car, Toyota
hasOutput: Price
serviceProduct: Used-vehicle, New-vehicle

Service3 (s3)
hasInput: Sedan, Toyota, Year
hasOutput: Price
serviceProduct: Used-vehicle

Fig. 2. A fragment of ontologies and service instances

Example 1 Fig. 2 presents a fragment of some domain-specific ontologies and service instances. The figure also presents properties and their value types of the instances. For example, "`hasInput=Two-doors, Solara`" of s_1 denotes that s_1 has a property `hasInput` with two values whose types are `Two-doors` and `Solara`, respectively.

q_1 = `SELECT` $?s$ `WHERE` $?s$ `presents` $?o$.
$?o$ `hasInput` $?in_1$, $?in_2$. $?in_1$ `a Familycar.` $?in_2$ `a Toyota.`

Query q_1 above contains type-matching pattern $\{?o$ `hasInput` $?in_1$, $?in_2$. $?in_1$ `a Familycar.` $?in_2$ `a Toyota.`$\}$. s_1 is a subclass instance of `hasInput` because it matches the query. Since it has two `hasInput` values, the `hasInput` is an inclusive property. Similarly, s_3 is a subclass instance of `hasInput` because it matches the query. Its `hasInput` property, however, is a non-inclusive one because the property has three values of which one type is `Year`. However, s_2 is a superclass instance of `hasInput` because it does not match q_1 and its `hasInput` value types are `Car` and `Toyota`, which are from the relative classes of `Familycar` and `Toyota`, respectively. The `hasInput` property is an inclusive one.

4.1 Query Rewriting

Upon receiving query results from a repository, the matchmaker orders them based on *the degree of match* between concepts included in the query and in matched services. This paper uses the degree of match defined in [11] (see Section 3.1). The results, however, do not contain the data or information necessary for the ordering task. For example, applying q_1 to *iGraph* (of ontologies shown in Fig 2) returns s_1 and s_3, but the results lack the following information:

– *Superclass instances of property* `hasInput`. q_1 leaves s_2 as an unmatched service because s_2 is a superclass instance of `hasInput`. Lack of this data will prevent application of the Subsume match calculation.
– *Immediate subclass/superclass information.* For example, `Two-doors` (which is the `hasInput` value type of s_1) is not an immediate subclass of `Familycar`, unlike `Sedan` (which is the `hasInput` value type of s_3). Lack of this information will prevent application of Exact and Plug-in match calculations.
– *Property inclusiveness information.* For example, `hasInput` of s_1 is an inclusive property, whereas that of s_3 is a non-inclusive one. Lack of this information will prevent application of the Fail match calculation.

This paper examines the following requester query[4].

SELECT $?s$
WHERE otherGP. $?s$ presents $?o$.
$?o\ p_1\ ?var_{1,1},\ \ldots,\ ?var_{1,k_1}.\ ?var_{1,1}$ a $c_{1,1}.\ \ldots\ ?var_{1,k_1}$ a $c_{1,k_1}.$
...
$?o\ p_m\ ?var_{m,1},\ \ldots,\ ?var_{m,k_m}.\ ?var_{m,1}$ a $c_{m,1}.\ \ldots\ ?var_{m,k_m}$ a $c_{m,k_m}.$

The query contains a set of type-matching patterns and other graph patterns (`otherGP`). `otherGP` can be optional, alternative, and/or value constraint graph patterns [7].

The query rewriting procedure involves rewriting each type-matching pattern such that pattern solutions will contain superclass and subclass instances of the target property, class subsumption information, and target property inclusiveness information. More formally, given the following query, where $1 \leq i \leq m$,

SELECT $?s$
WHERE otherGP. $?s$ presents $?o$.
$?o\ p_i\ ?var_{i,1},\ \ldots,\ ?var_{i,k_i}.\ ?var_{i,1}$ a $c_{i,1}.\ \ldots\ ?var_{i,k_i}$ a $c_{i,k_i}.$

```
1:  SELECT distinct $?s$ $?otype_i$ $?sup_{i,1}$ ... $?sup_{i,k_i}$ $?sub_{i,1}$ ...
    $?sub_{i,k_i}$ $?isup_{i,1}$ ... $?isup_{i,k_i}$ $?isub_{i,1}$ ... $?isub_{i,k_i}$
2:  WHERE otherGP. $?s$ presents ?o.
3:    {$c_{i,1}$ rs $?sup_{i,1}$. FILTER($?sup_{i,1} \neq c_{i,1}$)}UNION{$?sub_{i,1}$ rs $c_{i,1}$} ...
4:    {$c_{i,k_i}$ rs $?sup_{i,k_i}$. FILTER($?sup_{i,k_i} \neq c_{i,k_i}$)}UNION{$?sub_{i,k_i}$ rs $c_{i,k_i}$}
5:    GRAPH uGraph {
6:      ?o $p_i$ $?var_{i,1}$. $?var_{i,1}$ a $?type_{i,1}$.
7:        FILTER($?type_{i,1}$ =$?sup_{i,1}$||$?type_{i,1}$ =$?sub_{i,1}$) ...
8:      ?o $p_i$ $?var_{i,k_i}$. $?var_{i,k_i}$ a $?type_{i,k_i}$.
9:        FILTER($?type_{i,k_i}$ =$?sup_{i,k_i}$||$?type_{i,k_i}$ =$?sub_{i,k_i}$)
10:     OPTIONAL{?o $p_i$ $?ovar_i$. $?ovar_i$ a $?otype_i$.
11:       FILTER($?otype_i \neq ?type_{i,1}$ && ... && $?otype_i \neq ?type_{i,k_i}$)}
12:     OPTIONAL{$?isub_{i,1}$ rs $c_{i,1}$. FILTER($?isub_{i,1}$ =$?type_{i,1}$)}
13:     OPTIONAL{$c_{i,1}$ rs $?isup_{i,1}$. FILTER($?isup_{i,1}$ =$?type_{i,1}$)}
14:     ...
15:     OPTIONAL{$?isub_{i,k_i}$ rs $c_{i,k_i}$. FILTER($?isub_{i,k_i}$ =$?type_{i,k_i}$)}
16:     OPTIONAL{$c_{i,k_i}$ rs $?isup_{i,k_i}$. FILTER($?isup_{i,k_i}$ =$?type_{i,k_i}$)}
17:   }
```

Fig. 3. Query rewriting procedure

The query is rewritten as shown in Fig 3.

The upper part (lines 2–4) is matched against the default *iGraph* while the lower part (lines 6–16) is matched against *uGraph*. Lines 3 and 4 of the upper part extract the relative classes of tpv classes $c_{i,1}, \ldots, c_{i,k_i}$ by using the **rdfs:subClassOf** property (abbreviated as *rs*). Value constraints (**FILTER**) are used to reduce the result size because a class is a subclass and superclass of itself. The lower part can be further divided into three subparts:

1. The first part (lines 6–9) matches instances in which target property value types are (relative) classes matched by the upper part. $?var_{i,j}$ is bound to a target property value and $?type_{i,j}$ is bound to an *uninferred class* (i.e., a class defined in *uGraph*) of the value ($1 \leq j \leq k$). This part corresponds to the type-matching patterns in the original query.
2. The second part (lines 10 and 11) retrieves property inclusiveness information. The optional graph pattern verifies the existence of other p_i values of types other than relative classes. This is done by defining a new variable $?ovar_i$ for the value and excluding relative classes during type matching for the value. Information about p_i property inclusiveness can easily be obtained from the pattern solution by examining the binding value of $?otype_i$. If this is bound to some value, then p_i is a non-inclusive property; otherwise it is an inclusive one (see Example 2 below).

[4] For simplicity, the path length from {$?s$ presents $?o$} to type-matching patterns is set here at zero, but the path could be set at any arbitrary length.

3. The third part (lines 12–16) retrieves immediate subclass and superclass information. This part uses the `rdfs:subClassOf` property combined with an optional graph pattern to bind the immediate subclass and superclass of tpv class $c_{i,j}$ to variable $?isub_{i,j}$ and $?isup_{i,j}$, respectively $(1 \leq j \leq k_i)$.

Let s_o and s_r be a set of services contained in the solutions to a requester query and the corresponding rewritten query, respectively. Then, $s_o \subseteq s_r$ and $s_r - s_o = s_{sup}$, where s_{sup} is a set of superclass instances of the target properties. The proof is that the lower part of the rewritten query does not add any restrictions to the services bound to variable $?s$, except that the type of p_i value (which is bound to $?var_{i,j}$) *must* be $c_{i,j}$ (or its subclass) or a superclass of $c_{i,j}$. The former restriction is identical to that in the original query while the latter *relaxes* the query to include superclass instances. Note that the optional graph patterns in the lower part do not give any matching restrictions while `otherGP` is matched against the default graph as in the original query.

Example 2 q_1 shown above is rewritten as follow and the binding solutions are shown in Table 1.

```
SELECT distinct $?s$ $?otype_1$ $?sup_1$ $?sup_2$ $?sub_1$ $?sub_2$ $?isup_1$ $?isup_2$
$?isub_1$ $?isub_2$
WHERE $?s$ presents $?o$.
  {Familycar rs $?sup_1$. FILTER($?sup_1 \neq$ Familycar)}
   UNION{$?sub_1$ rs Familycar}
  {Toyota rs $?sup_2$. FILTER($?sup_2 \neq$ Toyota)}
   UNION{$?sub_2$ rs Toyota}
  GRAPH uGraph {
   $?o$ hasInput $?var_1$. $?var_1$ a $?type_1$.
     FILTER($?type_1 =?sup_1 || ?type_1 =?sub_1$).
   $?o$ hasInput $?var_2$. $var_2$ a $?type_2$.
     FILTER($?type_2 =?sup_2 || ?type_2 =?sub_2$).
   OPTIONAL{$?o$ hasInput $?ovar_1$. $?ovar_1$ a $?otype_1$.
     FILTER($?otype_1 \neq ?type_1$ && $?otype_1 \neq ?type_2$)}
   OPTIONAL{$?isub_1$ rs Familycar. FILTER($?isub_1 =?type_1$)}
   OPTIONAL{Familycar rs $?isup_1$. FILTER($?isup_1 =?type_1$)}
   OPTIONAL{$?isub_2$ rs Toyota. FILTER($?isub_2 =?type_2$)}
   OPTIONAL{Toyota rs $?isup_2$. FILTER($?isup_2 =?type_2$)}
  }
```

Table 1. Binding solutions

	$?s$	$?otype_1$	$?sup_1$	$?sup_2$	$?sub_1$	$?sub_2$	$?isup_1$	$?isup_2$	$?isub_1$	$?isub_2$
b_1:	s_1				Two-doors	Solara				Solara
b_2:	s_2		Car			Toyota	Car			
b_3:	s_3	Year			Sedan	Toyota			Sedan	

There are three binding solutions b_1, b_2, and b_3 which bind values in instances s_1, s_2, and s_3, respectively, to some variables. From b_1, the `hasInput` of s_1 is an inclusive property because $?otype_1$ is unbound. $?sub_1$ binding value indicates that the first `hasInput` value type of s_1 (i.e., `Two-doors`) is a subclass of `Familycar` while $?sub_2$ and $?isub_2$ values indicate that the second value type of s_1 (i.e., `Solara`) is an immediate subclass of `Toyota`. Similarly, from b_2, the `hasInput` of s_2 is an inclusive property, and the first `hasInput` value type (i.e., `Car`) is an immediate superclass of `Familycar` while the second value type is equal to `Toyota`. The last solution b_3 indicates that the `hasInput` of s_3 is a non-inclusive property because $?otype_1$ is bound to some value (i.e., `Year`). The first `hasInput` value type of s_3 is an immediate subclass of `Familycar` while the second value type is equal to `Toyota`.

It is important to note that subclass and superclass variables are only bound to classes that are "actually used“ in service instances. For example, $?sup_2$ is unbound because no `hasInput` value types of matched services are superclasses of `Toyota`. This will greatly reduce the size of return results [5]

4.2 Service Ordering

Generally, a requester query consists of one *service function matching* (based on service inputs and outputs) and optionally one or more *service characteristics matching* (based on service characterization according to some domain-specific ontologies). The next sections describe the scoring mechanisms used in matching.

Service Function Scoring Given requester query q containing a service function type-matching pattern

$\{?o$ `hasInput` $?in_1, \ldots, ?in_{k_{in}}$. $?in_1$ `a` cin_1 . ... $?in_{k_{in}}$ `a` $cin_{k_{in}}$.
$?o$ `hasOutput` $?out_1, \ldots, ?out_{k_{out}}$. $?out_1$ `a` $cout_1$ . ... $?out_{k_{out}}$ `a` $cout_{k_{out}}$.$\}$

and binding solution b from matching results of q's rewritten query: The degree of match between q and b with respect to target property `hasInput` is given by Eq. 1.

$$d_{in}(q, b) = \text{DIN}(cin_1, b.c_1) + \ldots + \text{DIN}(cin_{k_{in}}, b.c_{k_{in}}) \qquad (1)$$

$\text{DIN}(cin_i, b.c_i)$ is the degree of match between class cin_i and the corresponding class c_i which is bound to subclass/superclass variables in b. The degree of match could be Exact, Plug-in, or Subsume, where Exact>Plug-in>Subsume. The scoring formula indicates that the larger $d_{in}(q, b)$ the more similar inputs of the matched service (which is bound to variable $?s$ in b) to inputs specified in q.

Fig. 4 shows computation of DIN which is similar to [11], but without the Fail match. b is considered to be a Fail match when the `hasInput` (of a service bound to variable $?s$ in b) is a non-inclusive property. This results from the fact

[5] This is one reason we use the *uGraph*. Another reason is to obtain the uninferred class and immediate subclass information of target property values, which is impossible to do when matching the pattern against *iGraph*.

that the requester is not able to provide the bound service in b all the input required for correct operation.

The degree of match for b with respect to target property **hasOutput** is given by Eq. 2. It uses DOUT function shown in Fig. 5. Different from DIN, DOUT uses subclass relationships to determine Exact and Plug-in matches.

$$d_{out}(q, b) = \text{DOUT}(cout_1, b.c_1) + \ldots + \text{DOUT}(cout_{k_{out}}, b.c_{k_{out}}) \tag{2}$$

```
DIN(inR, inA)
 if inR=inA return Exact
 if inR immediateSuperclassOf inA
   return Exact
 if inR superclassOf inA
   return Plug-in
 otherwise return Subsume
```

Fig. 4. DIN calculation

```
DOUT(outR, outA)
 if outR=outA return Exact
 if outR immediateSubclassOf outA
   return Exact
 if outR subclassOf outA
   return Plug-in
 otherwise return Subsume
```

Fig. 5. DOUT calculation

Finally, summing up the degree of match of inputs and outputs and normalizing the value between 0 and 1 gives the degree of match with respect to the service function (Eq. 3).

$$d_f(q, b) = \frac{k_{out} \times d_{in}(q, b) + k_{in} \times d_{out}(q, b)}{2 \times Exact \times k_{in} \times k_{out}} \tag{3}$$

Note that when $d_{in}(q, b)$ fails, $d_f(q, b)$ also fails.

Example 3 Let Exact=3, Plug-in=2, and Subsume=1. The rewritten query of q_2 below has three binding solutions b_1, b_2, and b_3 that bind s_1, s_2, and s_3, respectively. Matchmaker assigns $d_f(q_2, b_1) > d_f(q_2, b_2)$ and $d_f(q_2, b_3)$ =Fail.

```
q2 = SELECT ?s WHERE ?s presents ?o,
?o hasInput ?in1, in2, ?in1 a Familycar. in2 a Toyota,
?o hasOutput ?out. ?out a Price.
```

$d_{in}(q_2, b_1) = \text{DIN}(\text{Familycar, Two-doors}) + \text{DIN}(\text{Toyota, Solara}) = 5$,
$d_{in}(q_2, b_2) = \text{DIN}(\text{Familycar, Car}) + \text{DIN}(\text{Toyota, Toyota}) = 4$,
$d_{out}(q_2, b_1) = d_{out}(q_2, b_2) = \text{DOUT}(\text{Price, Price}) = 3$,
$d_f(q_2, b_1) = 0.92$, and $d_f(q_2, b_2) = 0.83$.

Service Characteristics Scoring Service characteristics describe the categories, classifications, and other features owned by a service. A domain-specific ontology may be used to describe these characteristics. For example, the OWL ontologies of NAICS [24] and UNSPSC [25] can be used to describe service classification and products, respectively. In many cases, it is desirable to let a service requester specify a service-ordering preference based on domain-specific ontology matching. For example, during the car selling service discovery, a requester may give a higher ordering preference to services that sell only new cars or that sell only Toyota cars. In a grid environment, a requester may give a higher ordering preference to grid resources that are managed only by a specific organization.

Given requester query q containing a service characteristic type-matching pattern, where target property p_i specifies service characteristics,

$\{?o\ p_i\ ?var_1,\ \ldots,\ ?var_k.\ ?var_1\ \texttt{a}\ cs_1.\ \ldots\ ?var_k\ \texttt{a}\ cs_k.\}$

and binding solution b from matching results of q's rewritten query: The degree of match between q and b with respect to p_i is given by Eq. 4.

$$d_{p_i}(q,b) = \begin{cases} 1 + \frac{\mathrm{DOUT}(cs_1, b.c_1) + \ldots + \mathrm{DOUT}(cs_k, b.c_k)}{Exact \times k} & b \text{ is given a higher preference} \\ \frac{\mathrm{DOUT}(cs_1, b.c_1) + \ldots + \mathrm{DOUT}(cs_k, b.c_k)}{Exact \times k} & \text{otherwise} \end{cases} \tag{4}$$

The value of the dividend that sums up the degree of match of the service characteristic is normalized between 0 and 1 by dividing it with $Exact \times k$. DOUT is used for degree of match calculation because service characteristics are a kind of requested output. By default, b is given a higher ordering preference if the p_i (of the bound service) is an inclusive property[6]. To specify a higher ordering preference to the non-inclusive property, p_i' `ORDER BY (!`p_i'`)` is used[7].

Finally, the total degree of match between q and b with respect to service characteristics specified by target properties $p_1, \ldots, p_m$ is given by Eq. 5. The divisor $2 \times m$ normalizes the match degree between 0 and 1.

$$d_c(q,b) = \frac{\sum_{i=1}^{m} d_{p_i}(q,b)}{2 \times m} \tag{5}$$

Example 4 The rewritten query of q_3 below has two binding solutions $b_1$and b_2 that bind s_1 and s_2, respectively. The matchmaker gives a higher ordering preference to b_1 because s_1 sells only new vehicles (i.e., the `serviceProduct` property of s_1 is inclusive). Putting `ORDER BY (!serviceProduct)` in the query will reverse the ordering preference.

```
q3 = SELECT ?s WHERE ?s presents ?o,
?o hasInput ?in1, in2, ?in1 a Familycar. in2 a Toyota,
?o hasOutput ?out. ?out a Price.
?o serviceProduct ?d. ?d a New-vehicle.
```

[6] `ORDER BY` (p_i) can be used to explicitly specify the ordering preference.

[7] Since this construct is only used for the purposes of result ordering, the matchmaker removes it before sending the rewritten query to a repository system.

Overall Scoring and Service Ordering Given requester query q with target properties $p_1, \ldots, p_k$ specifying service characteristics and a set of binding solutions bs from matching results of q's rewritten query. For each binding solution $b \in bs$, the matchmaker calculates the overall degree of match between q and b, $d(q,b)$ $(0 \leq d(q,b) \leq 1)$ using Eq. 6 shown below and orders the solutions in descending order of their degrees of match.

$$d(q,b) = \alpha \times d_f(q,b) + (1-\alpha) \times d_c(q,b), \text{where } 0 \leq \alpha \leq 1 \qquad (6)$$

α is a weight that specifies the portion of $d_f(q,b)$ and $d_c(q,b)$ that includes in the total scoring. By appropriately setting the weight value, the ordering scheme not only can meet a user specific ordering requirement, but also can be used for other ordering purposes. For example, by setting $\alpha = 0$ the ordering scheme can be used to order general SPARQL query results. It is important to note that $d(q,b)$ fails when $d_f(q,b)$ fails.

5 Semantic Resource Matching in the Grid

As described earlier, resources in the Grid are accessed via Web services and resource discovery is done by identifying Web services that provide access to the resources. In this sense, if the Web services are described using the OWL-S language, such as proposed in [19], our scheme can be applied for basic resource matching in the Grid, i.e., bilateral match.

In addition, use of SPARQL as the request language enables congruous and gang matches, which are important during grid resource discovery. These matches are made possible because SPARQL allows several graph patterns to be used in a query and allows them to refer to the same common variables. The following example from [21] demonstrates a gang match using SPARQL. The purpose is to match a job (resource request) with two types of resources: workstations and software package licenses. A job that uses the packages needs to allocate both a machine and a license before it can run. Licensing terms may entail that some licenses are valid only on some machines, while others may be valid in certain subnets. Assuming the two resources are described by different RDF graphs and the workstation RDF graph is the default graph, the following query performs a gang match against the two resources. Note that the two graph patterns refer to the same variable *?address*.

```
SELECT ?machine ?license
WHERE ?machine hasArchitecture ?arch. ?arch a Pentium.
   ?machine hasMemory ?mem. FILTER(?mem > 1000).
   ?machine hasAddress ?address. FILTER(?address =”foo.go.jp“).
   GRAPH PackageLicense{
     ?license hasValidHost ?address.
     ?license hasApplication ?app. FILTER(?app =”sim_app“).
   }
```

It is important to note that bilateral, congruous, and gang matches using our matchmaking scheme are done based on the OWL-S service description that allows semantic matching. This is different from the conventional grid resource discovery systems [20, 21, 6, 22] that use text matching to perform the matching tasks.

6 Discussion and Future Work

The optional graph pattern is one of the interesting features provided by the SPARQL query language. It defines additional graph patterns that do not cause solutions to be rejected if the solutions cannot be matched, but do bind the solutions when they can be matched. This feature is not supported by existing matchmaker systems, but can be used by a service requester for complex input/output and service characteristics matching. For example, it can match a service that provides additional outputs or characteristics to the required ones.

The other interesting feature provided by the SPARQL query language is its value constraints. It restricts solutions by imposing constraints on values that can be bound to variables. This feature is useful, for example, to filter services based on the `serviceName` and `textDescription` of the `serviceProfile` using regular expressions. The language also allows application-specific constraints on values in a solution.

The other feature, alternative graph patterns, can be useful when matching several graph patterns. A pattern can be nested and used together with other graph patterns such as optional ones. In this paper, we do not allow these patterns to be used with type-matching patterns (e.g., to connect two type-matching patterns). A requester should be able to send multiple queries and obtain the same results.

The above graph patterns and value constraints do not affect service ordering. Optional graph patterns may contain type-matching patterns but an ordering preference cannot be assigned to the patterns[8].

We have implemented the matchmaker using Java and use Jena SPARQL query parser [26] for query rewriting and result ordering. The matchmaker also incorporates Jena SPARQL query and inference engines so that it can process RDF data stored in a file system or in a repository that does not have SPARQL query and inference engines.

We are currently expanding work in certain directions. The first is finding ways to support the use of alternative graph patterns for type-matching patterns. To enable this, we must extend the query rewriting and scoring mechanisms. The second is finding ways to support the set match for grid resource discovery. Since SPARQL does not currently provide aggregate functions, the matchmaker is not able to use a SPARQL query engine to perform the set match. A matchmaker can, however, realize a set match by processing query results from a repository.

[8] These patterns are ignored by the query rewriting module.

References

1. http://www.daml.org/services/owl-s/1.1/.
2. http://www.wsmo.org/2004/d2/v0.2/20040306/.
3. http://www.w3.org/Submission/WSDL-S/.
4. http://www.w3.org/TR/owl-features/.
5. http://www.oasis-open.org/committees/wsrf.
6. http://www.globus.org/toolkit/docs/4.0/info/index/.
7. http://www.w3.org/TR/rdf-sparql-query/.
8. http://esw.w3.org/topic/SparqlImplementations.
9. http://www.w3.org/TR/2006/CR-rdf-sparql-XMLres-20060406/.
10. http://www.w3.org/TR/2006/CR-rdf-sparql-protocol-20060406/.
11. Paolucci, M., et al.: Semantic matching of web services capabilities. In: Proc. of the First International Semantic Web Conference on The Semantic Web. (2002) 333–347
12. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Proc. of the 12th international conference on World Wide Web. (2003) 331–339
13. Kuang, L., et al.: Exploring semantic technologies in service matchmaking. In: Proc. of the Third IEEE European Conference on Web Services. (2005) 226–234
14. Cardoso, J., Sheth, A.: Semantic e-workflow composition. Journal of Intelligent Information System **21** (2003) 191–225
15. Klusch, M., et al.: Owls-mx: Hybrid semantic web service retrieval. In: Proc. of International AAAI Fall Symposium on Agents and the Semantic Web. (2005)
16. Jaeger, M.C., et al.: Ranked matching for service descriptions using owl-s. In: Kommunikation in verteilten Systemen (KiVS 2005). (2005) 91–102
17. Benatallah, B., et al.: Semantic reasoning for web services discovery. In: WWW2003 Workshop on E-Services and the Semantic Web. (2003)
18. http://www.globus.org/toolkit/.
19. Said, M.P., Kojima, I.: Towards Automatic Service Discovery and Monitoring in WS-Resource Framework. In: Proc. of the First International Conference on Semantics, Knowledge and Grid. (2005) 932–938
20. Raman, R., Livny, M., Solomon, M.: Matchmaking: Distributed resource management for high throughput computing. In: Proc. of the Seventh IEEE International Symposium on High Performance Distributed Computing. (1998) 140–146
21. Raman, R., Livny, M., Solomon, M.: Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching. In: Proc. of the 12th IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-12). (2003) 80–89
22. Liu, C., Foster, I.: A Constraint Language Approach to Matchmaking. In: Proc. of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04). (2004) 7–14
23. Tangmunarunkit, H., et al.: Ontology-based Resource Matching in the Grid—The Grid meets the Semantic Web. In: Proc. of SemPG03. Volume 2870. (2003) 706–721
24. http://www.naics.com/.
25. http://www.unspsc.org/.
26. http://jena.sourceforge.net/.

A flexible model for Web service discovery

Rubén Lara[1], Miguel Ángel Corella[2] and Pablo Castells[2]

[1] Tecnología, Información y Finanzas, Madrid, Spain
`rlara@afi.es`
[2] Universidad Autónoma de Madrid, Spain.
{`miguel.corella,pablo.castells`}`@uam.es`

Abstract. The advent of the SOA paradigm is expected to cause an increase in the number of available Web services. In this setting, advanced facilities for the discovery of Web services that provide a given functionality are required so that this increase does not create a bottleneck for the exploitation of services in an SOA. In this paper, we present a model for the discovery of Web services which relies on alternative views of Web service functional capabilities, allowing for different trade-offs between the accuracy of discovery results and the efficiency of the discovery process. Up to three filtering steps, with increasing complexity, can be successively applied in order to refine discovery results. Furthermore, the use of taxonomies of functional categories, close to the intuition of average users, is introduced by our model with a two-fold purpose: a) supporting the user in the description of goals and Web services, and b) allowing for a coarse-grained but highly efficient discovery. Due to its flexibility, the model proposed is expected to cover a wide range of use cases.

1 Introduction

Service-Oriented Architectures (SOAs) are receiving increasing attention, as they promise an important gain in flexibility and scalability of IT systems and a reduction of the integration burden. However, the adoption of the SOA paradigm is not only expected to bring benefits but also to pose new research challenges. One of these challenges is the automatic location of Web services that can fulfill a given functional goal, and it constitutes the focus of this paper.

We concentrate on the exploitation of the formal description of Web service functional capabilities and of customer goals and, building on previous works ([13, 14, 16, 17]), we propose a Web service discovery model with some features that we believe can make it usable in a wide variety of scenarios, namely: a) it allows for different trade-offs between accuracy of results and response times, based on the application of different filters, b) it supports users in the description of their goals and Web services, c) it can provide meaningful results in short times if only the simplest filter is applied, and d) it can yield results of considerable accuracy if all available filters are successively applied. Additionally, a prototype of a registry and a discovery component implementing the model designed has been built as a proof-of-concept of our approach. This prototype is presented in the paper as well as some preliminary performance evaluation and complexity results based on the logical languages and reasoning tasks employed.

The paper is structured as follows: Section 2 introduces alternative views of Web service capabilities used in our model. The publication of Web services by providers and the description of goals by customers are presented in Sections 3 and 4, respectively. In Section 5, the application of registry-side filters for locating Web services that can resolve the goal at hand is described, and customer-side filters are presented in Section 6. Relevant related work is discussed in Section 7. Finally, our conclusions and future work are summarized in Section 8.

2 Web Service Capabilities

Web services are computational entities accessible using particular standards and protocols, and usable to request the provision of some services, being a service understood as a provision of value in some domain [13, 22]. For example, an airline can publish a Web service for the booking of its flights; this Web service can be used to request different particular bookings, each booking being a service.

The provision of a service results on the provision of some information to the customer (*service outputs*) e.g. the provision of weather information and/or on some changes in the real world (*service effects*) e.g. the actual booking of a seat on a flight; the functional value of the Web service is given by the outputs and effects associated to the services that can be requested through it. We call this functional value the functional *capability* of the Web service [23].

In our model, a provider can describe the functional capability of its Web service in different ways, namely: a) only syntactically using WSDL [5], b) assigning the Web service to one or more categories, and/or c) semantically, with different level of detail (only inputs, only results, or also the relation between both). The publisher will be free to choose among these types of description and to combine them.

For describing Web services, regardless of the type of description used, we use the WSMO framework [23] and the WSML [9] family of languages. However, our model could work with other frameworks. In this section, we present the alternative descriptions allowed in our model and how they are incorporated into the WSMO framework. We do not explain how WSDL Web services are described; for details on how WSDL is used to syntactically describe the function a Web service offers we refer to [5].

2.1 Semantic description with input-results relation

The service provided by a Web service and, thus, the results (outputs and effects) of the Web service execution depend on the information given by the customer to the Web service as *input values*, and on the state of the world that holds when a request is issued to the Web service. The booking provided by the airline Web service mentioned above will depend e.g. on the flight data given to the Web service and on seat availability, which is part of the state of the world. If we abstract from the possibly complex interaction required by Web services for providing their functionality, a Web service can be seen as a function that maps input values and the state of the world to some outputs and effects.

The first kind of description we will consider is a formal description of valid input values for the Web service and of possible results depending on the particular input values provided by the customer. We drop the dependency of Web service results on the current state of the world from the description, as the discovery process will have partial visibility of the current state of the world and, thus, a distributed, recursive evaluation of capability descriptions would be required if such dependency has to be considered by the discovery process, which can considerably degrade efficiency [16].

Figure 1 shows the description of a flight booking Web service. Required input values are encoded by a WSMO precondition: information of the flight to be booked (variable $?flight$), which must be between European cities and operated by airline *air*, and a *MasterCard* credit card (variable $?cc$), are required. The WSML language used for describing preconditions is WSML-Flight [9], as it will be explained in Section 6.

The conditions results of the Web service fulfill, dependent on the particular input values provided by the customer, are encoded by *input-dependent* postconditions, identified by the value of the *postconditionType* non-functional property. In the example, every result (variable $?result$) of the Web service will be a booking of a flight $?flight$, paid with credit card $?cc$. Variables $?flight$ and $?cc$ are declared in the capability as shared between the precondition and the postcondition and, therefore, the formalization of Web service results is dependent on the values given to these variables i.e. on the input values given by the customer, as we will see in Section 6.

The modelling style is the same one used in [16], and the description of input-dependent results is regarded as a not yet named Description Logics (DL) [20] concept definition that formalizes the conditions all results of the Web service fulfill, dependent on the input values given by the customer. The expressivity is restricted to the $\mathcal{SHOIQ}(\mathcal{D})$ DL [20], which is the description logic underlying WSML-DL with the addition of nominals, as shared variables in such concept definitions will be replaced by instances during discovery (see Section 6), and very close to the DL underlying OWL ($\mathcal{SHOIN}(\mathcal{D})$) [2]. In the following, we will denote WSML-DL *plus* nominals by WSML-DL+.

Notice that all results of the Web service, both outputs and effects, are encoded in the postcondition of the Web service capability. Outputs and effects are distinguished ontologically, as e.g. the $Booking$ concept is a subconcept of $Effect$ in the domain ontologies used, and there is a concept $InfoProvision$ for identifying the provision of information i.e. outputs. Ontologies are not shown due to space constraints.

2.2 Formal description without inputs-results relation

The second type of description we will consider is the formal description of valid input values and of possible results of the Web service, but without their relation. Remarkably, this is close to the kind of descriptions used by pure DL-based approaches to Web service discovery such as [4, 17, 21].

The description of valid input values is the same one introduced above, encoded in the precondition of the Web service of Figure 1, while the possible results of the Web service, independently of the particular input values given, are encoded by the *input-independent* postcondition of Figure 1, identified by the value of the *postconditionType* non-functional property. Notice that there is no shared variable between the precondition and the input independent postcondition.

The description of the input-independent postcondition, in WSML-DL+, is regarded as a not yet named DL concept definition that formalizes the conditions results of the Web service fulfill, *independently of input values*. The input-dependent and input-independent formalizations of results are related in the way described in [16].

2.3 Categorization

Web services can be assigned to functional categories i.e. to categories that capture a given functional value. For example, a Web service for booking flights can be assigned to a transport means booking category, and this category reflects to some extent the functionality of the Web service. The last type of description we will consider is the assignment of the Web service to (one or more) functional categories.

In Figure 1, the category the Web service is assigned to is *TransportMeansBooking@http://www.tifbrewery.com/seta/Taxonomy1* i.e. a transport means booking category defined at taxonomy *http://www.tifbrewery.com/seta/Taxonomy1*, and the assignment is encoded by the *category* non-functional property of the capability.

Taxonomies of functional categories are intuitive for average users if properly defined i.e. with an appropriate number of categories, properly arranged, with a clear meaning and a clear textual explanation, etc. Furthermore, we associate a formal meaning to categories in taxonomies; when a taxonomy of categories is defined, we associate to each category a formal and general description of the results expected from Web services assigned to that category, independently of possible input values. For example, the following WSML description is associated to a transport means booking category:

```
?result memberOf Booking[ofitem hasValue ?item] and ?item memberOf TransportMeans.
```

The description above formalizes general results, and particular Web services assigned to this category can more precisely formalize the results they offer e.g. booking of flights, possibly including how these results depend on input values. The language used for describing these general results is also WSML-DL+, and descriptions like the one above are regarded as a not yet named DL concept definition formalizing the conditions general results of Web services in a category fulfill.

3 Web Service Publication

Service providers can make their services accessible via Web service interfaces. In order to make a Web service usable by other parties, a provider will *publish* the Web service description at some network location reachable by target users. It is a common practice to publish syntactic WSDL descriptions of Web services at UDDI [3] repositories, which act as a common entry point for the location of Web services and provide keyword-based search facilities as well as search based on categories in taxonomies such as UNSPC. However, the publication of WSDL descriptions at UDDI repositories has two major limitations: a) the assignment of Web services to categories is completely manual, and b) the use of syntactic descriptions does not allow for advanced search based on formal semantics [24].

```
webService _"http://www.tifbrewery.com/seta/FlightBooking"
  ...
  capability FlightBookingCapability
    nonFunctionalProperties
      afi#category hasValue "TransportMeansBooking@http://www.tifbrewery.com/seta/Taxonomy1"
    endNonFunctionalProperties

    sharedVariables {?flight, ?cc}

    precondition
      definedBy
         ?flight  memberOf Flight[cityOrigin hasValue ?co, cityDestination hasValue ?cd,
          operatedBy hasValue air] and ?co memberOf City[inContinent hasValue europe] and
          ?cd memberOf City[inContinent hasValue europe] and ?cc memberOf MasterCard.

    postcondition
      nonFunctionalProperties
         afi#postconditionType hasValue "inputDependentPostcondition"
          afi#intention  hasValue "all"
      endNonFunctionalProperties
      definedBy
         ?result  memberOf Booking[ofItem hasValue ?flight, withPaymentMethod hasValue ?cc].

    postcondition
      nonFunctionalProperties
         afi#postconditionType hasValue "inputIndependentPostcondition"
          afi#intention  hasValue "all"
      endNonFunctionalProperties
      definedBy
         ?result  memberOf Booking[ofitem hasValue ?item, withPaymentMethod hasValue ?pm] and
                ?item memberOf Flight[cityOrigin hasValue ?co, cityDestination hasValue ?cd,
                operatedBy hasValue air] and ?co memberOf City[inContinent hasValue europe] and ?cd
                memberOf City[inContinent hasValue europe] and ?pm memberOf MasterCard.
```

Fig. 1. Example Web service

In this setting, we have implemented a registry (Figure 2) using a UDDI repository (jUDDI[1]) and a DL reasoner (RacerPro [1]), enabling the retrieval of Web services based on the semantic description of the results they offer and supporting the publisher in describing Web services. A taxonomy manager, with a Web service interface, allows for the management of category taxonomies, and a publication manager is in charge of the publication of Web services at the registry, making use of jUDDI, RacerPro, and a categorizer. Both are introduced in this section. The query manager allows for the location of Web services in the registry (see Section 5). We note that the UDDI API implemented by jUDDI can also be directly used for the syntactic search of Web services.

The registry is based on [24], but we allow for arbitrary combinations of concepts in domain ontologies to describe the results of Web services, and we incorporate the usage of taxonomies of categories.

3.1 Classification in categories

When a Web service is published at the registry, it is first assigned to one or more functional categories as it will be explained below. However, before any Web service

[1] http://www.juddi.org/

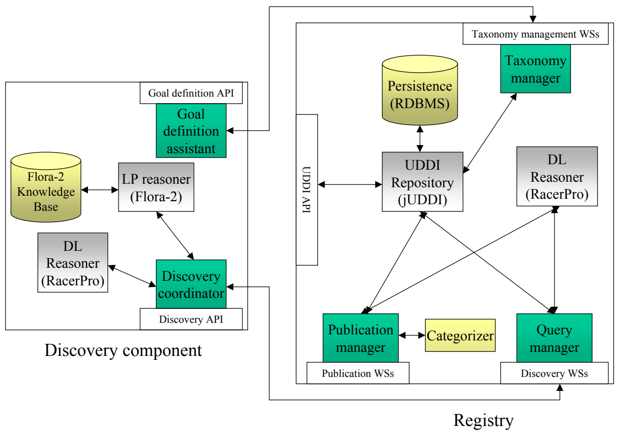


Fig. 2. Registry and discovery component architecture

can be assigned to a category, such category must be included in the registry. This is done through the taxonomy manager, which allows for the definition of taxonomies of categories and publishes them at the UDDI repository used by our registry. The definition of categories includes the formal description of the general results offered by Web services in that category (see Section 2.3).

Categorization of WSDL Web services. If the publisher only has available a WSDL description of the Web service, he can only *manually* browse category taxonomies known by the taxonomy manager and assign his Web service to (one or more) categories, as done in current UDDI repositories. If no category is selected and no further information is given by the publisher, we will have no semantic description of the Web service and it will be only published at jUDDI and searchable through the UDDI API.

If the Web service is assigned to one or more categories, the formal description of the general results expected from Web services in these categories will be retrieved and shown to the publisher so that it serves as a starting point for formally describing his Web service. The publisher can refine this description and possibly add the formal description of input values and the relation between Web service results and these values. For example, if the category chosen is *TransportMeansBooking*, the formal description retrieved will be the one given in Section 2.3, which the publisher can refine to describe e.g. that the Web service only offers the booking of flights, as done in the input independent description of results of Figure 1. In the simplest case, the customer will not refine the results of the categories used, and the formalized results of the Web service will be the intersection of the results of these categories. A WSMO description is generated to include the formal details available for publication.

Categorization of semantic Web services. If the publisher has already available not only a WSDL description but also a semantic description (of results and optionally of input values) of his Web service, our classification framework can assist him in the assignment of the Web service to categories. The publication manager uses the categorizer in Figure 2 to heuristically, and based on the semantic description of inputs and results, propose the categories from existing taxonomies that best fit the Web service. This is done by measuring the similarity between the semantic description of the Web service inputs and results, and the inputs and results of Web services published at the registry and already assigned to a given category (see [6, 7] for details). In our current implementation, only the type descriptions (i.e. named classes from domain ontologies involved in inputs and results descriptions) are used.

After this heuristic classification, the publisher is given an ordered list of the categories his Web service would most likely belong to, thereby assisting him in its categorization. Still, the publisher will make the final choice of categories.

We have measured classification times using a repository of 164 classified semantic Web services[2]. This repository has been automatically translated from OWL-S (its current format) to WSMO. Since the classification heuristic needs a repository of classified Web services (the training set) in order to provide meaningful results, it is not possible to use randomly generated services for testing due to the cost of manually classifying them. Therefore, our tests are limited to a set of 164 Web services, obtaining a response time of around 650 milliseconds for classifying a new Web service when 156 of them (the training set) are already classified[3].

Consistency checking. Once the publisher has assigned his Web service to categories and sent its final WSMO description to the publication manager, the description of the input-independent results of the Web service and the description of the results associated to the selected categories are checked for consistency [20] so that the Web service is not assigned to categories whose general results are not consistent with the results the Web service declares. For example, we will check that the input-independent postcondition of the Web service in Figure 1 is consistent with the formalization of results of the transport means booking category in Section 2.3. If only a WSDL description was initially given, this check will only be necessary if the publisher refines the formalization of results associated to the selected categories.

Both descriptions, in WSML-DL+, are checked for consistency using RacerPro. However, RacerPro only offers sound and complete reasoning for $\mathcal{SHIQ}(\mathcal{D})$ and, therefore, nominals [20] are translated into pair-wise disjoint concepts before they are sent to the reasoner. Some incorrect inferences can be drawn from the resulting translation [11], but in most cases the subsumption and satisfiability relations computed will be correct. Recently, a sound and complete reasoning procedure for $\mathcal{SHOIQ}(\mathcal{D})$ has been implemented in the Pellet[4] reasoner; testing the use of Pellet instead of RacerPro will be part of our future work.

We have generated random variations of an initial set of Web services in order to measure the time required for consistency checking. This task is time consuming, as checking concept satisfiability is NExpTime-complete for the $\mathcal{SHOIQ}(\mathcal{D})$ DL [19]. In our tests, we have measured times of around 20 seconds for checking satisfiability

[2] From http://www.few.vu.nl/~andreas/projects/annotator/owl-ds.html

[3] For all tests in the paper, an Intel Pentium 4 2.8 GHz, 1GB RAM computer has been used.

[4] http://www.mindswap.org/2003/pellet/

of concepts referring to an ontology of around 800 concepts. However, these times are acceptable as publication is not time-critical[5].

3.2 Results-based classification

After checking consistency of category results and Web service results, the publication manager will store a business service [3] at jUDDI, which will have a unique key $key_{\mathcal{W}}$ assigned automatically by the repository. This business service includes: a) the complete WSMO description of the Web service, b) the functional categories the Web service has been assigned to, and c) the WSDL description of the Web service, if available, so that the Web service can be searched syntactically through the UDDI API.

Afterwards, the Web service will be classified in a subsumption hierarchy attending to the formal, input-independent description of its results. For this purpose, a $\mathcal{SHOIQ}(\mathcal{D})$ concept is defined, being the name of the concept the unique key $Key_{\mathcal{W}}$ given by the UDDI repository, and the definition of the concept the one given by the input-independent postcondition of the Web service. For example, the following concept is defined for the Web service from Figure 1:

```
?result memberOf Key equivalent ?result memberOf Booking[ofitem hasValue ?item,
withPaymentMethod hasValue ?pm] and ?item memberOf Flight[cityOrigin hasValue ?co,
cityDestination hasValue ?cd, operatedBy hasValue air] and ?co memberOf City[inContinent hasValue
europe] and ?cd memberOf City[inContinent hasValue europe] and ?pm memberOf MasterCard.
```

This concept is sent to the T-Box [20] of the DL reasoner (after translating nominals), and the T-Box is classified i.e. the subsumption relation between concepts in the T-Box is computed. In this way, published Web services will be arranged in a subsumption hierarchy in terms of the results they can provide, which we will be able to query efficiently. As checking subsumption can be reduced to the reasoning task of checking satisfiability, T-Box classification is also NExpTime-complete. We have measured times of around 160 seconds for classifying the T-Box when 2000 Web services are already published, and using an ontology of around 800 concepts [16]. However, classification is done off-line and, therefore, it is not a time-critical task as long as publication times are kept within certain limits.

Summarizing, when a Web service is published it will be assigned to categories, stored at the jUDDI repository, and classified at the T-Box of the DL reasoner according to the formalization of the results it can provide. Remarkably, the publisher is assisted in assigning Web services to categories and in formalizing their results based on the formal meaning associated to categories in taxonomies. In addition, Web services described with different level of detail can be published, including purely syntactic WSDL descriptions. However, the kind of semantic description given at publication time will condition how Web services can be later evaluated by the discovery process.

4 Customer Goals

Customers will be interested in using Web services because of the functional value they offer i.e. because of the outputs and effects they can provide. Therefore, customer goals

[5] We note, however, that beyond a certain number of Web services published, the times required for consistency checking can start to be too high and indeed problematic

have to describe the functional value expected from Web services. In this section, we discuss how goals are described in our model.

4.1 Description of goals

The functional value expected by a customer can be described: a) by giving functional categories in available taxonomies, or b) by formalizing the results (outputs and effects) expected. In the first case, the customer can browse category taxonomies and select the categories sought Web services must belong to. The use of categories for locating Web services is intuitive for the end-user, as he does not have to deal with formal descriptions, and, as we will see in the next section, response times are low.

However, the results obtained using categories for discovery are coarse-grained. For this reason, we also try to make the use of formal descriptions easier for customers so that they can specify their expected results in more detail if required. In particular, we use the formal descriptions associated to categories as follows: a customer can browse category taxonomies and select categories he is interested in; when these categories are selected, we retrieve the formal description of results defined for these categories so that they can be refined by the customer for accurately describing his goal. In this way, we provide the user with basic support for formalizing the results he requires.

Figure 3 shows a goal description, where a category is encoded as a non-functional property of the goal, and the results expected (the booking of a flight $myFlight$, paid with credit card $myCC$) are encoded in the postcondition. The language used for formalizing expected results is WSML-DL+.

The customer must also associate a universal or existential intention to the formalization of expected results (encoded as a non-functional property *intention*), meaning that he wants to obtain all or only some of the results from the set formalized. This intention will influence what Web services are considered a match, as explained in [13].

We have implemented a discovery tool that supports the user in describing goals by exploiting taxonomies of categories. In particular, it enables browsing taxonomies and retrieving the formal descriptions associated to categories so that they can be used as a starting point for formalizing expected results. The tool uses the discovery component of Figure 2, which is installed at the customer side. In particular, it uses its goal definition assistant, which communicates with the taxonomy manager in the registry to retrieve information about taxonomies of categories.

Level of accuracy. Once either a functional category or a formalization of expected results is given (including its intention), the second ingredient in the description of a goal is the level of accuracy expected from discovery results. In particular, the customer can choose to apply different filters to obtain Web services that can potentially fulfill his goal. We split these filters into registry-side filters, which do not take inputs into account, and customer-side filters, which are applied over the Web services obtained from the registry after applying registry-side filters and which focus on availability of input values and on how these values condition Web service results. Only customer-side filters consider input values because, as we will explain below, potential input information is kept locally by the customer.

The customer can choose to apply one of these two registry-side filters: 1) category-based, or 2) results-based. Two customer-side filters can be successively applied: a) input-availability, and b) input-results relation. Registry-side and customer-side filters will be presented in Sections 5 and 6, respectively. For formal details on the results-based, input-availability and input-results relation filters, we refer the reader to [16].

4.2 Input information

If customer-side filters are applied, we evaluate whether the customer has appropriate input values for relevant Web services, and possibly whether these input values would lead to the results required, is evaluated. In this setting, we assume a knowledge base KB exists, containing: a) the information the customer knows and he is willing to disclose for achieving his goal, and b) the domain ontologies $\mathcal{O}$ providing the domain vocabulary.

KB will be kept local, as it might be big in volume and it might include customer's sensitive information e.g. credit card details, which should not be disclosed to the registry or to any third-party at discovery time [16]. In our current prototype, KB is implemented as a Flora-2[6] knowledge base which will be queried as described in Section 6, and which is defined once and used for resolving all goals issued by the customer. The information in the Flora-2 knowledge base will also be stored in the DL reasoner of the discovery component (see Figure 2).

Regarding the expressivity allowed for describing information in KB, ontologies in $\mathcal{O}$ must be described using WSML-Core [9] and, thus, restricted in expressivity to Description Logic Programs (DLP) [10]. Furthermore, the information the customer knows must be described as instances of concepts in $\mathcal{O}$ and, thus, as instances of WSML-Core concepts. In this way, we can consistently refer to instances of the same domain ontologies from WSML-Flight and WSML-DL+ descriptions (see [9, 16] for details), and use the same domain model and customer knowledge in Flora-2 and RacerPro. Remarkably, and according to [26], most ontologies in popular ontology libraries fall in the DLP fragment of first-order logic.

In a nutshell, goals are defined by either categories in category taxonomies or a WSML-DL+ formalization of expected results, plus the filters that must be applied for finding relevant Web services (encoded by the *filter* non-functional property of the goal). Additionally, if customer-side filters are selected, a knowledge base KB containing the information on the customer side that can be used as input values to relevant Web services must be in place.

5 Registry-side Filters

The discovery component depicted in Figure 2 is installed at the customer side, and the customer communicates with it to resolve his goal. In particular, the customer submits his goal to the discovery coordinator of Figure 2, which first sends it to the registry query manager using the Web service interface exposed for this purpose. The query

[6] http://flora.sourceforge.net

```
goal _"http :// www.tifbrewery.com/seta/FlightBookingGoal"
  ...
  capability FlightBookingGoalCapability
    nonFunctionalProperties
      afi#category hasValue "TransportMeansBooking@http://www.tifbrewery.com/seta/Taxonomy1"
        afi#filter   hasValue {"ResultsBased", "InputAvailability"}
    endNonFunctionalProperties

    postcondition
      nonFunctionalProperties
        afi#intention  hasValue "some"
      endNonFunctionalProperties
      definedBy
        ?result  memberOf Booking[ofitem hasValue myFlight, withPaymentMethod hasValue myCC].
```

Fig. 3. Example Goal

manager will apply the registry-side filter selected and return relevant results to the discovery coordinator, as it will be explained in the following.

5.1 Category-based filter

If the category-based filter is selected, the query manager extracts the categories specified by the *category* non-functional property, which are the categories sought Web services must belong to (interpreted as a logical *and*).

As Web services in the registry are already categorized, the query manager will simply invoke the UDDI *find_service* operation [3] specifying the categories given by the goal, and the keys of Web services assigned to these categories will be retrieved. Afterwards, the query manager will query the UDDI repository through the *get_serviceDetail* operation, specifying the keys of relevant Web services, in order to retrieve their complete descriptions. The complete descriptions of the obtained Web services, categorized as required by the goal, are returned to the discovery coordinator.

The operations above perform simple lookups in the relational database used for persistency of the UDDI repository and, thus, response times are low. According to our tests, response times are below 300 milliseconds for 2000 Web services categorized at the registry and remain almost constant. While results will be provided in short times, and based on the use of categories intuitive for end-users, they will be coarse-grained, as the granularity of results will be limited by the granularity of categories in taxonomies.

5.2 Results-based filter

If the results-based filter is applied instead, the query manager will query the DL reasoner T-Box for concepts: a) equivalent to, b) more general than, c) more specific than, and d) with a non-empty intersection with the description of the set of results expected by the customer, given by the postcondition in the goal description after replacing nominals i.e. concepts describing sets of offered results related in some way to the results expected are retrieved. In practice, as RacerPro does not allow for querying for *all* concepts intersecting a given concept, we query for Web services *not intersecting* the set of results expected. Intersecting concepts will be obtained by taking all available concepts representing Web service results but those not intersecting the goal and those having a subsumption (equivalent, more general, or more specific) relation with the goal. Once these concepts are retrieved, we will use the matching notions in [13] to determine which of them are a perfect, possible perfect, partial, or possible partial match for the goal, taking into account intentions.

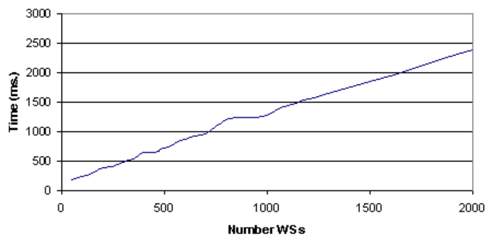


Fig. 4. Times for the application of the results-based filter

The keys of matching Web services are obtained from the name of the concepts retrieved, and jUDDI is queried for the complete description of Web services with these keys. The query manager returns to the discovery coordinator four lists with complete Web service descriptions, one per type of match. In Figure 4, the time required for retrieving Web services from the registry applying the results-based filter is shown as a function of the number of Web services published with an input-independent description of their results. A remark is in place: querying RacerPro for equivalent, more general or more specific concepts yields response times below 20 milliseconds for 2000 Web services published (see [16]), while querying for non-intersecting concepts (in order to afterwards obtain intersecting concepts) is the most time-consuming operation and increases with the number of available Web services.

6 Customer-side Filters

The application of a registry-side filter results in a number of relevant Web services being returned to the discovery coordinator. If the results-based filter is applied, four lists of Web services are returned, corresponding to the degrees of match identified in [13]. If the category-based filter is applied, a single list is returned, being all Web services considered perfect matches. If no customer-side filter is chosen, the discovery coordinator directly provides these Web services to the customer. Otherwise, customer-side filters are applied as explained below.

6.1 Input availability filter

Besides checking whether a Web service is relevant for solving a customer's goal, either in terms of the categories it belongs to or in terms of the results it can provide, the customer might want to know whether the input values required by the Web service can be provided i.e. whether the customer has appropriate information available to be submitted to the Web service as input values.

For this purpose we will use the preconditions of Web services returned after the application of a registry-side filter as queries to the knowledge base KB formed by the customer knowledge and the domain knowledge i.e. we query for valid input values for the Web service. For example, the precondition in Figure 1 will be used as a query for values for variables $?flight$ and $?cc$. Preconditions are expressed in WSML-Flight and, thus, they have Logic Programming (LP) semantics [18]. As discussed in [16], LP semantics are appropriate in this setting as we want to determine whether valid input values can be provided to the Web service *from knowledge base KB* (close world) i.e. whether there is, at the customer side, information to be provided as input values to the Web service. Furthermore, for obtaining valid input values we are only interested in efficient query answering, not in general entailment.

The knowledge base KB, implemented as a Flora-2 knowledge base, is initialized before any goal is issued to the discovery component. Therefore, it is already compiled and loaded when a goal has to be resolved, which makes querying for valid input values using the description given by Web service preconditions efficient (less than 50 milliseconds for a knowledge base with more than 14000 randomly generated facts). In fact, query answering for WSML-Flight (Datalog with stratified negation and inequality) can be done in polynomial time [8].

Input availability is a boolean filter i.e. the input information requirements of a Web service are either fulfilled or not. However, as we allow providers to publish Web services with different level of detail in their description, there might be Web services retrieved from the registry that do not describe the input values they require. Therefore, the discovery coordinator will return to the customer: 1) Web services that passed the input availability filter, and 2) those to which the filter could not be applied. Web services will be grouped into these categories, and their complete descriptions will be returned to the customer along with their degree of match given by the application of registry-side filters.

For Web services that pass the input availability filter, we obtain what valid input values were found. In particular, the queries described by Web service preconditions yield a set of assignments of instances in KB to input variables. In particular, if the set of input variables is $I = \{i_1, \ldots, i_n\}$, different sets of assignments of instances to variables can be obtained, being each such set denoted by $I[j]$. If we consider the Web service in Figure 1, we could assign e.g. some instances $myFlight$ and $myCC$ to variables $?flight$ and $?cc$ (assignment set $I[1] = \{flight/myFlight, cc/myCC\}$), or some instances $myFlight$ and $myCC2$ (assignment set $I[2] = \{flight/myFlight, cc/myCC2\}$), where $myFlight$ is a flight between European cities operated by airline air, and both $myCC$ and $myCC2$ are MasterCard credit cards. These instances, and the complete knowledge base KB, are available both to the DL and the LP reasoners.

As a WSML reasoner is not completely available yet, we use directly Flora-2. For this purpose, we need to translate WSML-Flight descriptions to Flora-2 syntax. This is currently done manually before publishing Web services, and preconditions are encoded using Flora-2 syntax besides WSML syntax in Web service descriptions [16].

The same applies to the use of RacerPro: we currently translate WSML-DL+ descriptions to RacerPro syntax and incorporate descriptions in both syntaxes into goal and Web services.

6.2 Input-results relation filter

As discussed in Section 2, there is a relation between Web service results and the input values provided by the customer. If the input-results relation filter is selected, we will not only check whether the customer has available appropriate input values for relevant Web services retrieved from the registry, but also the relation between these input values and Web service results.

Given the assignment sets obtained from applying the input-availability filter, we can replace shared variables in the input-dependent postcondition (which are part of the set of input variables) by input values, yielding a WSML-DL+ concept definition. Notice that, as these values are instances of WSML-Core ontologies, we can consistently use them in WSML-DL+ descriptions. Each assignment set, corresponding to the usage of the Web service with different input values, is considered. The concept defining the results offered by a Web service for available input values is defined by the union of the results for each such set of input values. If we continue with the example above, the assignment sets $I[1]$ and $I[2]$ yield a concept WS (where WS is the Web service identifier) as follows:

```
?result memberOf WS equivalent ?result memberOf Booking[ofItem hasValue myFlight,
      withPaymentMethod hasValue myCC] or ?result memberOf Booking[ofItem hasValue myFlight,
      withPaymentMethod hasValue myCC2].
```

This concept will be published at the T-Box of the DL reasoner used by the discovery component (Figure 2) after replacing nominals. The discovery coordinator will then query for concepts: a) equivalent to, b) more general than, c) more specific than, and d) intersecting with the description of the set of results expected by the goal (with nominals replaced). The corresponding intentions will be applied as described in [13], yielding the list of Web services that are a perfect, possible perfect, partial, or possible partial match for the goal for the input values available. In a nutshell, we obtain Web services that offer, *for the input values available*, the results expected by the customer, considering how these results depend on the input values provided i.e. we evaluate the relation between input values and Web service results.

This filter can only be applied if the goal formalizes the set of results expected, as this formalization must be used in the queries to the DL reasoner for relevant Web services. Furthermore, as Web services might not describe input-dependent postconditions, the discovery coordinator returns to the customer: 1) Web services that passed the input-results relation filter, 2) those that passed the input availability filter but to which the input-results relation filter could not be applied, and 3) those to which none of the filters could be applied. Web services will be grouped into these categories, and their complete descriptions are returned to the customer along with their degree of match.

Besides querying the Flora-2 knowledge base for valid input values, the DL concepts obtained from replacing parameters with these values have to be published and the DL reasoner has to be queried for concepts having certain relation with the results required by the goal. This querying is time-consuming, as new concepts are published during discovery and, thus, the T-Box of the DL Reasoner cannot be fully classified before-hand. While we expect most Web services to be discarded by previous filters so that not many Web services have to be evaluated in this way, the times required are still high e.g. around 60 seconds for 500 Web services evaluated. Therefore, this filter should only be applied if a detailed evaluation of Web services is needed.

7 Related Work

Most (semantic) web service discovery proposals are based on the semantic matching of Web services and goals based on DL reasoning e.g. [4, 17, 24], focusing on a single logical filter and not considering neither the relation between input values and Web service results, nor alternative filters.

Works such as [14] and [12] take into account how the results of using a Web service depend on the input values provided to it. The former makes use of Transaction Logic and Logic Programming, which presents some problems with the treatment of unspecified information in goals and Web service capability descriptions. The latter, based on DL, expects customers to specify what kind of relation they expect between results and inputs, which we believe is a less usable approach than that taken by our model. None of them explicitly considers the combined application of other filters.

The only approaches we are aware of that apply different types of filters are LARKS [25] (for the matching of agents) and OWLS-MX [15] (for the matching of OWL-S Web services). While they are close in spirit to our model, allowing for the application of different types of filters, they differ in the following major respects: a) logical filters are restricted to first-order semantics, b) whether the customer can actually provide appropriate input values is not evaluated, c) the relation between available input values and Web service results is not described and exploited, d) all matching is done at the registry side, and e) customers are not supported in describing goals and Web services.

It must be noted that the type of filters considered by OWLS-MX could be incorporated to our model as registry-side filters. We are currently adding to our prototype a filter that matches textual descriptions of goals and Web services, but only based on the simple keyword matching provided by jUDDI; nevertheless, extending it in the direction of more complex syntactic similarity measures is a feasible task.

8 Conclusions and Future Work

We have presented a model and a prototype that accommodates different levels of accuracy in the description of Web services and goals, and offers discovery results with different trade-offs between accuracy and efficiency, so that it is usable in different application scenarios. We also incorporate the use of taxonomies of categories in order to provide basic assistance to the user in the semantic description of Web services and goals, and in order to obtain coarse-grained discovery results in short times. Future work will concentrate on optimizing the discovery prototype, on the refinement of the heuristic classification of Web services in taxonomies, on the proposal of new filters and the integration of existing ones such as those used by OWLS-MX, and on studying possible extensions of the expressivity allowed for domain ontologies.

References

1. RacerPro user's guide. version 1.9. Technical report, RACER Systems GmbH & Co. KG, December 2005.
2. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. Technical report, W3C Recommendation, Feb 2004.
3. T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, Maryann Hondo, Y.L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen. UDDI Version 3.0, 2002.
4. B. Benatallah, M. Hacid, A. Leger, C. Rey, and F. Toumani. On automating web services discovery. *VLDB*, 14:84–96, 2005.
5. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, March 2001.
6. M. A. Corella and P. Castells. A heuristic approach to semantic web services classification. In *KES-2006*, 2006.
7. M. A. Corella and P. Castells. Semi-automatic semantic-based web service classification. In *semantics4ws'06 workshop at BPM 2006*, 2006.
8. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of Logic Programming. *ACM Computing Surveys (CSUR)*, 33(3):347–425, 2001.
9. de Bruijn, J. (ed.). The Web Service Modeling Language WSML. WSML d16.1v0.21, 2005.
10. B.N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *WWW'03*, 2003.
11. I. Horrocks and U. Sattler. Optimised Reasoning for $\mathcal{SHIQ}$. In *ECAI 2002*, pages 277–281, July 2002.
12. D. Hull, E. Zolin, A. Bovykin, I. Horrocks, U. Sattler, and R. Stevens. Deciding semantic matching of stateless services. In *AAAI-06*, 2006.
13. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *ESWC 2005*, Heraklion, Greece, May 2005.
14. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *SWSs Worshop at ISWC 2004*, 2004.
15. M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *AAMAS 2006*, 2006.
16. R. Lara. Two-phased web service discovery. In *AI-driven Service Oriented Computing workshop at AAAI 2006*, July 2006.
17. L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *WWW'03*, Budapest, Hungary, May 2003.
18. J. W. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag, 1987.
19. C. Lutz. An improved nexptime-hardness result for description logic alc extended with inverse roles, nominals and countins. Technical report, Technical University Dresden, 2004.
20. D. Nardi, F. Baader, D. Calvanese, D. L. McGuinness, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge, January 2003.
21. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In *ISWC 2002*, pages 333–347. Springer Verlag, 2002.
22. C. Preist. A Conceptual Architecture for Semantic Web Services. In *ISWC 2004*, 2004.
23. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, D. Fensel, and C. Bussler. Web Service Modeling Ontology. *Applied Ontology Journal*, 1(1), 2005.
24. N. Srinivasan, M. Paolucci, and K. Sycara. Adding OWL-S to UDDI, implementation and throughput. In *SWSWPC Workshop at ICWS 2004*, 2004.
25. K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *AAMAS 2002*, 2002.
26. Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, 2004.

Recovery of Faulty Web Applications through Service Discovery

M.G. Fugini, E. Mussi

Politecnico di Milano
Dipartimento di Elettronica ed Informazione
Via Ponzio 34/5 - I-20133 Milano
fugini,mussi@elet.polimi.it

Abstract. Failures during Web service execution may depend on a wide variety of causes. In this paper we illustrate how, upon faults of a Web application, the faults can be handled by discovering substitute Web services and re-orchestrating the application. Self-healing capabilities are added to Web service environments. Possible recovery actions at the *Web service* and *Web application* levels are illustrated and discussed with respect to a running example involving coordinated Web services.

1 Introduction

Even if various efforts have been recently done in Service Oriented Computing, the adoption of Web services still remains limited. Especially under the open world assumption, the Service Oriented Architecture, on which Service Computing relies, presents some limitations. In fact, if we consider public available Web service registries, as UDDI, it is easy to find Web services no longer available or Web services having a description which does not correspond to the currently provided Web service, due to new versions or modifications of functionality of the Web services.

As discussed in this paper, even in a more controlled closed world assumption, there are many possible causes of failure. As closed world, we consider the adaptive Web service environment studied in the MAIS (Mobile Adaptive Information Systems) project [13]. In such an environment, adaptivity is provided at both the front-end and the back-end of Web applications. These are dynamically created composing Web services and are able to satisfy both functional and non-functional user requirements. This approach results in a context-aware Web service discovery and selection, which can be performed at runtime, during Web application execution. Under this approach, the management of *failures* poses new research problems related to the identification of causes, of possibly overlapping failure, and to their automatic correction.

The WS-Diamond (Web service DIAgnosability, MONitoring, and Diagnosis) project is aimed at developing methodologies for the creation of *self-healing Web Services*, able to detect anomalous situations, which may manifest as the inability to provide a service or to fulfill Quality of Service (QoS) requirements, and to recover from these situations, e.g., by rearranging or reconfiguring the network of services. WS-Diamond will also provide methodologies for the design of services, supporting service design and service execution mechanisms that guarantee diagnosability and reparability of run-time failures (e.g., thanks to the availability of set of observables, of exception handlers, or of sets of redundancies in the services or alternatives in the execution strategies).

The goal of this paper is to present an architecture for self-healing Web services and applications, which contains modules for the detection, diagnosis, and repair of faults. Our goal is to show how faults in Web application can be repaired using repair actions that include switching to a substitute service, by searching it in UDDI registries, based on similarity criteria. In particular, we focus on healing mechanisms based both on service selection and substitution and on addition of new services in composed processes to support self-healing functionalities.

Section 2 introduces self-healing, diagnosis, and quality of service issues in Web services and describes some approaches proposed in the literature. Section 3 presents our architectural scenario of a self-healing Web service environment, and an example. Section 4 introduces an architecture supporting the detection and recovery of possible faults. Section 5 discusses *reactive* and *proactive* healing techniques, based on service execution analysis.

2 Related Work

Recently, self-healing systems have attracted vast attention by the research community [11]. Examples of self-healing architectures are provided in [16]. The work in [16] presents a system architecture to monitor, interpret and analyze system events in order to implement self-healing and self-adaptive systems. The architecture presented in [15] is focused on service level agreement management in a Service Oriented Architecture. The goal is to re-optimize the provisioning infrastructure as a consequence of QoS violations. The work in [9] presents the requirements of a Web Service Management (WSM) framework which also includes the typical functionalities addressed in self-healing systems, analyzing and comparing multiple alternative architectures for the implementation of WSM systems i.e., centralized, federated and peer-to-peer. They propose Web service substitution and complex service re-compositions as repair actions.

In [2, 3, 17], authors propose a model based approach to implement diagnosis functionalities in Web services self-healing environments. The works in [2, 3] model a composite service as a Petri Net and classify Web service invocations. The goal is to correlate input and output parameters and to determine if an activity carries out a computation that may fail and produce erroneous outputs. A service invocation forwards an input parameter, if an output parameter coincides with the value of an input. If the output parameter is created during the service invocation this is classified as a source. Finally, when an output parameter is computed during the service invocation from one or more inputs the service invocation is an *elaboration.* A service invocation can introduce errors in parameters it computes (elaborations) or it produces (sources), while for forwarded parameters it possibly propagates errors generated by other invocations. Authors propose a centralized diagnoser which identifies faults from "alarms" identified by comparing the value of state variables at checkpoints introduced by the composite Web service designer.

A different approach is used in [10], where authors describe an extended Petri net model for specifying exceptional behavior in workflow systems. The core of this approach are the recovery policies, a combination of generic constructs and primitive operations that can be defined for both single tasks and sets of tasks (i.e., recovery regions). At design time, process designers model exceptions using the generic constructs, while at runtime the workflow system uses the primitive operations to handle the occurred exceptions.

With respect to the cited works, our approach uses dynamic Web service discovery and selection as one of the main repair strategy. Given a faulty application, we are able to substitute one or more component Web services with compatible ones. The selection of compatible services is performed using semantic criteria based on service descriptors stored in an enhances UDDI registry. Moreover the selection involves both functional and non functional similarity evaluations.

3 Adaptive Web Service

WS-Diamond aims at realizing a set of tools that can be plugged into existing Web service environments and provide self-healing features during the Web service life cycle.

With respect to the traditional SOA, the adopted system also supports Web service adaptation (as studied in MAIS [12]). In particular, we suppose hat a set of actors collaborates according to a shared choreography, which defines how and when the actors are involved during this collaboration. Operationally, each actor declares its ability to perform one or more activities included in the choreography specification, offering one or more Web services. According to this scenario, in this work we consider a *Web application* as a collaboration. Thus, a Web application is a distributed application and the choreography is responsible for describing the collaboration between all the involved Web services.

The reference architectural aspects are illustrated in the following with respect to the dimension of *cooperation among distributed nodes*. Figure 1 shows a global self-healing system in which WS-Diamond-enabled nodes can cooperate with non WS-Diamond-enabled nodes. In addition, each WS-Diamond-enabled node may provide only a subset of the modules developed in the project. The main modules of a complete WS-Diamond-enabled node are: i) a management interface for Web services; ii) a process orchestration engine for enacting composed services; iii) a repair action selector; iv) a diagnosis infrastructure; v) a fault detection infrastructure.

In the paper, we illustrate how the different modules cooperate *inside a node*. Cooperation and negotiation of available substitute services occur via a contract-based approach.

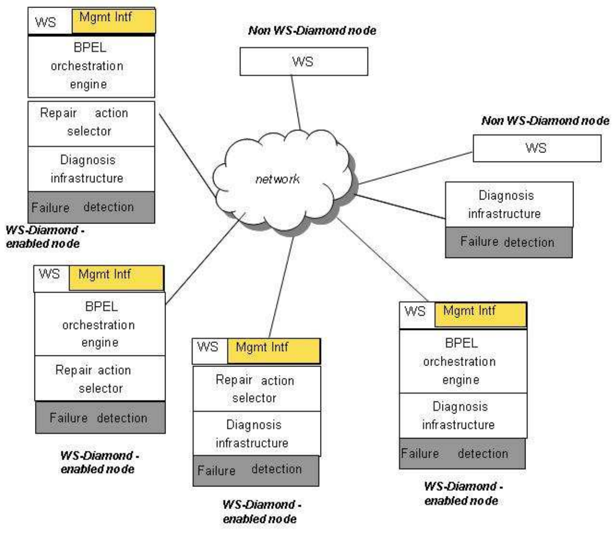


Fig. 1. Reference architecture

3.1 WS-Diamond Nodes

We assume that, within a WS-Diamond node, not all the diagnosys and repair features are available, but rather that each node is equipped with the fault and self-healing modules necessary to cooperate. The cooperation of the modules inside a node and within the execution environment main components views the diagnoser as the module to be notified of fault events through messages or events generated by the hardware/software infrastructure (including the self-healing system itself). By accessing messages, state logs and a fault database (all fault events are stored in a fault database), the diagnoser identifies which occurred fault needs to be recovered. The repair action selector performs a choice among a set of possible repair actions associated to each type of fault, as indicated in a *repair rules registry*. The selection triggers a repair action request to a repair module associated to the required action. A possible list of repair modules contains: a substitution module (to replace services during the orchestration of a composed service), a wrapper generator (to change parameters to solve incompatibility during invocation), a quality module (to perform data quality checks and improvements), and a reallocation module (to reallocate the resources exploited by the services).

Our cooperating environment might be technologically heterogeneous. In fact, an actor involved in the cooperation can rely either on a classical node or on a WS-Diamond node. About the former, Web services run over typical application servers providing a traditional Web service container. About the latter, we suppose to use a special application server, called MAIS-P [13], specifically designed to support adaptive Web services.

In the MAIS-P architecture, Web services run on the assumption that something might go wrong at invocation and execution time and, for this reason, a QoS driven approach is introduced to solve this possible critical situations. The service provider and a service requestor, in fact, agree on the *quality* that the Web service must guarantee during its execution. The requestor may specify quality requirements at Web service invocation time or these requirements may be implicitly specified in the user profile [5]. If a host realizes that the QoS of a Web service is decreasing to an unacceptable level, then two strategies can be adopted: channel/partner switching, to provide the Web service on a channel or through a partner with better QoS characteristics, or Web service substitution, selecting an alternative Web service for the user. Alternative Web services are searched by accessing an extended UDDI registry capable of assessing the similarity among Web services with respect to the provided functionality and the behavioral equivalence [4]. Similarity computation, supported by a Web service ontology, is based on a semantic-based analysis of the involved Web service. Since substituted and substituting Web services might have different signatures, an automatic wrapper generator can reconciliate differences in the provided interfaces, possibly with human intervention [8].

A sample application scenario may consist in four Web services: `BookShop WS`, `Publisher WS`, `Warehouse WS` and `Shipping WS`, supporting a book e-commerce process. In this scenario, a customer asks for a book invoking the `BookShop WS`, and submitting the book title. Each customer has a profile (custInf) which is used during Web service contracting and provisioning. The customer's profile collects various information about the customer, such as age, education, profession, interests, and book preferences, maximum budget. Such profile information is available to the `BookShop WS` using WS-enabled nodes to collect the customer's profile and to send it to the server [5]. We assume that the Web services in the application are *internally orchestrated*, while externally they are only *coordinated through the choreography.*

The goal of the set of interacting services is to send the right book to the right customer according to his preferences. In fact, a number of faults may occur during the execution of the process causing possible needs of substituting services and discovering new ones. For example, the wrong book is delivered [3], or a book is indefinitely reserved for a user who will not buy it, an order is indefinitely delayed, or one or more of the involved services indefinitely runs waiting for a reply from possibly faulty services.

4 Self-healing platform

On one hand, we aim at realizing a platform to support a self-healing Web service execution where faulty services can be substituted by discovering compatible services. This means that Web services should be able to detect possible failures caused by the Web services layer and, consequently, to enact a recovery action transparently to the user standpoint. On the other hand, since during a Web application execution a Web service may invoke external (non self-healing) Web services, our upgraded MAIS-P platform should be able to detect failures even at application level, i.e., caused by the external Web service. In this case, according to the loosely-coupled philosophy underlying the Service Oriented Computing, we are not interested on fixing the failed Web service, but we try to react to such a failures possibly substituting the failed one.

Figure 2 shows the modules, embedded in the WS-Diamond nodes, that support our goals. The *Diagnoser* identifies fault events or receives fault event notifications. The identification task operates on the information coming from the infrastructure & middleware layer, to detect internal node failures, and on the exchanged messages, to detect failures on partner's nodes.
Our architecture provides four modules to handle recovery actions:

– *reallocation module*: it optimizes resource reallocation requests on the underlying hardware and software infrastructure; resource reallocation policies are proposed in [1];
– *substitution module*: this module, available in the MAIS-P platform, allows the selection of substitute services according to the characteristics of the service request context, such as services which are functionally similar [13];
– *wrapper generator module*: this module, available both in MAIS and VISPO platforms [13, 8], allows the completion of missing parameters during service invocation, or to convert parameters in different formats;
– *quality module*: the PoliQual system [6] provides functionalities to insert data blocks before Web service invocation or upon receipt of a message, to assess data quality on-line and to perform reactive and proactive repair actions.

We assume that faults have to be recovered one at a time, and that the *Recovery action selector* module, by accessing the Fault registry, invokes the corresponding fault repair action.

4.1 Fault Registry

A basic set of known service faults is classified and stored in a *Fault Registry* at three *system levels*, for which, examples are provided in Table 1:

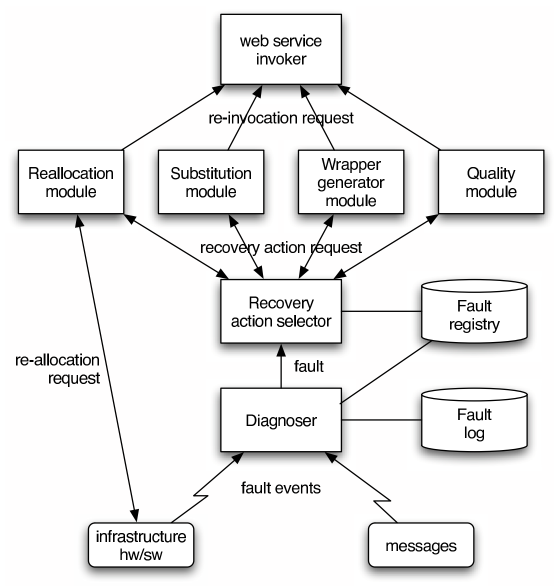


Fig. 2. Self-healing platform architecture

- the *Infrastructure & Middleware level* faults (not reported in the table) are due to failures in the underlying hardware, network, and system software infrastructure;
- the *Web service level* faults are due to failures in service invocation and service orchestration; these are captured by the WS-Diamond modules;
- the *Web application level* faults are malfunctions in the execution of Web applications due to data mismatches or coordination (choreography) failures. Also these faults are captured by the WS-Diamond modules.

Repair actions are designed according to the fault level and originate different recovery strategies according to the system components affected by the fault. For example, at the Web application level, the main goal is to provide: (i) services which respect the user requirements in terms of functionalities and QoS, (ii) business continuity, and (iii) fault masking. At the Web service level, the goal is to manage the service choreography correctly and to guarantee service continuity and QoS requirements, by substituting corrupted services with compatible ones available in the network.

4.2 Web service level faults

Faults at the Web service level are about the execution of a (set of) Web service(s). In particular, *Web service execution faults* raise either during service invocation or during execution of a simple Web service, while *Web Service coordination faults* result from composed Web services.

Level	Fault type	Examples
Web-application	Internal data faults	Data quality faults (value mismatch, e.g., inaccurate data in input parameter; missing data: null values).
	Application coordination faults	Application Failure due to reply timeout, resources not available at right time (Phase fault).
	Actor faults	Customer is not connected when a synchronous communication is needed.
	QoS violation faults	QoS value beyond threshold.
Web service	Web service execution faults	Missing parts in input message, wrong order of operation invocations (internal to a service).
	Web service coordination faults	Component service unavailable, process failure (time out).

Table 1. Fault Types

A *service execution* fault is raised during invocation if a service is not responding, or due to a wrong authorization of the end user, or to a parameter missing in the input SOAP message. A mismatch in the structure of messages to invoke a service may be due, for instance, to an update in the published service interfaces, which are not yet considered inside the invoking applications.
In our adaptive framework, a Web service execution fault may also occur when, upon substitution of a faulty Web service with a functionally equivalent one, no substitute is available. In that case, a discovery of a compatible service is needed in order to possibly substitute the faulty service. Discovery is performed using additional data about services, which are registered as elements of one or more *compatibility classes* [8], where each class identifies the required functionalities for the execution of a process activity. The rationale is that a provider associates a compatibility class with the service he is registering, if he thinks the service is able to satisfy the activity requirements. These requirements are stated when the class is firstly created and have to be satisfied by all class members. During the registration, compliance of the registered service to its compatibility classes is also evaluated and mapping information for semi-automatically building wrappers to adapt services to the process is generated.

A *service coordination fault* is typically due to a violation of the order of invocation of service operations or messages (e.g., a book payment is received before the corresponding book reservation). Coordination faults may occur when some of the Web services in a composed service are unavailable, or when a message is received that does not match the choreography protocol. Again, a phase of service discovery based on compatibility classes for substitution is needed.

4.3 Application level faults

Application level faults are related to the execution of a Web application. The mechanism used to detect a faulty service invocation is the *timeout*. If the result of an operation is not received by the due time, the application argues that some of the invoked services are not working properly.

Examples of application faults in the bookshop example are: unavailable goods, wrong book identification data, or session faults.

Fault at this level belong to the following categories: 1) Application Coordination faults; 2) Actor faults; 3) QoS violation faults; 4) Internal Data faults. Examples of *Application Coordination faults* in the bookshop example are:

– A session fault: e.g., loss of an HTTP session.
– A phase time fault: e.g., the book has been paid for and hence the Payment phase of the application has completed, but the confirmation of payment is received after an internal time out.
– A resource booking fault: e.g., not the whole resource pool necessary to complete the service has been reserved, for example the shipping system has not been reserved in advance.
– Inter-process faults: e.g., service data regarding the customer address or customer credit card have not been received in the correct sequence or at the right time.

An example of *Actor fault* is an authorization fault, e.g., the customer has entered a wrong password three times. *QoS violation faults* are related to local and global constraints specified by the user or by the application designer. For example, the user can require that the delivery time of the ordered book is lower than a given threshold (e.g., five days) or that the total price of the ordered book is lower than a given amount (e.g. 20$). Another category of QoS faults is bound to the *process design*, that is, to the way the application workflow has been developed. For example, an Unavailable goods fault, or a Payment failure fault should be treated by *exception handlers* specifically included in the process workflow by the designer. If some handlers have not been designed, the application could experience a deadlock or a total crash. Being the system self-healing, we expect the fault log to gather information useful to subsequently design the necessary handler.
Internal Data faults include data quality faults related to data manipulated during the execution of a service. Examples are the wrong title of an ordered book, or mismatched customer data. These faults may be regarded as QoS faults, but, since they specifically regard data internal to a service, they are evidenced as possibly bounded to specific filters and options to be treated apart by specific recovery actions.

5 Self-Healing Web services

The architecture shown in Figure 2, currently under development, connects the schema of the fault types with the schema of the recovery actions. Namely, the fault registry will be accessed by the diagnosys module in order to blame the faulty Web service. Once that the fault type has been identified, the Recovery Action Selector accesses the recovery actions table (see Table 2) to select the proper repair action that has to be undertaken. This selection is performed at runtime, taking into account context information (e.g., the network status or the workflow execution status). Hence, the same fault type can be handled by different recevery action modules, according to the particular context in which the fault has occurred.

Recovery action type	**Actions**	**Fault type**	**Type**
Service-oriented recovery action	Retry	Infrastructural, WS execution	Reactive
	Substitute	WS execution, WS coordination	Reactive
	Completion of missing message parts	WS execution	Reactive
	Reallocate	QoS	Reactive/proactive
	Change process structure	All	Proactive
	Process-oriented methods	Application level	Proactive
Data quality recovery actions	Insert data quality block	Internal data	Reactive
	Process-oriented methods	Application level	Proactive

Table 2. Recovery actions

This Section presents the set of recovery actions that can be employed to recover from failures. In particular, we focus on the management of Web application and Web service level faults. With respect to the way in which these actions are performed, two types of recovery actions are identified: *reactive recovery actions* and *proactive recovery actions*.

Reactive recovery actions are performed along with the execution of Web services and try/allow the recovery of running services. *Proactive recovery actions* are mostly based on data mining techniques and can mainly be executed in an off-line mode; proactive recovery actions are complex and require the support of an environment able to execute services, to detect runtime faults, and to perform recovery actions with no damage to the running instances of the monitored Web services. A long term approach to self-healing is adopted, where recovery actions have the goal of improving Web services and Web applications in order to avoid future failures. These actions can be oriented to provide a one-shot improvement action or to modify the service provisioning process for a permanent data and process improvement.

Recovery actions can be also classified in *service-oriented recovery actions* and *data quality recovery actions*. While the former deals with invocation, orchestration and choreography aspects of Web services, the latter pertains mainly to the management of data quality faults. For each fault type, several candidate recovery actions may be proposed, depending on the fault type and whether a reactive or proactive approach is taken, as synthesized in Table 2.

In the bookshop example, assume the invocation of the *sendBook* operation on the `Publisher WS` fails. The Diagnoser detects that the `Publisher WS` is experiencing a quality degradation that slows down its execution so that it cannot be used by other Web services anylonger. Consequently, the Diagnoser stores the detected fault in the fault log and invokes the Recovery Action Selector module. This module repairs the fault using the Substitution module to search for a compatible Web service to substitute the `Publisher WS`. The workflow execution is then restarted by invoking the *sendBook* operation on the substitute Web service. If the interface of the substitute service is different from the interface of the original service, the invocation is mediated by the Wrapper generator module.

Meanwhile, since the Diagnoser has detected that the quality of service degradation was due to a server crash at the publisher site, the WS-Diamond architecture proactively activates also the Reallocation module. Based on the analysis of the fault logs, the Reallocation module reallocates all the Web services that were running on the crashed server.

5.1 Service-oriented recovery actions

The techniques employed to realize this kind of recovery strictly depend on the model used to describe Web services. Some models describe how Web services act internally (i.e., orchestration), while other models only describe how different Web services collaborate (i.e., choreography). While both choreography and orchestration are exploited to detect faults, reactive recovery actions only rely on the orchestration model of the service. With our architecture, if a Web service has an internal behavior specified using a WS-BPEL process that composes other services, we are able to perform recovery actions over the Web service controlling the execution of its internal process. Under our approach, the recovery actions that can be applied over a WS-BPEL process in a MAIS-enabled node are the following: i) *retry* the invocation of a failed Web service, ii) *substitute* a failed Web service, iii) *reallocate* a failed Web service, and iv) *change* the structure of the process.

Retry Web services invocation
This recovery action is applied when faults point out a temporary unavailability of one or more services that compose the internal process of the analyzed Web service. In this case, the solution is to suspend the execution of the process and retry the invocation of the unavailable services until they return available. This solution is quite simple and does not require any sophisticated methodologies to manage the service invocation.

Substitute Web services
A more complicated situation is the case where one or more services are considered as definitely unavailable and, in order to complete the process execution, it is necessary to substitute each failed service.

It is possible to overcome this problem using the MAIS-P architecture that supports Web service compatibility evaluation and Web service substitution.

The compatibility evaluation between two Web services is performed comparing their functional interfaces (i.e., WSDL documents) and their provided QoS. If two services are defined as compatible, MAIS-P is able to automatically create a *service wrapper*, starting from their WSDL descriptions. Once that the wrapper is created, service substitution can be easily performed. The process structure is not modified, and the WS-BPEL orchestration engine continues the execution of the process without considering the service substitution. The management of the substitution is left to the MAIS-P architecture which exploits the wrapper to translate the parameters sent by the orchestrator into the parameters accepted by the substitute service and vice-versa.

Completion of missing parameters
Service invocation may fail when some of the input message parts are missing. Possible recovery actions may be based on knowledge of the role of parameters. In [8], we propose a technique to dynamically evaluate message composition of invoked Web service operations and look for missing information when parameters are necessary for message execution, while optional parts are ignored. The technique is based on an adaptive service invocation infrastructure.

Reallocate Web services
This type of recovery action is very useful for the particular subset of QoS violation faults that derives from a lack of hardware or software resources on the service provider side. In this situation, reallocating and executing the service on different machines or application servers can solve the problem. Reallocation is possible only if Web services are provided with an ad-hoc management interface and the recovery manager has free access to all the resources (e.g., the recovery manager can determine the load balancing or the application priority in the operating system).
Reallocation may be performed as reactive actions, when QoS violations are detected, but also as proactive actions, when optimization of service execution plans is performed using predictive techniques on future states of the execution environment.

Change the process structure
When service substitution or service reallocation are not enough for resuming a failed process, another recovery action that can be applied consists in modifying the structure of the process itself. According to [7], data mining techniques can be exploited in order to proactively identify possible anomalous situations during process execution. Workflow systems, in fact, produce a large quantity of log information which states how the old processes have been executed and how the current process is going on. In this case, the process designer, possibly supported by a tool, can manually fix or modify the process in order to delete erroneous activities. Once that the process is modified, it can be resumed or re-executed and previously described recovery actions can be employed.

5.2 Data quality recovery actions

Run time recovery actions in data quality require the identification of the causes of data errors and their permanent elimination through an observation of the whole process where data are involved. As an example, we refer to the Information Product Map (IP-MAP) methodology [14] which graphically describes the process by which the information product is manufactured. Error detection and the correction can be performed using different methods:

– *Data cleaning by manual identification*: comparison between value stored in the database and value in the real world;
– *Data bashing (or Multiple sources identification)*: comparison of the values stored in different databases;
– *Cleaning using Data edits*: automatic procedures that verify that data representation satisfies specific requirements.

6 Concluding Remarks

We have presented our approach for fault management of Web applications based on self-healing systems. A reference architecture for faults treatment and a classification of faults have been given, together with a set of strategies for recovery. Fault occurring during Web service and Web application execution have been studied and discussed within a proposed architecture where faults detection and interpretation for repair requiring search of substitutive services have been presented. Mutual dependencies among faults originated at these different system abstraction levels are a relevant issue to be further investigated in our research. Moreover, the distribution vs. central coordination of fault events detection and repair action execution are an important aspect to be studied for the proposed architecture.

Future work will focus on careful design of repair strategies, and on the evaluation of the proposed approach in the Project testbed environment.

Acknowledgments
Part of this work has been supported by EU Commission within the FET-STREP Project WS-Diamond.

References

1. D. Ardagna, M. Trubian, and L. Zhang. SLA Based Profit Optimization in Multi-tier Systems. In *NCA 2005 Proc.*, 2005.
2. L. Ardissono, L. Console, A. Goy, G. Petrone, C. Picardi, M. Segnan, and D. T. Dupre: Advanced Fault Analysis in Web Service Composition. In *International WWW Conference, poster session*, pages 1090–1091, 2005.
3. L. Ardissono, L. Console, A. Goy, G. Petrone, C. Picardi, M. Segnan, and D. T. Dupre: Enhancing Web Services with Diagnostic Capabilities. In *ECOWS05 Proc.*, 2005.
4. D. Bianchini, V. De Antonellis, B. Pernici, and P. Plebani. Ontology-based methodology for e-service discovery. *Accepted for publication on Journal of Information Systems, Special Issue on Semantic Web and Web Services*, 2004.
5. C. Cappiello, M. Comuzzi, E. Mussi, and B. Pernici. Context Management for Adaptive Information Systems. In *International Workshop on Context for Web Services (CWS-05).* Elsevier, 2005.
6. C. Cappiello, C. Francalanci, and B. Pernici. A self-monitoring system to satisfy data quality requirements. In *Proc. ODBase05*, 2005.
7. M. Castellanos, F. Casati, M.-C. Shan, and U. Dayal. iBOM: A Platform for Intelligent Business Operation Management. In *Proc. of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, 2005.
8. V. De Antonellis, M. Melchiori, L. D. Santis, M. Mecella, E. Mussi, B. Pernici, and P. Plebani. A Layered Architecture for Flexible Web Service Invocation. *Software: Practice and Experience*, 36(2):191–223, February 2006.
9. E. Esfandiari and V. Tosic. Towards a Web Service Composition Management Framework. In *In ICWS05 Proc.*, 2005.
10. R. Hamadi and B. Benatallah. Recovery Nets: Towards Self-Adaptive Workflow Systems. In *WISE*, pages 439–453, 2004.
11. P. Koopman. Elements of the Self-Healing System Problem Space. In *ICSE WADS03 Proc.*, 2003.
12. MAIS Web Site. http://www.mais.project.it.
13. B. Pernici, editor. *Mobile Information Systems. Infrastructure and Design for Adaptivity and Flexibility.* Springer, April 2006.
14. G. Shankaranarayan, R. Y. Wang, and M. Ziad. Modeling the Manufacture of an Information Product with IP-MAP. In *Proceedings of the 6th International Conference on Information Quality*, 2000.
15. G. Wang, C. Wang, A. Chen, H. Wang, C. Fung, S. Uczekaj, Y. L. Chen, W. Guthemiller, and J. Lee. Service Level Management using QoS Monitoring, Diagnostic, and Adaptation for Networked Enterprise Systems. In *EDOC 2005 Proc.*, pages 239–248, 2005.
16. D. S. Wile and A. Egyed. An Externalized Infrastructure for Self-Healing Systems. In *Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, 2004.
17. Y. Yan, M. Cordier, Y. Pencolé, and A. Grastien. Monitoring Web Service Networks in a Model-based Approach. In *ECOWS05 Proc.*, 2005.

A Non-Monotonic Approach to Semantic Matchmaking and Request Refinement in E-Marketplaces

Tommaso Di Noia[1], Eugenio Di Sciascio[1], Francesco M. Donini[2]

[1] Politecnico di Bari, Via Re David, 200, I-70125, Bari, Italy
{t.dinoia,disciascio}@poliba.it
[2] Università della Tuscia, via San Carlo, 32, I-01100, Viterbo, Italy
donini@unitus.it

Abstract. We present and motivate a formal approach and algorithms for semantic matchmaking in an e-commerce framework. The proposed solution exploits nonmonotonic inferences to compute semantic-based ranking of offers and provides explanation services in the query-retrieval-refinement loop.

1 Introduction

Matchmaking is basically the process of computing an ordered list of appealing resources with respect to a given request. Semantic matchmaking can be hence described as the process of computing an ordered list of appealing resource taking into account also the semantics of resources annotation and request definition, given with reference to an ontology.

Studies on matchmaking go a long way back[3]. Recently, semantic matchmaking has been investigated, in particular in the framework of Description Logics (DLs) reasoning. Usually, proposed approaches have been aimed –although with notable exceptions, see *e.g.*, [1, 8, 13, 6]– to the exploitation of standard reasoning services (*i.e.*, subsumption and satisfiability)for classification of matches into coarse categories, without providing a way to semantically rank obtained matches, neither within the same match class nor regardless of the class [9, 11, 17, 16, 18, 14, 4, 3]. Yet ranking should be a core process in matchmaking, and a classification into satisfiable/unsatisfiable might be unfair when unsatisfiability is due, for example, to a neglectable detail that would nevertheless prevent the evaluation of that match.

In this paper we show how to exploit non-monotonic inferences (namely, abduction and contraction) in a semantic-matchmaking process for ranking the available supplies descriptions. Basically, if a demand is incompatible with the supplies advertised, it can be amended (via contraction) in order to become compatible with them –and the more the amendments needed, the less is the

[3] For a detailed report on general matchmaking issues and systems we refer the interested reader to [8].

degree of match. If a demand is compatible with a supply, but does not subsume it, then it is possible to "hypothesize" (via abduction) some extra features in the supply so that it gets subsumed by the demand –and the more we have to hypothesize, the less is the degree of match. This process results not only in a list of offers that match the user's request, semantically ordered w.r.t. the degree of their match, but also in the amendments of the request and the offer that serve as an explanation why such a degree of match was assigned, which could be later on used to refine/revise the demand.

The remaining of this paper is structured as follows: in the next section, with the aid of simple propositional logic, we illustrate and motivate the rationale of our approach; we move on to more expressive DLs in Section 3 and present our theoretical approach together with algorithms for semantic matchmaking and query refinement in DLs. Last section draws the conclusions.

2 Logic-based Matchmaking Principles

Matchmaking is a widely used term in a variety of frameworks, comprising several –quite different– approaches. We first provide a generic and sound definition of matchmaking.

Definition 1 (Matchmaking). *Matchmaking is an information retrieval task whereby queries (a.k.a. demands) and resources (a.k.a. supplies) are expressed using semi-structured text in the form of advertisements, and task results are ordered (ranked) lists of those resources best fulfilling the query.*

This simple definition implies that –differently from classical unstructured-text information retrieval approaches– some structure in the advertisements is expected in a matchmaking system, and matchmaking does not consider a fixed database-oriented structure. Furthermore, usually database systems provide answers to queries that do not include a relevance ranking, which should be instead considered in a matchmaking process.

Definition 2 (Semantic Matchmaking). *Semantic matchmaking is a matchmaking task whereby queries and resources advertisements are expressed with reference to a shared specification of a conceptualisation for the knowledge domain at hand,* i.e.*, an ontology.*

From now on, we concentrate on matchmaking in marketplaces, adopting specific terminology, to ease presentation of the approach. Nevertheless our approach applies to generic matchmaking of semantically annotated resources.

2.1 Foundation of Logical Matchmaking

Our research is inspired by the application of Description Logics to E-Commerce. However, we believe that foundations of a logical approach to matchmaking are more evident in a simple propositional setting first, then we will move on to DLs.

As usual, an interpretation assigns values true, false to elements of a propositional ontology $\mathcal{T}$, and truth values are assigned to formulae according to usual truth tables for $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$. A set of formulae is *consistent* if there is at least one interpretation assigning true to all formulae.

Satisfiability and classification w.r.t. an ontology $\mathcal{T}$ can be formally explained in terms of interpretations for a logical theory. In fact if we reduce an ontology to its logical axioms, then we can formalize both classification and satisfiability in terms of logical interpretations of a theory $\mathcal{T}$.

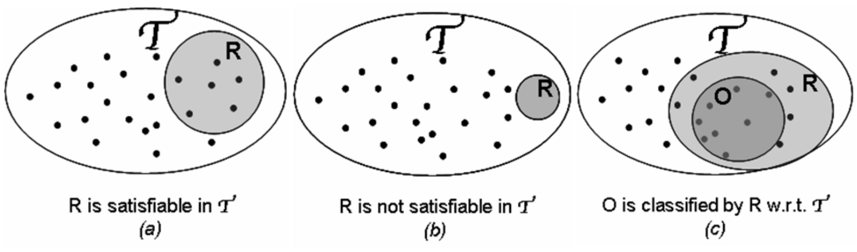


Fig. 1. Satisfiability and Classification

A request R (conversely a resource O) is satisfiable w.r.t. $\mathcal{T}$ if there is at least one interpretation in all the interpretations for $\mathcal{T}$ which is also a interpretation for R (conversely for O) – see Figure 1(a-b). For what concerns classification, given R and O both satisfiable w.r.t. $\mathcal{T}$, we say that O is classified by R if all the interpretations for O are also interpretations for R see Figure 1(c).

Formally, let $\mathcal{M}$ be the interpretations for $\mathcal{T}$ and $\mathcal{M}_{\mathsf{R}}$ the interpretations in $\mathcal{M}$ that satisfy the request R (respectively $\mathcal{M}_{\mathsf{O}}$ for the resource O). We have R (conversely O) is satisfiable if $\mathcal{M}_{\mathsf{R}} \neq \emptyset$ and R classifies O if $\mathcal{M}_{\mathsf{O}} \subseteq \mathcal{M}_{\mathsf{R}}$.

Given R and O both satisfiable w.r.t. an ontology, logic based approaches to matchmaking proposed in the literature [16, 14, 8] use classification and satisfiability to grade match results in five categories:

1. **Exact.** Figure 2(a). $\mathcal{M}_{\mathsf{R}} = \mathcal{M}_{\mathsf{O}}$ – All the interpretations for R are also interpretations for O – in formulae $\mathcal{T} \models \mathsf{R} \Leftrightarrow \mathsf{O}$.

2. **Full - Subsumption.** Figure 2(b). $\mathcal{M}_{\mathsf{O}} \subseteq \mathcal{M}_{\mathsf{R}}$ – All the interpretations for O are also interpretations for R – in formulae $\mathcal{T} \models \mathsf{O} \Rightarrow \mathsf{R}$.

3. **Plug-In.** Figure 2(c). $\mathcal{M}_{\mathsf{R}} \subseteq \mathcal{M}_{\mathsf{O}}$ – All the interpretations for R are also interpretations for O – in formulae $\mathcal{T} \models \mathsf{R} \Rightarrow \mathsf{O}$.

4. **Potential - Intersection.** Figure 2(d). $\mathcal{M}_{\mathsf{R}} \cap \mathcal{M}_{\mathsf{O}} \neq \emptyset$ – Some interpretations for R are also interpretations for O – In formulae $\mathcal{T} \not\models \neg(\mathsf{R} \wedge \mathsf{O})$.

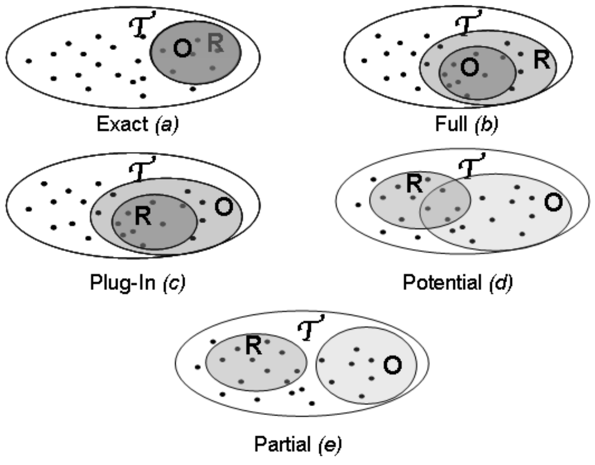


Fig. 2. Match Types

5. Partial - Disjoint. Figure 2(e). $\mathcal{M}_{\mathsf{R}} \cap \mathcal{M}_{\mathsf{O}} = \emptyset$ – No interpretations for R are also interpretations for O – In formulae $\mathcal{T} \models \neg(\mathsf{R} \wedge \mathsf{O})$.

In a marketplace framework, the aim of a matchmaking process is to satisfy a user request as far as possible, so the best match type is obviously the first one, *i.e.*, **Exact**, because both R and O express the same preferences and then, since all the resource characteristics requested in R are semantically implied by O and vice versa, the user finds exactly what she is looking for. In a **Full** match all the interpretations for O are surely also interpretations for R and then O completely satisfies R, this means that all the all the resource characteristics requested in R are semantically implied by O but not, in general, vice versa. Then, in a **Full** match, O may expose some unrequested characteristics. **Plug-In** match expresses the situation when O is more generic than R and then it is possible that the latter can be satisfied by the former. Some characteristics in R are not specified, implicitly or explicitly, in O which is more generic than R. With **Potential** match we can only say that a similarity degree exists between O and R and then O might potentially satisfy R, as an Open World Assumption implies that what is not specified has not to be interpreted as a constraint of absence. Finally **Partial** match states that there is no common interpretation between R and O. We prefer to name this match type **Partial** instead of ***Disjoint*** [14] or ***Nonmatch*** [12] because, as in the example below, this situation should not be inevitably considered an unrecoverable match.

The following example shows, in a propositional logic setting, the rationale of such categorization. Suppose to have the following set $\mathcal{T}$ of propositional axioms modeling an automotive domain:

$$\mathcal{T} = \begin{cases} \texttt{ReflexSilver} \Leftrightarrow \texttt{Silver} \wedge \texttt{Metallic} \\ \texttt{PassengerAirbag} \Rightarrow \texttt{Airbag} \\ \texttt{EcoDiesel} \Rightarrow \texttt{Diesel} \\ \texttt{Gasoline} \Leftrightarrow \neg\texttt{Diesel} \\ \texttt{SatelliteAlarm} \Rightarrow \texttt{AlarmSystem} \end{cases}$$

Now imagine to have a request R and five offers O_{par}, O_{pot}, O_{full}, O_{plug}, O_{ex} as in the following:

$$\begin{aligned}
\mathbf{R} &= \texttt{Sedan} \wedge \texttt{PassengerAirbag} \wedge \texttt{EcoDiesel} \wedge \texttt{ReflexSilver} \wedge \texttt{AlarmSystem} \\
\mathbf{O}_{par} &= \texttt{Sedan} \wedge \texttt{Gasoline} \wedge \texttt{AlarmSystem} \wedge \texttt{PassengerAirbag} \wedge \texttt{ReflexSilver} \\
\mathbf{O}_{pot} &= \texttt{Sedan} \wedge \texttt{Airbag} \wedge \texttt{Silver} \wedge \texttt{SatelliteAlarm} \wedge \texttt{EcoDiesel} \\
\mathbf{O}_{full} &= \texttt{Sedan}\wedge\texttt{PassengerAirbag}\wedge\texttt{EcoDiesel}\wedge\texttt{ReflexSilver}\wedge\texttt{SatelliteAlarm} \\
\mathbf{O}_{plug} &= \texttt{Sedan} \wedge \texttt{PassengerAirbag} \wedge \texttt{Diesel} \wedge \texttt{ReflexSilver} \wedge \texttt{AlarmSystem} \\
\mathbf{O}_{ex} &= \texttt{Sedan}\wedge\texttt{PassengerAirbag}\wedge\texttt{EcoDiesel}\wedge\texttt{Metallic}\wedge\texttt{Silver}\wedge\texttt{AlarmSystem}
\end{aligned}$$

In Table 1the interpretations are shown of $\mathcal{T}$satisfying R, O_{par}, O_{pot}, O_{full}, O_{plug} and O_{ex}.

$\mathcal{M}_{\mathsf{R}}$

Sedan	PassengerAirbag	EcoDiesel	ReflexSilver	Gasoline	Airbag	Silver	Metallic	Diesel	AlarmSystem	SatelliteAlarm
T	T	T	T	F	T	T	T	T	T	T
T	T	T	T	F	T	T	T	T	T	F

$\mathcal{M}_{\mathsf{O}_{par}}$

Sedan	PassengerAirbag	EcoDiesel	ReflexSilver	Gasoline	Airbag	Silver	Metallic	Diesel	AlarmSystem	SatelliteAlarm
T	T	F	T	T	T	T	T	F	T	T
T	T	F	T	T	T	T	T	F	T	F

Partial Match: $\mathcal{M} \cap \mathcal{M}_{\mathsf{O}_{par}} = \emptyset$

$\mathcal{M}_{\mathsf{O}_{pot}}$

Sedan	PassengerAirbag	EcoDiesel	ReflexSilver	Gasoline	Airbag	Silver	Metallic	Diesel	AlarmSystem	SatelliteAlarm
T	T	T	T	F	T	T	T	T	T	T
T	T	T	F	F	T	T	F	T	T	T
T	F	T	T	F	T	T	T	T	T	T
T	F	T	F	F	T	T	F	T	T	T

Potential Match: $\mathcal{M}_{\mathsf{R}} \cap \mathcal{M}_{\mathsf{O}_{pot}} \neq \emptyset$

$\mathcal{M}_{\mathsf{O}_{full}}$

Sedan	PassengerAirbag	EcoDiesel	ReflexSilver	Gasoline	Airbag	Silver	Metallic	Diesel	AlarmSystem	SatelliteAlarm
T	T	T	T	F	T	T	T	T	T	T

Full Match: $\mathcal{M}_{\mathsf{O}_{full}} \subseteq \mathcal{M}_{\mathsf{R}}$

$\mathcal{M}_{\mathsf{O}_{plug}}$

Sedan	PassengerAirbag	EcoDiesel	ReflexSilver	Gasoline	Airbag	Silver	Metallic	Diesel	AlarmSystem	SatelliteAlarm
T	T	T	T	F	T	T	T	T	T	T
T	T	T	T	F	T	T	T	T	T	F
T	T	F	T	F	T	T	T	T	T	T
T	T	F	T	F	T	T	T	T	T	F

Plug-In Match: $\mathcal{M}_{\mathsf{R}} \subseteq \mathcal{M}_{\mathsf{O}_{plug}}$

$\mathcal{M}_{\mathsf{O}_{ex}}$

Sedan	PassengerAirbag	EcoDiesel	ReflexSilver	Gasoline	Airbag	Silver	Metallic	Diesel	AlarmSystem	SatelliteAlarm
T	T	T	T	F	T	T	T	T	T	T
T	T	T	T	F	T	T	T	T	T	F

Exact Match: $\mathcal{M}_{\mathsf{R}} = \mathcal{M}_{\mathsf{O}_{ex}}$

Table 1. Matches

Actually, it is questionable whether a Plug-In match type should be considered better than Potential one in a marketplace scenario. We note that some researchers also consider Plug-In match more favorable than full match (*e.g.*, see [14, 16]), motivating this choice with the idea that if $\mathcal{M}_{\mathsf{R}} \subseteq \mathcal{M}_{\mathsf{O}}$ one may expect that the advertiser offering resource O will probably have also more specific resources; in an e-commerce setting if the advertiser offers a *sedan* car it will also probably offer specific types of *sedans*. Nevertheless we argue that this idea prevents a fully automated matchmaking, which is possible when $\mathcal{M}_{\mathsf{O}} \subseteq \mathcal{M}_{\mathsf{R}}$, and furthermore it favors underspecified resource description, *i.e.*, an advertisement offering a sedan will always Plug-In match any request for a specific sedan, but will leave on the requester the burden to determine the right one – if any is actually available – for his/her needs. Even though Exact match is surely the best match, Full match might be considered –not always anyway– equivalent from the requester point of view, because it states that at least all the features specified in R are also expressed in O.

Usually, logic-based approaches only allow, as illustrated above, a categorization within match types. But while exact and full matches can be rare (and basically equivalent), a user may get several potential and partial matches. Then a useful logic-based matchmaker should provide a –logic– ordering of available resources vs. the request, but what we get using classification and satisfiability is a boolean answer. Also partial matches, as pointed out in [8], might be just "near miss", *e.g.*, maybe just one requirement is in conflict, but a pure satisfiability check returns a hopeless *false* result, while it could be interesting to order "not so bad" offers according to their similarity to the request.

One may be tempted to revert to classical and well-assessed information retrieval (IR) algorithms to get a rank for approximate matches (*e.g.*, so-called hybrid approaches [13]), but regardless of well-known limits of unstructured text retrieval, there is something IR algorithms cannot do, while a logic approach can: provide explanations for matches result and suggest revision of requests.

2.2 Penalty Functions

Matches classification based on implication and satisfiability is still a coarse one, relying directly on known logical relations between formulae. In fact, the result of matchmaking should be a *rank* of resources/supplies/counteroffers, according to some criteria – possibly explicit – so that a user trusting the system would know whom to contact first, and in case of failure, whom next, and so on. Such a ranking process should satisfy some criteria that a Knowledge Representation approach suggests. We formulate ranking requirements by referring to properties of penalty functions $p(\mathsf{O}, \mathsf{R}, \mathcal{T})$, O, R being two formulae and $\mathcal{T}$ an ontology.

$$p : \langle \mathsf{O}, \mathsf{R}, \mathcal{T} \rangle \rightarrow \Re$$

We use penalty functions to rank O for a given R w.r.t. a ontology $\mathcal{T}$. Intuitively, for two given $\mathsf{O}_1, \mathsf{O}_2$ in the marketplace, if $p(\mathsf{O}_1, \mathsf{R}, \mathcal{T}) < p(\mathsf{O}_2, \mathsf{R}, \mathcal{T})$ then the issuer of demand R should rank O_1 better than O_2 when deciding whom to contact first. Clearly, a 0-penalty should be ranked best, and counteroffers with the same penalties should be ranked breaking ties.

The first property we recall is Non-symmetric evaluation of proposals.

Definition 3. *A penalty function $p(\cdot, \cdot, \cdot)$ is non-symmetric if there exists formulae R, O and an ontology $\mathcal{T}$ such that $p(O, R, \mathcal{T}) \neq p(R, O, \mathcal{T})$.*

This property is evident when all constraints of R are fulfilled by O but not vice versa. Hence, O should be among the top-ranked counteroffers in the list of potential partners of R, while R should *not* necessarily appear at the top in the list of potential partners of O.

Secondly, if logic is used to give some meaning to descriptions of supplies and demands, then proposals with the same meaning should be equally penalized, independently of their syntactic descriptions.

Definition 4. *A penalty function $p(\cdot,\cdot,\cdot)$ is* syntax independent *if for every triple of formulae O_1, O_2, R, and ontology $\mathcal{T}$, when $\mathcal{T} \models O_1 \Leftrightarrow O_2$ then $p(O_1, R, \mathcal{T}) = p(O_2, R, \mathcal{T})$, and the same holds also for the second argument ,* i.e., $p(R, O_1, \mathcal{T}) = p(R, O_2, \mathcal{T})$

Clearly, when the logic admits a normal form of expressions — as CNF or DNF for propositional logic, or the normal form of concepts for some DLs — using such a normal form in the computation of $p(\cdot,\cdot,\cdot)$ ensures by itself syntax independence.

We now consider the relation between penalties and implication. We separately consider penalty functions for ranking potential matches — $p_{\Rightarrow}(\cdot,\cdot,\cdot)$ — from those for ranking partial (conflicting) matches — $p_{\emptyset}(\cdot,\cdot,\cdot)$.

Definition 5. *A penalty function for potential matches is* monotonic over implication *whenever for every issued demand R, for every pair of counteroffers O_1 and O_2, and ontology $\mathcal{T}$, if O_1 and O_2 are both potential matches for R w.r.t. $\mathcal{T}$, and $\mathcal{T} \models (O_1 \Rightarrow O_2)$, then $p_{\Rightarrow}(O_1, R, \mathcal{T}) \leq p_{\Rightarrow}(O_2, R, \mathcal{T})$*

Intuitively, the above definition could be read of as: if $\mathcal{T} \models (\mathsf{O}_1 \Rightarrow \mathsf{O}_2)$ then O_1 is more specific than O_2 and should be penalized (and then ranked) either the same, or better than O_2. *A ranking of potential matches is monotonic over implication if the more specific, the better.* In other words, O_1 exposes more characteristics than O_2 that can satisfy the ones requested by R. A dual property could be stated for the second argument: if $\mathcal{T} \models (\mathsf{R}_1 \Rightarrow \mathsf{R}_2)$ then a counteroffer O is less likely to fulfill all characteristics required by R_1 than R_2. However, since our scenario is: "given an issuer of a demand R looking for a match in the marketplace, rank all possible supplies $\mathsf{O}_1, \mathsf{O}_2, \ldots$, from the best one to the worst", we do not deal here with this duality between first and second argument of $p_{\Rightarrow}(\cdot,\cdot,\cdot)$.

When turning to partial matches, in which some properties are already in conflict between supply and demand, the picture reverses. Now, adding another characteristic to an unsatisfactory proposal may only worsen this ranking (when another characteristic is violated) or keep it the same (when the new characteristic is not in conflict).

Definition 6. *A penalty function for partial matches is* antimonotonic over implication *whenever for every issued demand R, for every pair of supplies O_1 and O_2, and ontology $\mathcal{T}$, if O_1 and O_2 are both partial matches for R w.r.t. $\mathcal{T}$, and $\mathcal{T} \models (O_1 \Rightarrow O_2)$, then $p_{\emptyset}(O_1, R, \mathcal{T}) \geq p_{\emptyset}(O_2, R, \mathcal{T})$*

If $\mathcal{T} \models (\mathsf{O}_1 \Rightarrow \mathsf{O}_2)$ then O_1 should be penalized (and then ranked) either the same, or worse than O_2. In fact, if O_1 and O_2 are both partial matches for R and O_1 is more specific than O_2, then there are more characteristic in O_1 that can conflict with the ones in R than in O_2. In other words, *A ranking of partial matches is antimonotonic over implication if the more specific, the worse.* The same property should hold also for the second argument, since conjunction is commutative.

2.3 From Partial to (quasi-)Exact Match

We illustrate the rationale of our approach by computing what is needed in order to "climb the classification list" and reach a **Full** or an **Exact** match.

In particular, if we get a Partial match we could revise R relaxing some restrictions, in order to reach a Potential match. Once we get a Potential match we can hypothesize what is not specified in O in order to reach a Full match and subsequently we can suggest to the user what is not specified in the relaxed R but is in O. The ideal sequence should be then:

Partial → **Potential** → **Full** (→ **Exact**)

With reference to our previous example we show how this process can be performed in a propositional framework. Let us define the following R and O we will use in the rest of the Section:

$$\mathbf{R} = \texttt{Sedan} \wedge \texttt{PassengerAirbag} \wedge \texttt{EcoDiesel} \wedge \texttt{ReflexSilver}$$
$$\mathbf{O} = \texttt{Sedan} \wedge \texttt{Gasoline} \wedge \texttt{AlarmSystem} \wedge \texttt{Silver}$$

It is easy to understand that due to the axiom $\texttt{Gasoline} \Leftrightarrow \neg\texttt{Diesel}$ in $\mathcal{T}$, $\mathcal{M}_{\mathsf{R}} \cap \mathcal{M}_{\mathsf{O}} = \emptyset$ and then a Partial match occurs.

In this example we can easily identify the source of inconsistency and notice it is due to the `EcoDiesel` specification in R and the `Gasoline` one in O. So if the requester relaxes R giving up `EcoDiesel` specification — we call this subformula G for **Give Up** — then we have a new contracted request. We call this new request K for **Keep**.

$$\mathsf{K} = \texttt{Sedan} \wedge \texttt{PassengerAirbag} \wedge \texttt{ReflexSilver}$$

The contracted part of R *i.e.*, K is now a Potential match with respect to O That is $\mathcal{M}_{\mathsf{K}} \cap \mathcal{M}_{\mathsf{O}} \neq \emptyset$.

Now to reach a Full match we should reduce $\mathcal{M}_{\mathsf{O}}$. This is possible hypothesizing some unspecified characteristic H (for Hypothesis) in O such that $\mathcal{M}_{\mathsf{O} \wedge \mathsf{H}} \subseteq \mathcal{M}_{\mathsf{R}}$. If we hypothesize $\mathsf{H} = \texttt{Metallic} \wedge \texttt{PassengerAirbag}$ the previous set relation holds. Then, having

$$\mathsf{O} \wedge \mathsf{H} = \texttt{Sedan} \wedge \texttt{Gasoline} \wedge \texttt{AlarmSystem} \wedge \texttt{Silver} \wedge \texttt{Metallic} \wedge \texttt{PassengerAirbag}$$

a Full match occurs *i.e.*, $\mathcal{M}_{\mathsf{O} \wedge \mathsf{H}} \subseteq \mathcal{M}_{\mathsf{K}}$.
The previous example shows that some revision and hypotheses are needed in order to perform an extended matchmaking process and move through match classes.

Partial → Potential. Contract R to K giving up elements G conflicting with O –Extend $\mathcal{M}_{\mathsf{R}}$ to $\mathcal{M}_{\mathsf{K}}$.
Clearly, this is a non-monotonic task, particularly it can be reduced to contraction in a belief revision framework[10]. If the match type we obtain is a Partial one, with respect to an ontology $\mathcal{T}$, the following relation holds: $\mathcal{T} \models \neg(\mathsf{R} \wedge \mathsf{O})$. Now consider $\mathcal{T}'$ as an ontology equivalent to $\mathcal{T}$ and such that $\mathcal{T}' = \mathcal{T} \cup \{\rho \Leftrightarrow \mathsf{R}, \omega \Leftrightarrow \mathsf{O}\}$, with ρ and ω being two new propositional symbols. The previous relation can be rewritten as $\mathcal{T}' \models \neg(\rho \wedge \omega)$. In order to reach a Potential match you can give up some information in the axiom involving R and consider the contracted ontology $\mathcal{T}_c' = \mathcal{T} \cup \{\rho \Leftrightarrow \mathsf{K}, \omega \Leftrightarrow \mathsf{O}\}$ such that $\mathcal{T}_c' \not\models \neg(\rho \wedge \omega)$.

Potential → Full. Hypothesize missing characteristics H in O in order to completely satisfy K–Reduce $\mathcal{M}_{\mathsf{O}}$.
Also this process can be modeled as a non-monotonic task. In particular, moving from a Potential match to a Full match, we exploit abductive reasoning. In fact, we are not able to explain (satisfy in all the models) K given the manifestation O w.r.t. the ontology $\mathcal{T}$, and we look for a set of possible hypotheses H such that (1)$\mathcal{T} \cup \mathsf{H} \cup \{\mathsf{O}\} \not\models \mathsf{false}$ and (2)$\mathcal{T} \cup \mathsf{H} \cup \{\mathsf{O}\} \models \mathsf{K}$, where the set of all possible hypotheses is, due to (1), a subset of the propositional alphabet used to model $\mathcal{T}$, R and O.

While performing contraction and abduction, minimality criteria have to be taken into account. If we are a demander:

– we wish to keep as much as possible of the original request. Trivially, if we contract all the request, we surely obtain a Potential match with the supply, but it should be obvious that this is not a good solution from the demander point of view.
– we wish to hypothesize as less as possible features in the supply in order to be satisfied. We could hypothesize all the characteristics we requested in the supply, and reach a Full match. Again this is a trivial solution.

Based on the previous observation, if we have more than one supply during the contraction phase the demander should be willing to choose the one with the minimum number of features to be contracted. During the abduction process the demander will prefer supplies with less requested features to be hypothesized in O.

Observe that we are not asking the requester to actually go through the whole process; the process outlined so far simply explains the rationale for computing penalty functions in terms of what is conflicting and what is missing when matching R and O $\dot{\text{Y}}$et our approach has a twofold advantage: the requester can use provided information to actually revise her request but the information we extract is also all what is needed to compute a penalty function to be used by a broker in a e-marketplace.

Once we know what has to be contracted and hypothesized it is possible to compute a match degree based on K and H, that is what is needed in order to reach a Full match. In case of multiple resources, we can use this match degree as a score to rank such resources with respect to R. Such a penalty function should be in the form:

$$\rho : \langle \mathsf{R}, \mathsf{O}, \mathsf{K}, \mathsf{H}, \mathcal{T} \rangle \rightarrow \Re$$

ρ combines all the causes for lack of a Full match. Notice that ρ needs also $\mathcal{T}$; in fact in $\mathcal{T}$ the semantics of K, H, R and O are modeled, which should be taken into account when evaluating how to weigh them with respect to O and R.

Before making the final step beyond, moving from a Full to an Exact match, some considerations are needed.

In an *Open World* semantics, what is not specified in a formula has not to be interpreted as a constraint of absence. It is a "don't care" specifications. In a resource retrieval scenario – as a marketplace is – this can be due to basically two reasons:

- the user really does not care about the unspecified information.
- the user does not own that knowledge. She is not aware that it is possible to specify some other characteristics in the request, or she simply did not consider further possible specifications. She is not necessarily an expert of the marketplace knowledge domain.

In the second case a further refinement makes sense. A way to do this is to present to the user all the knowledge modeled in $\mathcal{T}$and ask her to refine the query, adding characteristics found in $\mathcal{T}$. This approach has at least two main drawbacks:

1. The user must be bored browsing all $\mathcal{T}$in order to find something interesting to be added to the request.
2. She can choose something in $\mathcal{T}$which is not in any offer in the marketplace. Then after the query refinement she would not see any change in the list ranked using $\rho(\mathsf{R}, \mathsf{O}, \mathsf{K}, \mathsf{H}, \mathcal{T})$.

To avoid the above drawbacks, we might suggest to the requester only those characteristics able to change the ranked list of offers within the marketplace. Then (in an ideal marketplace where the only offer is O) we could suggest to the user to refine the contracted request adding $\mathsf{B}' = \mathtt{AlarmSystem} \wedge \mathtt{Gasoline}$, where B' (for Bonus) represents what is specified in O but is not in K. But notice that in B' we have `Gasoline`, that is the source of inconsistency of the original request R with the original O. Then it would be very strange if the user refined her request by adding something which is in conflict with her initial preferences. The user is likely to refine adding at most $\mathsf{B} = \mathtt{AlarmSystem}$.

Full → quasi-Exact. Suggest to the requester what should be added to K looking at non requested features B (for bonus) in O –Reduce $\mathcal{M}_{\mathsf{K}}$.

In order to propose only appealing features to the user, first of all it is necessary to remove from O the causes for inconsistency with K. This is clearly the contraction problem seen in action to move from Partial to Potential match with K and O flipped over. This time we want to contract O w.r.t. K. Once we have deleted the inconsistencies from O reducing it to K_{O}, we need to make some hypotheses on how the user could refine the query so that the (contracted) request completely overlap the contracted supply K_{O}. Again an abduction problem has to be solved in order cope with this problem. We have to hypothesize B such that $\mathcal{T} \models (\mathsf{K} \land \mathsf{B} \Rightarrow \mathsf{K}_{\mathsf{O}})$.

2.4 Penalty Functions and Non-Monotonic Tasks for Resources Ranking

In this section we will discuss relations between penalty functions introduced in Section 2.2 and the process of moving through match classes outlined in the previous Section.

A possible $p_{\emptyset}(\cdot,\cdot,\cdot)$, can measure how difficult it is to move from Partial to Potential match, or in other words how big is G with respect to the original R. Looking at the contraction process highlighted in case of Partial match, it is easy to show that it is a non-symmetric process. If we contract R w.r.t. O, we obtain a new request K which is, generally, different from the contracted version of O w.r.t. R. Then property 3 is satisfied. Surely, a contraction reasoning task is syntax independent and then also property 4 is satisfied. For what concerns property 6, intuitively, if we add a new feature to R (or equivalently O) and it is not in conflict with O (or R), the information to give up G remain the same; if the new feature we add to R (O) is in conflict with O (R), then we have to give up this new feature and increase G.

For $p_{\Rightarrow}(\cdot,\cdot,\cdot)$, we might define a function computing how far is a Potential match from a Full match, *i.e.*, a function measuring H. The abduction process to move from Potential to Full match is both non-symmetric w.r.t. R and O and syntax independent, so properties 5 and 6 are satisfied. The last property to be satisfied is the monotonicity of $p_{\Rightarrow}(\cdot,\cdot,\cdot)$. Again, in terms of abductive reasoning, adding to R (respectively O) a feature which is not implied by O (respectively R), H increases. If we add a feature which is implied by O (respectively by R), the hypotheses to be formulated are exactly the same and also H remains unchanged.

Based on the above consideration we can redefine ρ, computing an overall match degree of R w.r.t. O, as a function of $p_{\emptyset}(\cdot,\cdot,\cdot)$ and $p_{\Rightarrow}(\cdot,\cdot,\cdot)$.

$$\rho(\mathsf{R},\mathsf{O},\mathsf{K},\mathsf{H},\mathcal{T}) = \rho(p_{\emptyset}(\mathsf{O},\mathsf{R},\mathcal{T}), p_{\Rightarrow}(\mathsf{O},\mathsf{R},\mathcal{T}))$$

Actually, if R and O are a Potential match, only $p_{\Rightarrow}(\mathsf{O},\mathsf{R},\mathcal{T})$ contributes to the match degree computation. In this case $\rho(p_{\emptyset}(\mathsf{O},\mathsf{R},\mathcal{T}), p_{\Rightarrow}(\mathsf{O},\mathsf{R},\mathcal{T})) = p_{\Rightarrow}(\mathsf{O},\mathsf{R},\mathcal{T})$. We remark that modeling the matchmaking process as a sequence of non-monotonic reasoning tasks, and computing the match degree based on them, we are also able to provide explanations for the match degree. That is, when the system presents a ranked list of possible supplies to the requester, it is also able to justify the results – and then increase the trust level of the user in the system – but also guide the user in the refinement process.

3 DL Inference Services for Matchmaking

Although useful to model simplified examples, propositional logic is obviously limited for what concerns expressiveness. In the following we will refer to Description Logics (DL) whose formal semantics is the basis of the Ontology Web Language OWL-DL [15], and model a DL-based framework to cope with the issues introduced here. Please refer to [2] for a comprehensive survey on DLs.

DL-based systems usually provide at least two basic reasoning services:

1. *Concept Satisfiability*: $\mathcal{T} \models \mathsf{R} \not\sqsubseteq \bot$ –Given a TBox $\mathcal{T}$and a concept R, does there exist at least one model of $\mathcal{T}$assigning a non-empty extension to R?
2. *Subsumption*: $\mathcal{T} \models \mathsf{R} \sqsubseteq \mathsf{O}$ –Given a TBox $\mathcal{T}$and two concepts R and O, is R more general than O in any model of $\mathcal{T}$?

Matchmaking services outlined in the previous section call for other, non-monotonic inference services, we briefly recall hereafter.

Let us consider concepts O and R and an ontology $\mathcal{T}$. If a partial match occurs, *i.e.*, they are not compatible with each other w.r.t. $\mathcal{T}$, one may want to retract specifications in R, G (for *Give up*), to obtain a concept K (for *Keep*) such that $\mathsf{K} \sqcap \mathsf{O}$ is satisfiable in $\mathcal{T}$. In [5] the Concept Contraction problem was defined as follows:

Definition 7. *Let $\mathcal{L}$ be a DL, O, R, be two concepts in $\mathcal{L}$ and $\mathcal{T}$ be a set of axioms in $\mathcal{L}$, where both O and R are satisfiable in $\mathcal{T}$. A* Concept Contraction Problem *(CCP), identified by $\langle \mathcal{L}, O, R, \mathcal{T} \rangle$, is finding a pair of concepts $\langle G, K \rangle \in \mathcal{L} \times \mathcal{L}$ such that $\mathcal{T} \models R \equiv G \sqcap K$, and $K \sqcap O$ is satisfiable in $\mathcal{T}$. Then K is a* contraction *of R according to O and $\mathcal{T}$.*

If nothing can be kept in R during the contraction process, we get the worst solution — from a matchmaking point of view — $\langle \mathsf{G}, \mathsf{K} \rangle = \langle \mathsf{R}, \top \rangle$, that is give up everything of R. If $\mathsf{R} \sqcap \mathsf{O}$ is satisfiable in $\mathcal{T}$, that is a potential match occurs, nothing has to be given up and the solution is$\langle \top, \mathsf{R} \rangle$, that is, give up nothing. Hence, a Concept Contraction problem amounts to an extension of a satisfiable one. Since usually one wants to give up as few things as possible, some minimality criteria in the contraction must be defined [10]. In most cases a pure logic-based approach could be not sufficient to decide between which beliefs to give up and which to keep. There is the need to model and define some extra-logical information, which have to be taken into account. For instance, one could be interested in contracting only some specification in her request, while others have to be considered strict [6].

If the offered resource O and the request R are in a potential match, it is necessary to assess what should be hypothesized H in O in order to completely satisfy R and then move to a full match. In [7] the Concept Abduction problem was defined as follows:

Definition 8. *Let $\mathcal{L}$ be a DL, O, R, be two concepts in $\mathcal{L}$, and $\mathcal{T}$ be a set of axioms in $\mathcal{L}$, where both O and R are satisfiable in $\mathcal{T}$. A* Concept Abduction Problem *(CAP), identified by $\langle \mathcal{L}, R, O, \mathcal{T} \rangle$, is finding a concept $H \in \mathcal{L}$ such that $\mathcal{T} \models O \sqcap H \sqsubseteq R$, and moreover $O \sqcap H$ is satisfiable in $\mathcal{T}$. We call H a* hypothesis *about O according to R and $\mathcal{T}$.*

Obviously the definition refers to satisfiable O and R, since R unsatisfiable implies that the CAP has no solution at all, while O unsatisfiable leads to counterintuitive results ($\neg\mathsf{R}$ would be a solution in that case). If $\mathsf{O} \sqsubseteq \mathsf{R}$ then we have $\mathsf{H} = \top$ as a solution to the related CAP. Hence, Concept Abduction amounts to extending subsumption. On the other hand, if $\mathsf{O} \equiv \top$ then $\mathsf{H} \sqsubseteq \mathsf{R}$.

Concept Abduction and Concept Contraction can be used for respectively subsumption and satisfiability explanation. For Concept Contraction , having two concepts whose conjunction is unsatisfiable, in the solution $\langle \mathsf{G}, \mathsf{K} \rangle$ to the CCP $\langle \mathcal{L}, \mathsf{R}, \mathsf{O}, \mathcal{T} \rangle$, G represents "why" $\mathsf{R} \sqcap \mathsf{O}$ are not compatible. For Concept Abduction , having R and O such that $\mathsf{O} \not\sqsubseteq \mathsf{R}$, the solution H to the CAP $\langle \mathcal{L}, \mathsf{R}, \mathsf{O}, \mathcal{T} \rangle$ represents "why" the subsumption relation does not hold. H can be interpreted as *what is specified in R and not in O.*

4 Making the Match

With respect to the match classification presented in Section 2 we show how it is possible to exploit both montonic and non-monotonic inference services for DLs in order to identify match classes, move from a Partial match to a Full (or Exact) match, and use the obtained information to provide a semantic-based score measuring similarity w.r.t. the request.

Using Subsumption and Concept Satisfiability we can rewrite match classes in terms of DLs. Given an ontology $\mathcal{T}$and a demand and a supply, expressed as DL complex concepts R and O, both satisfiable w.r.t. $\mathcal{T}$, we have:

$$\begin{aligned}
\textbf{Exact} &: \mathcal{T} \models \mathsf{R} \equiv \mathsf{O} \\
\textbf{Full} &: \mathcal{T} \models \mathsf{O} \sqsubseteq \mathsf{R} \\
\textbf{Plug-In} &: \mathcal{T} \models \mathsf{R} \sqsubseteq \mathsf{O} \\
\textbf{Potential} &: \mathcal{T} \not\models \mathsf{R} \sqcap \mathsf{O} \sqsubseteq \bot \\
\textbf{Partial} &: \mathcal{T} \models \mathsf{R} \sqcap \mathsf{O} \sqsubseteq \bot
\end{aligned}$$

Both Concept Abduction and Concept Contraction can be used to suggest guidelines on what, given O, has to be revised and/or hypothesized to obtain a Full match with R.

4.1 Explanation for Ranking

Partial→Potential If $\mathsf{R} \sqcap \mathsf{O} \equiv \bot$ – Partial match – then solving the related Concept Contraction Problem we have $\mathsf{R} \equiv \mathsf{G}_{\mathsf{R}} \sqcap \mathsf{K}_{\mathsf{R}}$ such that $\mathsf{K}_{\mathsf{R}} \sqcap \mathsf{O} \not\equiv \bot$ w.r.t.

$\mathcal{T}$. That is, we contract R to $\mathsf{K_R}$ such that there is a Potential match between the contracted request and O.
Potential→Full Once we are in a Potential match, we can formulate hypotheses on what should be hypothesized in O in order to completely satisfy the contracted R. If we solve the related Concept Abduction Problem, we can compute an hypothesis H such that $\mathsf{O} \sqcap \mathsf{H} \sqsubseteq \mathsf{K_R}$ and reach a Full match with the contracted request.
The above concepts can be formalized in the following simple algorithm:

```
Algorithm retrieve($\mathsf{R}, \mathsf{O}, \mathcal{T}$)
input $\mathsf{O}, \mathsf{R} \equiv \mathsf{K} \sqcap \mathsf{G}$ concepts satisfiable w.r.t. $\mathcal{T}$
output $\langle \mathsf{G}, \mathsf{H} \rangle$, i.e., the part in $\mathsf{R}$ that should be retracted
and the part in $\mathsf{O}$ that should be hypothesized to have a
full match between $\mathsf{O}$ and $\mathsf{K}$ (the contracted $\mathsf{R}$)
begin algorithm
1:   if $\mathcal{T} \models \mathsf{R} \sqcap \mathsf{O} \sqsubseteq \bot$ then
2:      $\langle \mathsf{G}, \mathsf{K} \rangle = contract(\mathsf{O}, \mathsf{R}, \mathcal{T})$;
3:      $\mathsf{H_K} = abduce(\mathsf{O}, \mathsf{K}, \mathcal{T})$;
4:      return $\langle \mathsf{G}, \mathsf{H} \rangle$;
5:   else
6:      $\mathsf{H} = abduce(\mathsf{O}, \mathsf{R}, \mathcal{T})$;
7:      return $\langle \top, \mathsf{H} \rangle$;
end algorithm
```

Notice that $\mathsf{H} = abduce(\mathsf{O}, \mathsf{R}, \mathcal{T})$ [rows 3,6] determines a concept H such that $\mathsf{O} \sqcap \mathsf{H} \sqsubseteq \mathsf{R}$, $\langle \mathsf{G}, \mathsf{K} \rangle = contract(\mathsf{O}, \mathsf{R}, \mathcal{T})$ [row 2] determines two concepts G and K such that $\mathsf{R} \equiv \mathsf{G} \sqcap \mathsf{K}$ and $\mathcal{T} \models \mathsf{K} \sqcap \mathsf{O} \not\sqsubseteq \bot$ following minimality criteria as suggested in [7, 5]. Also notice that, given a CAP or a CCP, usually there is not only one single solution. But in order to provide a match explanation, in rows 2,3 and 6 only one solution has to be returned. In this case context-aware information may be used as criteria for solution selection.

4.2 Query Refinement

Full→quasi-Exact During this step, in order to overcome the suggestion of "fake bonuses", we have to identify which part of O generated the inconsistency with R before contracting. We can solve a Concept Contraction Problem between O and R contracting O. That is we have $\mathsf{O} \equiv \mathsf{G_O} \sqcap \mathsf{K_O}$ such that $\mathsf{K_O} \sqcap \mathsf{R} \not\sqsubseteq \bot$ w.r.t. $\mathcal{T}$. In [7], among others, the conjunction minimal solution to a CAP is proposed for DLs admitting a normal form with conjunctions of concepts. A solution belonging to such solution is in the form $\mathsf{B} = \sqcap_{j=1..k} C_j$, where C_j are DL concepts and is **irreducible**, *i.e.*, B is such that for each $h \in 1, ..., k$, $\sqcap_{j=1..h-1,h+1..k} C_j$ is not a solution for the CAP.

In the following, the algorithm *computeBonus*$(\mathsf{O}, \mathsf{R}, \mathcal{T})$ is presented, able to compute what should be hypothesized in the requester preferences in order to get a better match result, and –if possible– an Exact match (see 2). It takes as input an offer O, a request R and the ontology $\mathcal{T}$they refer to.

```
**Algorithm** $computeBonus(\mathsf{O}, \mathsf{R}, \mathcal{T})$
**input** $\mathsf{O}$ and $\mathsf{R}$ DL concepts both satisfiable w.r.t. $\mathcal{T}$
reference ontology
**output** $B_{irr}$ a set of DL concepts representing bonuses
**begin algorithm**
1:    $B = \emptyset$;
2:    $B_{irr} = \emptyset$;
3:    $\langle \mathsf{G_R}, \mathsf{K_R} \rangle = contract(\mathsf{O}, \mathsf{R}, \mathcal{T})$;
4:    $\langle \mathsf{G_O}, \mathsf{K_O} \rangle = contract(\mathsf{R}, \mathsf{O}, \mathcal{T})$;
5:    $\mathsf{B} = abduce(\mathsf{K_R}, \mathsf{K_O}, \mathcal{T})$;
6:    **for each** $C_j \in \mathsf{B}$
7:        $B_{irr} = B_{irr} \cup \{C_j\}$;
8:    **return** $B_{irr}$;
**end algorithm**
```

The problem of ***fake bonuses*** is taken into account in rows 3-5 of *computeBonus*. In row 3, a Concept Contraction Problem is solved, contracting R in $\mathsf{K_R}$ and identifying in $\mathsf{G_R}$ the source of inconsistency with O. In row 4 the same is performed for O identifying in $\mathsf{K_O}$ the part of the offer which is compatible with R and in $\mathsf{G_O}$ the incompatible one and then likely to contain the ***fake bonuses***. In row 5 we compute B, solution of Concept Abduction Problem such that $\mathsf{K_R} \sqcap \mathsf{B} \sqsubseteq \mathsf{K_O}$. Notice that adding $\mathsf{B}$to $\mathsf{K_R}$ we are neither in a Plug-In match nor in an Exact one with respect to the contracted request $\mathsf{K_R}$. In fact, we would have a Plug-In match if $\mathsf{K_R} \sqcap \mathsf{B} \sqsubseteq \mathsf{O}$ rather than $\mathsf{K_O}$ and we could have an Exact match adding also ***fake bonuses*** which are isolated now in $\mathsf{G_O}$.

5 Conclusion

In this paper we have motivated the need for nonmonotonic inference services, particularly abduction and contraction in a belief revision framework, for logic-based matchmaking and query refinement in e-commerce scenarios. We have also outlined some guidelines on how to exploit these services for matches explanation and ranking.

Acknowledgments

We wish to acknowledge partial support of Apulia Strategic Project DIPIS (Distributed Production as Innovative System). We also thank the anonymous reviewers for their valuable suggestions.

References

1. S. Agarwal and S. Lamparter. smart - a semantic matchmaking portal for electronic markets. In *Proceedings of the 7th International IEEE Conference on E-Commerce Technology 2005*, 2005.

2. F. Baader, D. Calvanese, D. Mc Guinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.
3. B. Benatallah, M-S. Hacid, C. Rey, and F. Toumani. Request Rewriting-Based Web Service Discovery. In *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 2003.
4. B. Benatallah, M-S. Hacid, C. Rey, and F. Toumani. Semantic Reasoning for Web Services Discovery. In *Proc. of Workshop on E-Services and the Semantic Web at WWW 2003*, May 2003.
5. S. Colucci, T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Concept Abduction and Contraction in Description Logics. In *Proceedings of the 16th International Workshop on Description Logics (DL'03)*, volume 81 of *CEUR Workshop Proceedings*, September 2003.
6. S. Colucci, T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Concept Abduction and Contraction for Semantic-based Discovery of Matches and Negotiation Spaces in an E-Marketplace. *Electronic Commerce Research and Applications*, 4(4):345–361, 2005.
7. T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Abductive matchmaking using description logics. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 337–342, Acapulco, Messico, August 9–15 2003. Morgan Kaufmann, Los Altos.
8. T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. A system for principled Matchmaking in an electronic marketplace. *International Journal of Electronic Commerce*, 8(4):9–37, 2004.
9. E. Di Sciascio, F.M. Donini, M. Mongiello, and G. Piscitelli. A Knowledge-Based System for Person-to-Person E-Commerce. In *Proceedings of the KI-2001 Workshop on Applications of Description Logics (ADL-2001)*, volume 44 of *CEUR Workshop Proceedings*, 2001.
10. P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* Bradford Books, MIT Press, Cambridge, MA, 1988.
11. J. Gonzales-Castillo, D. Trastour, and C. Bartolini. Description Logics for Matchmaking of Services. In *Proceedings of the KI-2001 Workshop on Applications of Description Logics (ADL-2001)*, volume 44. CEUR Workshop Proceedings, 2001.
12. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *Proceedings of the Second European Semantic Web Conf. ESWC '05*, 2005.
13. M. Klusch, B. Fries, M. Khalid, and Katia Sycara. Owls-mx: Hybrid owl-s service matchmaking. In *Proceedings of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, 2005.
14. L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *Proc. International World Wide Web Conference (WWW '03)*, pages 331–339, Budapest, Hungary, May 20–24 2003. ACM, New York.
15. OWL. Web Ontology Language. www.w3.org/TR/owl-features/, 2004.
16. M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *The Semantic Web - ISWC 2002*, number 2342 in Lecture Notes in Computer Science, pages 333–347. Springer-Verlag, 2002.
17. K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneus Software Agents in Cyberspace. *Autonomous agents and multi-agent systems*, 5:173–203, 2002.
18. D. Trastour, C. Bartolini, and C. Priest. Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In *Proc. International World Wide Web Conference (WWW) '02*, pages 89–98. ACM, 2002.

Semantic Matchmaking using Ranked Instance Retrieval

Matthias Beck and Burkhard Freitag

Universität Passau, Fakultät für Mathematik und Informatik,
{Matthias.Beck, Burkhard.Freitag}@uni-passau.de

Abstract. Finding matching answers to a given request is hard, in particular if the objective is to find the most suitable partner. This especially holds in the context of the Semantic Web and its underlying specification language, i. e., description logics (DL)[1]. We present a clean, formal approach allowing users to devise preferences for queries on description logics knowledge bases. Thereby it is possible to use established DL-reasoners, particularly those tailored at instance retrieval like KAON2. Furthermore, our approach allows the formulation of soft constraints which is considered an important feature of semantic matchmaking.

1 Introduction

It is no news that matchmaking plays a crucial role in many areas of information systems. In the context of the Semantic Web, for example, matchmaking plays a central role in the discovery of appropriate semantic web services (cf. [1]) where appropriate means best complying with the user's request. So it is natural to provide the user with means to express his or her criteria for determining quality.

The contribution of this paper is a uniform method for the annotation of description logics instance retrieval queries with user preferences thereby allowing soft constraints resp. mere sorting. Since only the query is annotated and the knowledge base remains unchanged, it is possible to use standard reasoners like KAON2 [2], RACER [3] or, more generally, all reasoners supporting the so called DIG-Interface [4].

2 Motivating Example

Let us have a look at an example. Assume, you want to buy the latest "Harry Potter" DVD. You would probably look only for those web services that allow to order DVDs. Assume further that there is a registry which stores the service profiles of the particular web services in an appropriate formalism like OWL-S [5], which allows for more sophisticated semantic service descriptions as compared to standard formalisms like UDDI (cf. [6]). This easy query can be formulated as

[1] Other specification languages like the object-oriented Flora-2 are possible for the Semantic Web. Nonetheless, OWL is the current recommendation of the W3C.

$Q_1 := \mathit{OffersDVD}$. Retrieving results for Q_1 might be a quite tiresome experience since you would probably get tons of hits.

Of course you would provide more search criteria to narrow the search. For instance, if the DVD is meant to be a present and you are a little late, you would prefer shipping within 24 hours. But in any case shipping time should not exceed three days. This yields a new query

$$Q_2 := \mathit{OffersDVD} \sqcap (\mathit{24HoursShipping} \sqcup \mathit{3DaysShipping}).$$

That produces the correct result set[2], but without specifying user preferences, you get an unordered result that does not fit your needs. So the next step is to annotate the query with (numerical) user preferences. This results in the query

$$Q_3 := \mathit{OffersDVD}^1 \sqcap (\mathit{24HoursShipping}^2 \sqcup \mathit{3DaysShipping}^1).$$

The intuitive meaning is the following: in the result set individuals (e. g., web services) that provide 24 hours shipping[3] will get a higher rank than those offering three days shipping only. Note that membership in the result set is not affected by specifying preferences.

3 Preference Annotations

3.1 Ranking Tree

Due to the structure of queries and hence preferences a single numerical value is not sufficient to express rankings. The main reason is the existence of disjunctive knowledge in description logics. Consider the query $Q_4 := A^1 \sqcup (B^1 \sqcup C^2)^0$. Intuitively, since $B \sqcup C$ has a preference of 0, this expression should not contribute to the top level rank. On the other hand, if we get an equal top level rank for two individuals, an evaluation of $B^1 \sqcup C^2$ can be used to refine the ranking.

Definition 1 (Ranking Tree)

1. *For $r \in [0;1]$ (r) is a ranking tree.*
2. *Let $r \in [0;1]$ and $t_1, \ldots, t_n$ ranking trees with $n \geq 1$, then $(r, t_1, \ldots, t_n)$ is a ranking tree.*

$t_1 := (1, (1), (\frac{1}{3}, (1), (0)))$ is a sample ranking tree for query Q_4 (cf. section 3.2). Based on this definition we construct an ordering on ranking trees.

Definition 2 (Ordering $\trianglelefteq$ on ranking trees)
Let $a = (r_a, a_1, \ldots, a_n)$ and $b = (r_b, b_1, \ldots, b_n)$ with $n \in \mathbb{N}_0$ and $r_a, r_b \in \mathbb{R}$, a_i, b_i ranking trees with $i \leq n$. Then we define the relation $<$ by $a < b :\Leftrightarrow r_a < r_b$. Now we define that $a \trianglelefteq b$ holds if and only if

[2] The $\sqcup$-Operator is equivalent to a logical "or" while $\sqcap$ corresponds to a logical "and".
[3] In a more realsitic scenario you would possibly replace $\mathit{24HoursShipping}^2$ by $(\exists \mathit{shipsWithin}. \leq_{24})^2$ in query Q_3. The same holds for *3DaysShipping*. For the sake of brevity we stick to the shorter form.

$$
\begin{aligned}
r_{KB}(C_1^{r_1} \diamond \ldots \diamond C_n^{r_n}, o) &= (d, r_{KB}(C_1, o), \ldots, r_{KB}(C_n, o)) \\
\text{with} \quad d &= \begin{cases} 0, & \text{if } r_i = 0, \text{ for } i \in \{1, \ldots, n\} \\ \frac{\sum_{i=1}^{n} c_i r_i}{\sum_{i=1}^{n} r_i}, & \text{else} \end{cases} \\
\text{and} \quad c_i &= \begin{cases} 1, \text{if } KB \models C_i(o) \\ 0, \text{else} \end{cases} , \diamond \in \{\sqcup, \sqcap\} \\
r^{\neg}_{KB}(C_1^{r_1} \diamond \ldots \diamond C_n^{r_n}, o) &= (d^{\neg}, r^{\neg}_{KB}(C_1, o), \ldots, r^{\neg}_{KB}(C_n, o)) \\
\text{with} \quad d^{\neg} &= \begin{cases} 0, & \text{if } r_i = 0, \text{ for } i \in \{1, \ldots, n\} \\ \frac{\sum_{i=1}^{n} c_i^{\neg} r_i}{\sum_{i=1}^{n} r_i}, & \text{else} \end{cases} \\
\text{and} \quad c_i^{\neg} &= \begin{cases} 1, \text{if } KB \models \neg C_i(o) \\ 0, \text{else} \end{cases} , \diamond \in \{\sqcup, \sqcap\} \\
r_{KB}(\neg C, o) &= r^{\neg}_{KB}(C, o) \\
r^{\neg}_{KB}(\neg C, o) &= r_{KB}(C, o) \\
r_{KB}(\mathcal{X} R.C, o) &= -1, \mathcal{X} \in \{\exists, \forall, \leq n, \geq n\}, n \in \mathbb{N} \\
r^{\neg}_{KB}(\mathcal{X} R.C, o) &= -1, \mathcal{X} \in \{\exists, \forall, \leq n, \geq n\}, n \in \mathbb{N} \\
r_{KB}(A, o) &= \begin{cases} (1), & \text{if } KB \models A(o) \\ (0), & \text{else} \end{cases} \\
r^{\neg}_{KB}(A, o) &= \begin{cases} (1), & \text{if } KB \models \neg A(o) \\ (0), & \text{else} \end{cases}
\end{aligned}
$$

KB: knowledge base; $C, C_1 \ldots, C_n$: arbitrary concept expressions
A: atomic concept; R: role symbol; o: individual

Fig. 1. Ranking mapping

1. *$a < b$ or*
2. *$r_a = r_b$ and $\exists i : a_i < b_i$ and $\forall 1 \leq j \leq n : b_j \not< a_j$ or*
3. *$r_a = r_b \wedge \forall 1 \leq i \leq n : a_i \trianglelefteq b_i$*

Giving this definition a closer look, we can see that $a \trianglelefteq b$ if the top-level rank of a is smaller. If the top-level ranks are equal, there are two possibilities: one child top-level rank is smaller (and other child ranks are not greater) or all child trees are smaller w. r. t. $\trianglelefteq$. For example, if we have a second ranking tree $t_2 := (1, (1), (0, (0), (0)))$, then $t_2 \trianglelefteq t_1$ holds. Note that $\trianglelefteq$ is a partial order relation on the set of ranking trees (having the same structure).

3.2 Ranking Mapping

Given an annotated query and an individual, we must evaluate the rank (ranking tree) of that individual with respect to the query. This is done by the mapping shown in figure 1. Note that the result of the ranking mapping is a ranking tree. It is only meaningful to compare ($\trianglelefteq$) ranking trees if they were assembled using the same query. We do not further elaborate on the definition since it is rather technical. For background information on description logics we refer to [7]. Nonetheless we present some remarks on the mapping.

Central part of the definition is the mapping for (n-ary) conjunctions and disjunctions. The definition of the ranking function is the same for both. The ranking tree is created by computing a top-level rank d and afterwards computing ranking trees for the operands. The top-level rank is the weighted average[4] of the preferences. Negation is treated by "saving" it inside the $r_{KB}^{\neg}$-function. The negation is then re-applied while computing the $c_i^{\neg}$-parameter. This scheme is passed on to the operands.

Preferences inside role expressions are cut off by setting the rank of the expression to -1. This is justified as follows: consider the query $\exists hasBeach(Rocky^1 \sqcup Sandy^2)$. We must build a ranking for locations that have beaches by evaluation of certain characteristics of these beaches, in this case *Rocky* and *Sandy*. If more than one beach is associated to a single location, there are several possible semantics. A possibility is trying to maximize the ranking. In this example that would mean trying to find an associated beach that is rocky and sandy. It is also possible to evaluate the average ranking over all associated beaches. There are other reasonable semantics, too.

We are currently examining preferences inside role expressions and plan for an annotation method allowing the user to specify the particular aggregation function to be used. For the time being, preferences inside role expressions are not considered to avoid ambiguity. The structure of the resulting ranking tree very much resembles the query structure. The ranking tree is more compact because negation nodes are left out and nodes resulting from role expressions are cut off.

Recall query $Q_4 := A^1 \sqcup (B^1 \sqcup C^2)^0$ and ranking tree $t_1 := (1, (1), (\frac{1}{3}, (1), (0)))$ from section 3.1. If there is an individual o that is an instance of A and B but not C, then t_1 is the ranking tree for o w. r. t. Q_4.

3.3 Soft Constraints and Sorting

Definitions of the ranking tree and the ordering $\trianglelefteq$ allow for formulation of soft constraints using our formalism. So another way of refining query Q_1 from section 2 is to specify some soft constraints that you prefer being satisfied without enforcing them. For example you could prefer paying by credit card. Specifying this is now easy using preferences: $Q_5 := OffersDVD^1 \sqcap (CreditCardPayment^1 \sqcup \top^0)$. Here $\top$ denotes the top concept, i. e., every individual is an instance of $\top$. The result set consists of all individuals being instances of *OffersDVD* but those also providing credit card payment get a higher rank. Since we allow for n-ary operators, it is also possible to combine soft constraints: $Q_6 := OffersDVD^1 \sqcap$

[4] Aside from the weighted average, the rank could be computed by an arbitrary function.

```
<owl:Class rdf:about="http://www.im.uni-passau.de/pref#Query">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="http://www.im.uni-passau.de/matchmaking#UPS">
      <pref:Pref>1</pref:Pref>
    </owl:Class>
    <owl:Class rdf:about="http://www.im.uni-passau.de/matchmaking#FedEx">
      <pref:Pref>2</pref:Pref>
    </owl:Class>
  </owl:unionOf>
</owl:Class>
```

Fig. 2. Preference annotated query in OWL ($\mathit{UPS}^1 \sqcup \mathit{FedEx}^2$)

($\mathit{CreditCardPayment}^1 \sqcup \mathit{DeliveryAsGift}^2 \sqcup \top^0$). Moreover nesting is possible[5]:

$$Q_7 := \mathit{OffersDVD}^1 \sqcap (\mathit{CreditCardPayment}^2 \sqcup (\mathit{FedEx}^1 \sqcup \mathit{UPS}^2)^1 \sqcup \top^0)$$

It is worth noting here that it is possible to rank uncertain (disjunctive) knowledge. If the knowledge base entails the fact that an individual o is an instance of $\mathit{FedEx} \sqcup \mathit{UPS}$, this knowledge is ranked using preference 1 in Q_7. Since description logics allow for disjunctive knowledge, this is possible even without explicitly knowing whether o is an instance of *FedEx* or *UPS*. On the other hand, if we knew that o is also an instance of *UPS* for example, this knowledge would increase the rank on a lower ranking level. In addition to soft constraints it is even possible to formulate mere sorting conditions:

$$Q_8 := \mathit{OffersDVD}^1 \sqcup (\mathit{CCardPaym}^2 \sqcap \mathit{24HoursDelivery}^1 \sqcap \mathit{FedEx}^3 \sqcap \bot^0)$$

Since no individual is an instance of $\bot$, the second part of the formula is not satisfiable. So all results to query Q_8 must be instances of *OffersDVD* and therefore *OffersDVD* cannot affect the ranking. Hence, the second part is alone responsible for the ranking. This is due to the fact that an individual i can never be instance of $\mathit{CCardPaym} \sqcap \mathit{24HoursDelivery} \sqcap \mathit{FedEx} \sqcap \bot$. However, i can be an instance of *CCardPaym*, *24HoursDelivery* or *FedEx*. This information is used to build the ranking.

4 Prototype

Our ranking algorithm is composed of three parts that essentially represent three consecutive phases. In phase one the preference annotated query given in OWL is (pre)processed, preferences are extracted and removed from the query. For an example query see fig. 2. Results to the unannotated query are retrieved via the reasoner in phase two. At the moment we build upon the KAON2 reasoner [2][8]

[5] Note that *FedEx* (the same holds for *UPS*) is short for *offersShippingByFedEx* or $\exists \mathit{offersShippingBy}.\mathit{FedEx}$ in Query Q_7.

since it is tailored to instance retrieval. We plan to support other reasoners like RACER [3] and other DIG [4] compatible reasoners in the future.

Phase three performs the actual ranking. This is done by computing the ranking tree for each result retrieved in phase two. The results of several test runs are encouraging. We are using an indexed cache to speed up the computation. In our test runs we observed that the time for computation of the ranked result set t_{rank} depends linearly on the time for un-ranked retrieval t_{ret}, i. e., $t_{rank} \approx n \cdot t_{ret}$. The factor n is determined by the complexity of the query.

5 Discussion and Summary

Many approaches to ranking and preferences have been proposed. Castillo et al. [9] studied matchmaking of services by means of description logics. They allow for neither ranking nor user preferences. A different approach is due to Colucci et al. [10]. They allow for both negotiable and strict requirements in a description and introduce two novel description logics inference services, concept abduction and concept contraction, to deal with them.

Kießling et al. [11] developed PreferenceSQL, a powerful SQL extension. The objective is similar to our approach. While PreferenceSQL is aimed at relational data, our target are description logics knowledge bases. So in PreferenceSQL preferences are set on the attribute level while we set preferences on class membership. Another difference is that we can handle disjunctive knowledge which is common in description logics but not in databases. Besides, we allow for preferences on arbitrary constraints while this is only allowed for soft constraints in PreferenceSQL.

Brewka [12] describes a rank based description language with qualitative preferences. He follows a slightly different approach: central component is the ranking of different *models* of the knowledge base, not a ranking of consequences with respect to a query as in our approach. To accomplish that, he deeply integrates preferences into the logical language. Consequently, standard reasoners cannot be used. Another difference lies in the fact that his approach admittedly does not work well with numerical (quantitative) preferences.

A Google-like approach to knowledge ranking is presented by Ding et al. in [13]. They do not allow for user-defined preferences.

A recent paper [14] on the combination of ontologies and preferences is due to Lukasiewicz et al. Their notion of preference differs from ours. Particularly, they allow for qualitative preferences only.

As a natural limitation of our approach, annotations of user preferences must follow the query (term) structure. We argue that this is not a serious restriction since query and preference structure often match. Otherwise it is possible to specify a sorting (cf. section 3.3).

We have introduced an approach to semantic matchmaking by ranked instance retrieval in knowledge bases represented in description logics. Ranking is solely based on the query which is annotated with preferences. One of the advantages of our ranking approach lies in the fact that each user gets exactly the ranking he or she specified. Therefore, ranking is personalized and self-explanatory.

A prototype implementation has been presented that shows promising behavior. Of course several optimizations of our algorithm are possible and already on the way. For example, in certain cases a complete computation of the ranking tree is not necessary since the ranking can be evaluated directly for some concepts without looking at subconcepts. Also a more intensive use of index structures will be investigated.

6 Acknowledgement

We wish to thank the anonymous reviewers for valuable comments.

References

1. Burstein, M.H., Bussler, C., Zaremba, M., Finin, T.W., Huhns, M.N., Paolucci, M., Sheth, A.P., Williams, S.K.: A Semantic Web Services Architecture. IEEE Internet Computing **9**(5) (2005) 72–81
2. Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB), Universität Karlsruhe (TH): KAON2 - Homepage. http://kaon2.semanticweb.org (2006) last visited May 2006.
3. Haarslev, V., Möller, R.: Description of the RACER System and its Applications. In: Proceedings International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August. (2001) 131–141
4. Bechhofer, S., Möller, R., Crowther, P.: The DIG Description Logic Interface. In: Description Logics. (2003)
5. Burstein, M., Hobbs, J., Lassila, O., Mcdermott, D., Mcilraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services. Website (2004)
6. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Importing the Semantic Web in UDDI. In: CAiSE '02/ WES '02: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web, London, UK, Springer-Verlag (2002) 225–236
7. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press (2003)
8. Hustadt, U., Motik, B., Sattler, U.: Reducing SHIQ-Description Logic to Disjunctive Datalog Programs. In: KR. (2004) 152–162
9. González-Castillo, J., Trastour, D., Bartolini, C.: Description Logics for Matchmaking of Services. In Görz, G., Haarslev, V., Lutz, C., Möller, R., eds.: Proceedings of the KI-2001 Workshop on Applications of Description Logics. (2001)
10. Colucci, S., Noia, T.D., Sciascio, E.D., Donini, F.M., Mongiello, M.: Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. Electronic Commerce Research and Applications **4**(4) (2005) 345–361
11. Kießling, W., Köstler, G.: Preference SQL - Design, Implementation, Experiences. In: VLDB. (2002) 990–1001

12. Brewka, G.: A Rank Based Description Language for Qualitative Preferences. In de Mántaras, R.L., Saitta, L., eds.: ECAI, IOS Press (2004) 303–307
13. Ding, L., Pan, R., Finin, T.W., Joshi, A., Peng, Y., Kolari, P.: Finding and Ranking Knowledge on the Semantic Web. In: International Semantic Web Conference. (2005) 156–170
14. Lukasiewicz, T., Schellhase, J.: Variable-Strength Conditional Preferences for Matchmaking in Description Logics. In: KR. (2006) 164–174

Semantic-based Taxonomic Categorization of Web Services

Miguel Ángel Corella and Pablo Castells

Universidad Autónoma de Madrid, Escuela Politécnica Superior
Campus de Cantoblanco, 28049 Madrid, Spain
{miguel.corella, pablo.castells}@uam.es

Abstract. With the envisioned proliferation of Web services available on the WWW and private repositories, new and better support techniques are needed for service discovery and organization to stay manageable. Service classification under hierarchic taxonomies is commonly a key feature for properly organizing service repositories in a rational way, as well as a good foundation for sophisticated retrieval techniques. In this paper, a heuristic approach for the semi-automatic classification of (semantic) Web services is proposed, based on matching new unclassified services to previously classified ones in a given corpus. This hypothesis is validated by an experimental test and the comparison with results achieved by other approaches.

1 Introduction

A major envisioned area where the emerging ontology-based technologies [3] are expected to bring key benefits is that of using semantic-based descriptions to endow Web services with a qualitatively higher potential for automation [19]. The basis of this trend is to add semantic information beyond current WSDL-based [6] descriptions, as a means to enable the manipulation of services by software programs easing the automation of such tasks as service selection, invocation, composition, and discovery [11]. Further service management tasks, besides the latter, can also take advantage of the additional semantics to provide with new automation facilities. In this paper we address the categorization (i.e. specification of their domain or business focus) of Web services in a repository with respect to a predefined taxonomy.

Nowadays, the most widely accepted and used protocol for publishing and searching Web services is UDDI [15]. This protocol enables the creation of service repositories, in which services are organized based on standard or proprietary taxonomies (e.g. UNSPSC – United Nations Standard Products and Service Codes, NAICS – North American Industry Classification System, etc.). Each service is classed under one (or more) categories, in order to ease different repository tasks performed by service publishers and consumers, as well as repository administrators.

Since the categorization of services and maintenance of repositories has to be done manually by human entities, the classification task becomes considerably difficult, heavy and error-prone in practice, due to several issues (e.g. huge size of taxonomies in real-world applications, multiple people involved in maintaining or sharing services in a common repository, several distributed repositories being shared, etc.).

According to this, it is our goal to provide automatic mechanisms to assist service publishers in the categorization task, in order to reduce the effort required, and promote globally consistent classification decisions, even when several users are involved. To do so, we propose a heuristic-based classification system that computes a ranked list of candidate classes from a given taxonomy, in which a new service better fits, by comparing the new service with the services that are already classified and published in a given repository. Besides this classification aid, the categorization information thus obtained can be used for the enhancement of further tasks, such as the ones related to semi-automatic service annotation and discovery [12].

The paper is organized as follows: Section 2 introduces some related work already presented in the domain of Web service classification. The definition of the problem of Web service classification in a formal way, and the motivation of why service semantics are needed in order to successfully solve it are given in Section 3. Our classification heuristic is presented in Section 4. Finally, Section 5 provides conclusions and outlines future work directions.

2 Related work

The problem of the automatic categorization of Web services has been addressed in prior work from two main approaches, that we may class as heuristic (e.g. [16]) and non-heuristic (e.g. [5] and [10]). In the proposal presented in [10] the authors offer two different strategies, the first one consisting of using the information contained in WSDL-based descriptions to select a category in which the service fits best. The second one aims at dynamically creating the classification taxonomy by using WSDL descriptions as input. Their procedure consists of the following steps: a) extraction of meaningful words from service descriptions, using Natural Language Processing techniques, b) construction of term vectors with those words, and c) application of different classification techniques for vector categorization, namely machine learning in their first strategy, and clustering techniques in the second. So, service classification problem is solved by a text classification problem approach.

The approach to classification in [5] proposes similar steps as the ones described in [10]. The main difference is the usage of Support Vector Machines as term vector classification mechanism. Again, service classification is reduced to a text classification problem. In addition, this proposal provides service publishers with extra information after the classification. More precisely, a concept lattice, extracted using Formal Concept Analysis over the term vectors, is presented to the publishers. This lattice allows developers to know how the words used in service descriptions (the ones extracted during classificaiton) contribute to the selection of a specific category. With this information, service developers could, for example, modify some description words which may cause ambiguity in the classification process.

The authors of [16] propose a framework for the automation of the semantic annotation of services. Using domain ontologies (i.e. the ones used for the annotation) as service categories, service classification is used to select the most suitable ontology in the taxonomy of domain ontologies. To do so, an algorithm to match Web service data types (in XML Schema) and concepts is defined, based on schema matching.

Another research area related to the work presented here is that of service matchmaking (see e.g. [13] and [17]). It is related to Web service classification in that our approach computes similarity degrees between services in order to assign them to a common category, and service matchmaking aims to find services that match a concrete capability description. The main difference between both research areas is that while service classification admits some degree of fuzziness in service matching, i.e. continuous similarity measures, service matchmaking typically does not; using discrete matching levels (e.g. "no match", "complete match", "partial match", etc.).

3 Semantics for service classification

As mentioned earlier, service classification is a common necessity to make service administration and retrieval manageable for human users. Moreover, it can serve as a complementary aid for automatic service discovery and selection techniques. Nevertheless, there are usability problems involved in service categorization which cause difficulties for this categorization in real-world environments. Such problems include:

- Classification taxonomies can be extremely large, comprising thousands of categories (e.g. UNSPSC ~ 20,000 classes, NAICS ~ 2,300 classes).
- The number of services in a repository can grow quite large, making it impossible for administrators to validate the information published along with a service.
- The placement of a service under a proper category requires a considerable amount of knowledge of the complete taxonomy in order to make appropriate decisions.

The work presented here aims at alleviating the administrator's work, and reducing the categorization effort for service providers. Our proposal approaches the classification problem as follows. Given a set of services already classified under a given taxonomy, and a new service description to be published, the unclassified service is compared with the classified ones, whereby a measure of the likelihood that the service should be assigned a certain category is computed.

WSDL descriptions provided by current technologies are not suitable for this purpose, as they only focus on the syntactic view of the services, which is not sufficient to support valid service classification criteria in practice. As stated in the specification of WSDL standard, descriptions in this language provide details about the operations a service supports, and the input / output information involved in their invocation. The fact is that, although this information enables comparisons between services, a much accurate matching is possible if semantic information is added to the descriptions. Consider this example: take two Web services, the first one defining currency conversion capabilities, and the second one, a service to compute distances between cities. The currency converter service could have one operation, involving:

- An input message with two currency codes, of type string.
- An output message with one part containing the conversion rate (a double).

On the other hand, the distance calculator would have also one operation, having:

- An input message with one part containing two city names of string type.
- An output message with one part containing the distance between them (a double).

From a conceptual point of view, these services should yield a low similarity measure value when compared. However, since their interfaces are syntactically equivalent (same number of operations and messages, same data types), comparing their WSDL descriptions would produce a very high result value. So, extra information beyond mere syntactic WSDL definitions is needed. There are at least two possibilities:

- Using identifiers of the different WSDL elements. The hypothesis here is that the names given to operations, parameters, messages, etc., are often meaningful, which, from our point of view, is too idealistic and often fails.
- Using semantic Web service descriptions. In the proposed example, it is clear that the relevant information about service parameters is their semantic meaning (e.g. currency codes, city names, etc.). WSDL descriptions do not foresee this kind of semantics, but ontology-based service descriptions do. Thus, supporting discrimination between syntactically equivalent parameters having different semantics.

The need for service semantics is thus clear. The syntactic information in WSDL-based descriptions is not sufficient, and would often lead to inconsistent similarity values, and therefore, to service misclassification. Semantic Web service descriptions can solve this problem by providing means to describe service inputs and outputs from a conceptual point of view. The approach here presented is compatible with every semantic description languages such as WSMO [18], OWL-S [14], WSDL-S [1] or SWSO [2].

4 Heuristic classification

The heuristic is divided into three granularity levels, corresponding to the comparison between different service elements involved in the categorization procedure.

Service category level. Since services have to be assigned a category as a result of the classification procedure, this level is needed in order to find evidence that a service should belong to a specific category (used to sort the ranked category list).

The proposed measure works as follows. Let $\mathcal{S}$ be the set of Web services in a repository, and let $\mathcal{C}$ be the classification taxonomy. If we allow a service to be classified under several categories of the taxonomy, we may define the classification by $\mathcal{C}$ as a mapping $\tau : \mathcal{S} \to 2^{\mathcal{C}}$. Given a new service s to be added to $\mathcal{S}$, we want to find the categories in $\mathcal{C}$ that best suit s. Given $c \in \mathcal{C}$, let $\mathrm{P}(s{:}c)$ be the probability that c is an appropriate classification for s, estimated here by comparison of s with the services classified under c. If we take $\mathrm{P}(s{:}c) \sim 0$ if $\{x \in \mathcal{S} \mid c \in \tau(x)\} = \emptyset$, we can write:

$$\mathrm{P}(s:c) \sim \mathrm{P}\Big(s:c \wedge \Big(\bigvee_{x\in\mathcal{S}} c \in \tau(x)\Big)\Big)$$

By rewriting the right hand-side using the inclusion-exclusion principle applied to probability [20], it can be seen that:

$$\mathrm{P}(s:c) \sim \sum_{A\subset\mathcal{S}} (-1)^{|A|+1} \prod_{x\in A} \mathrm{P}\big(c \in \tau(x)\big) \cdot \mathrm{P}\big(s:c \mid c \in \tau(x)\big)$$

provided that $s{:}c \wedge c \in \tau(x)$ are pairwise independent for all $x \in \mathcal{S}$. Since $c \in \tau(x)$ is true iff $x \in \{x \in \mathcal{S} \mid c \in \tau(x)\}$, and assuming a crisp service classification (i.e. $c \in \tau(x)$ is either true or false, as opposed to fuzzy classification where $\mathrm{P}(c \in \tau(x)) \in [0,1]$), we have:

$$\mathrm{P}(s:c) \sim \sum_{A \subset \tau^{-1}(c)} (-1)^{|A|+1} \prod_{x \in A} \mathrm{P}(s:c \mid c \in \tau(x))$$

Now we shall estimate $\mathrm{P}(s{:}c \mid c \in \tau(x))$ by a measure of similarity sim (s, x), that is:

$$\mathrm{P}(s:c) \sim \sum_{A \subset \tau^{-1}(c)} (-1)^{|A|+1} \prod_{x \in A} \mathrm{sim}(s,x)$$

whereby the appropriateness of a category for a service is computed in terms of the similarity between the service and the services classified under that category.

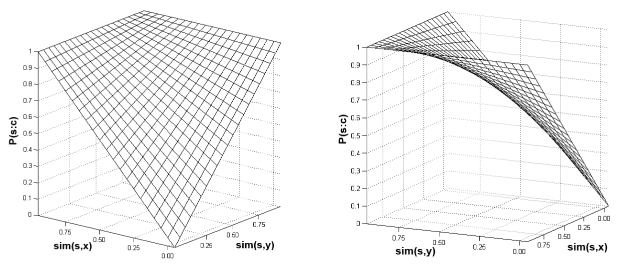


Fig 1. Graph (displayed from two angles) showing the behavior of the measure with respect to sim(s,x) and sim(s,y) (i.e. similarity between a service s and a category c having two services).

Note that $\mathrm{P}(s{:}c) \in [0,1]$, provided that sim $(s, x) \in [0,1]$, and increases monotonically with respect to sim (s, x). Figure 1 shows how $\mathrm{P}(s{:}c)$ behaves with respect to the sim (s, x) value.

Service description level. The comparison between services is based on the assumption that services of the same category deal with similar semantic concepts. Therefore, operation structures (i.e. conceptual roles and grouping of the service parameters) are relevant for service-level comparisons. In fact, the similarity between services is measured in terms of the similarity between operation sets, as follows:

$$\mathrm{sim}(s,s') = \frac{\sum_{i=1}^{\min(|OP|,|OP'|)} \mathrm{sim}(\mathrm{top}(OP_i, OP'_i))}{\max(|OP|,|OP'|)} \left\{ \mathrm{sim}(op,op') = \mathrm{sim}(I_{op}, I_{op'}) \cdot \mathrm{sim}(O_{op}, O_{op'}) \right\{ \mathrm{sim}(P,P') = \frac{\sum_{i=1}^{\min(|P|,|P'|)} \mathrm{sim}(\mathrm{top}(P_i, P'_i))}{\max(|P|,|P'|)}$$

where OP and OP' are the operation sets of services s and s'; I_{op}, $I_{op'}$, O_{op}, $O_{op'}$, are the sets of inputs and outputs of two operations op and op'; and P and P' are the set of parameters used as inputs/outputs in the services.

Note that the service comparison defined here returns values in the range [0,1], provided that the similarity between ontology concepts is also within that range. Figure 2 shows how the sim(s, x) behaves with respect to sim(I_{op}, $I_{op'}$) and sim(O_{op}, $O_{op'}$).

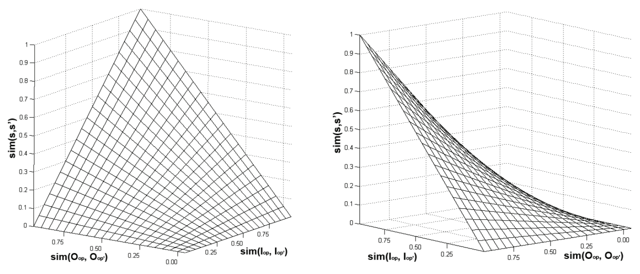


Fig. 2. Service to service similarity measure graph (displayed from two angles) showing the behavior of the similarity measure with respect to sim(I_{op},$I_{op'}$) and sim(O_{op},$O_{op'}$) (i.e. similarity between two services each one with only one operation).

Service parameter level. This last level aims to provide with a similarity value between annotated service parameters, this is, the ontology concepts used to annotate them, i.e. a measure enabling the comparison of domain ontology concepts. A lot of previous work has been developed in this area (e.g. [4], [8], [9]). Nevertheless, a new measure has been developed fitting our specific objectives.

This new measure works as follows. Let $\mathcal{T}$ denote the set of all concepts in the domain ontology. The similarity between two concepts is measured in terms of their distance in the ontology class hierarchy. Given two concepts $t,t' \in \mathcal{T}$, let t_0 be the lower common ancestor to t and t' in $\mathcal{T}$, and let d = dist(t,t_0) + 1, d' = dist(t',t_0) + 1 be the number of levels (plus 1) between t, t' and t_0 in the concept hierarchy. We define the similarity between t and t' as:

$$\mathrm{sim}(t,t') = \left(1 - \frac{\alpha}{\mathrm{h}(\mathcal{T})} \cdot \frac{|d-d'|}{d+d'}\right) \cdot \frac{1}{\min(d,d')} \cdot \left(1 - \frac{\max(d,d')-1}{\mathrm{h}(\mathcal{T})}\right)$$

where:

- h($\mathcal{T}$) is the total height of the concept hierarchy, which is introduced to measure the distance between concepts as a proportion of the total depth of the ontology.
- The term $\frac{|d-d'|}{d+d'}$ increases (that is, the similarity decreases) with the difference in the depth level between t and t'.
- $\alpha \in [0,1]$ (in our test α has been empirically tuned to 0.8) is a parameter that ensures a minimum non-zero similarity value in a way that similarity ranges in some interval [*min*,1] above 0, relaxing the influence of this level in the whole heuristic.
- The factor $1 - \frac{\max(d,d')-1}{\mathrm{h}(\mathcal{T})}$ reinforces the decrease of the similarity when one concept is super concept of the other (i.e. one of the concepts is the first common parent), to achieve a monotonic decrease of the similarity from 1 to 0.

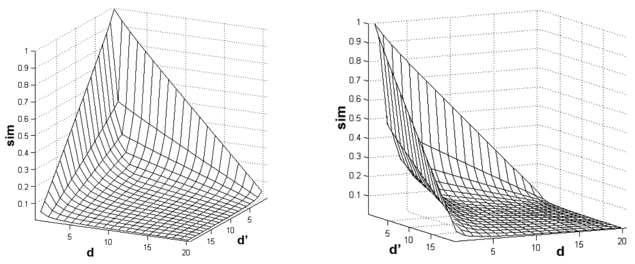


Fig. 3. Concept to concept similarity measure graph (displayed from two angles) showing the behaviour of the similarity measure with respect to *d* and *d*' in a twenty depth levels ontology.

Figure 3 shows how the similarity function sim(*t*,*t*') depends on the distance *d* and *d*' of the *t* and *t*' to their lowest common ancestor.

5 Conclusions and future work

We have presented a new classification approach based on the usage of conceptual service descriptions that can be used to assist service publishers, consumers and repository administrators, in the manual service categorization and retrieval tasks. Moreover, by proposing the classification of new services based on previous decisions made for similar services, consistency is implicitly enhanced against a potential drift over time or across multiple users. All the approach has been based on the hypothesis that the augmentation of service descriptions with semantic information enables a more precise comparison of service descriptions, and therefore an improved accuracy of the automatic classification. As part of the ongoing work we are working on a complete classification framework (implementing the heuristic presented) in order to have a platform in which formally and easily test our approach.

As a continuation of the work presented here we are investigating the potential of our classification approach to enhance service retrieval mechanisms. This line of research is already achieving promising results as reported in [12]. In addition, we envisage the extension of our algorithm to deal with complex descriptions of concepts using axiomatic descriptions of preconditions and post conditions (as supported by WSML) generalizing our matching functions in order to benefit from reasoning tools.

6 Acknowledgements

This research was supported by the Spanish Ministry of Industry, Tourism and Commerce (CDTI05-0436) and the Ministry of Science and Education (TIN2005-0685). Thanks are due to Rubén Lara for all his help and feedback on the research presented.

References

1. Akkiraju, R., Farrel, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A., Verma, K: Web Service Semantics – WSDL-S, Technical Note, Version 1.0, 2005.
2. Battle, S., Bernstein, A., Booley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuiness, D., Su, J., Tabet, S.: Semantic Web Service Ontology (SWSO), Version 1.0, 2005.
3. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American, 2001.
4. Bernstein, A., Kaufmann, E., Bürki, C., Klein, M.: How similar is it? Towards personalized similarity measures in ontologies. In the 7th Internationale Tagung Wirtschaftsinformaitk. Bamberg, Germany, 2005, pp. 1347-1366.
5. Bruno, M., Canfora, G., Di Penta, M., Scognamiglio, R.: An approach to support web service classification and annotation. In Proceedings of the IEEE International Conference on e-Technology, e-Commmerce and e-Services (EEE 2005), Hong Kong 2005.
6. Christiensen, E. et al: Web Service Description Language (WSDL), v1.1.
7. Corella, M. A., Castells, P.: A Heuristic Approach to Semantic Web Services Classification. In Proceedings of the 10th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems (KES 2006), Bournemouth, UK, 2006.
8. Culmore, R., Rossi, G., Merelli, E.: An ontology similarity algorithm for BioAgent. In NETTAB 02 Agents in Bioinformatics. Bologna, Italy, 2002.
9. Ehrig, M., Haase, P., Stojanovic, N.: Similarity for ontologies – a comprehensive framework. Workshop on Enterprise Modelling and Ontology at PAKM 2004. Austria, 2004.
10. Heβ, A., Kushmerick, N.: Automatically attaching semantic metadata to Web Services. In Workshop on Information Integration on the Web (IIWeb2003), Acapulco, Mexico, 2003.
11. Keller, U., Lara, R., Lausen, H., Polleres, A., Fensel, D.: Automatic Location of Services. In 2nd European Semantic Web Conference (ESWC 2005). LNCS Vol. 3532 pp. 1-16.
12. Lara, R., Corella, M.A., Castells.: A flexible model for the discovery of Web services. 1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives (SMR 2006), co-located with VLDB 2006. Seoul, Korea, 2006.
13. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In the International Journal of Electronic Commerce, 8(4):39 – 60. 2004.
14. Maritn, D., Burnstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B. et al: OWL-S: Semantic markup for web services, v1.1, 2004.
15. OASIS: UDDI: The UDDI technical white paper, 2004.
16. Oldham, N., Thomas, C., Sheth, A., Verma, K.: METEOR-S Web Service Annotation Framework with Machine Learning Classification. In Proc. of the 1st Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC'04), California, July 2004.
17. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic Matching of Web Service Capabilities. In Proceedings of the First International Semantic Web Conference, 2002.
18. Roman, D., Lausen, H., Keller, U., de Brujin, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Kifer, M., König-Ries, B., Kopecky, J., Lara, R., Oren, E., Polleres, A., Scicluna, J., Stollberg, M.: Web Service Modeling Ontology (WSMO), 2005.
19. Terziyan, V. Y., Kononenko, O.: Semantic web enabled web services: State-of-the-art and industrial challenges. In Proc. International Conference on Web Services (ICWS), 2003.
20. Whitworth, W. A.: Choice and Chance, with one thousand exercises. Hafner Pub. Co. New York, 1965.

Helping Architects In Retrieving Architecture Documents: A Semantic Based Approach

Rambabu Duddukuri[1], Prabhakar T.V[2]

[1] Member, Technical Staff, Oracle India Private Ltd,
Bangalore, India -560029.
Rambabu.Duddukuri@oracle.com

[2] Professor, Department Of Computer Science and Engineering,
Indian Institute of Technology Kanpur, Kanpur India - 208016
tvp@iitk.ac.in

Abstract. Large organizations have a need and challenge of archiving the architecture work done on software projects. Knowledge management in such organizations depends on how well the company preserves the knowledge acquired on completed projects and how well the company provides facilities to retrieve that architectural knowledge. Architecture properties such as styles, patterns, tactics and quality requirements play a major role while architecting the systems. Annotating the documents with such properties helps the architect in searching for architecture documents at a later date. There exist a large number of relationships between these architecture properties. A huge knowledge base is required to know about the best practices and the existing relationships between them. In this paper, we present an ontology for these architecture properties. We describe how this ontology can be used in various applications like semantic based search, academic purpose and building of new systems using the best practices.

Keywords: Software Architecture, Ontologies, Semantic search, Architecture styles, Patterns, Tactics, Archiving, Annotation.

1. Introduction

The architecture phase has become an integral part in the design process of large and complex systems. Architecting a system, deals with modeling high level structures of the system in terms of views, architecture styles and patterns. Documenting the architecture of a system is essential in understanding the design decisions taken during this process thereby serving as a medium for the stakeholders and project developers to communicate. Several frameworks [1, 2, 3, 4, and 5] exist for designing the architecture of the system. All the frameworks specify representing the system in several viewpoints in accordance to the stockholder's concerns. The typical job of an architect includes meeting the stakeholder concerns, making a collection of architectural requirements in various forms, turning them into quality scenarios and then architecting the system through various architectural styles, patterns and available architectural knowledge such that the constraints are met. The most important deliverable of the process of designing the architecture is the architecture document describing the structure of the system through its various views. This document is used to communicate to the customer, for analysis of various quality attributes, development and future maintenance. In essence they form the pivot around which all further activity about the software revolves.

In Software organizations where a large number of projects are documented daily, preserving the knowledge of the architectures for further reference plays a major role. Architectural level re-use is another activity that is impacted by documentation. Our previous work addresses these problems [11] and provides a new way of archiving the architecture documents and extracting these documents with a comprehensive search. In that approach, we have tried to annotate the architecture documents with architectural properties like styles, patterns, tactics, domain, technology components, quality requirements, and other framework standards. We represented this metadata as an XML file and search is performed on this metadata as an XML tag search.

Several relationships exist between the patterns and styles, which in turn relate to the problem domain of the system. For instance, Real time systems mainly deal with concurrency patterns, resource patterns, and safety and reliability patterns. Similarly financial and accounting systems have their own best practices defined. The distributed and layered architecture styles use certain pre-defined design patterns to solve the problems that might occur with regards to distributed communication, preserving data integrity, structuring the application logic etc. Suppose that, we try to find some architecture documents using layered architecture style, we might generally look for architecture documents using the patterns like structuring application logic, domain logic, application logic etc. These relationships and the associated vocabulary form a huge knowledge database. A taxonomy or ontology of this knowledge base helps the architect in searching and retrieving architecture documents in a semantic way. Figure 2 depicts the conceptual overview of the Ontology we suggest for use in searching. The major concentration is on the architectural properties like problem domain, architecture styles, patterns, architecture tactics, quality requirements and the relationships that exist among them.

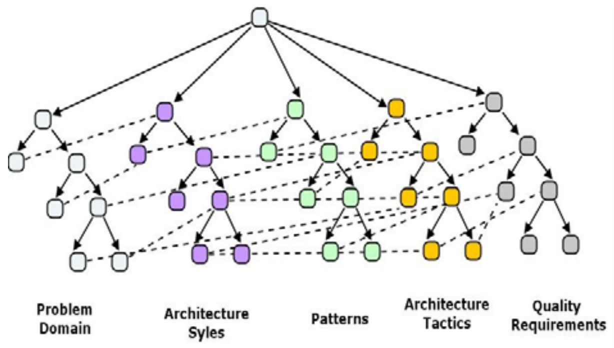


Figure 1: Conceptual overview Of Our Ontology

The rest of the paper is organized as follows. In the following section we provide an outline of the related work done in this area and briefly describe the motivation behind our work. Section 3 explains several architecture domain vocabularies that are used in this ontology and how these architectural properties help in retrieving the architecture documents. In section 4, we tabulate the relationships used between the terms in our ontology and briefly describe about how this ontology along with some applications of this Ontology. Finally in section 5, we conclude the paper by giving a brief outlook on future work.

2. Related Work

In Grady Booch's *Handbook of Software Architecture* [6], a large number of patterns are classified which allow comparisons across domains and architecture styles. However, he does not describe the relationships between architecture tactics and quality requirements, mapped to the real life problem domains, which can also be used for searching architecture documents. According to [8], a knowledge base is developed for representation and reuse of software patterns facilitating the semantic related search for the reuse of patterns. There is no such effort of mapping these patterns to other architecture properties, which will prove to be an efficient searching technique for architecture documents. Our approach to software architecture ontology is to provide a mapping between several architectural properties like patterns, styles, and tactics and problem domains. Figure 2 depicts several possible terms of the OntoSoftArch Ontology and the relationships between them as a concept map [9]

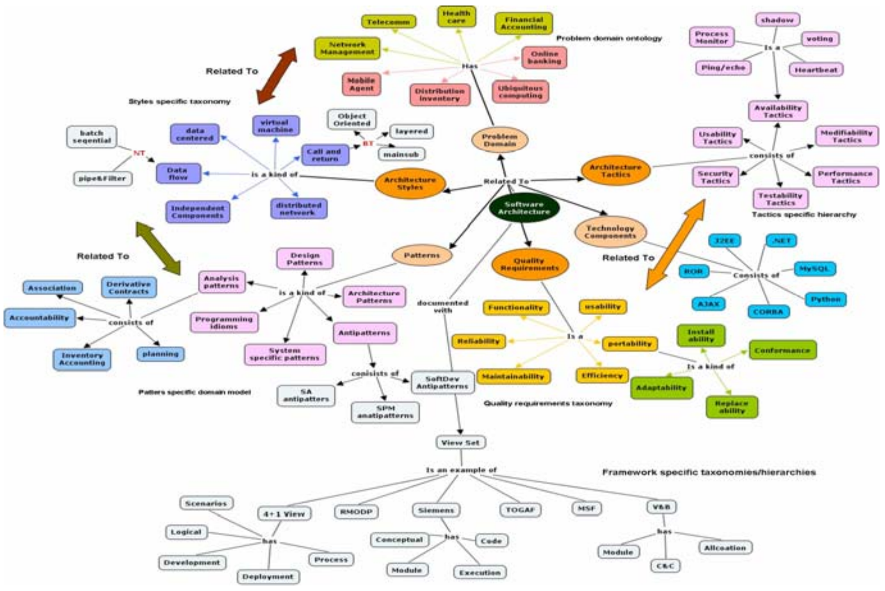


Figure 2: A fragment of Our Ontology as a Concept Map

3. Architectural Descriptions

We now describe the architecture properties that can be used in annotating the architecture documents. Our description regarding these architecture properties is necessarily brief and mainly concentrate on the search criteria of the documents.

Problem domain: A good architecture document depends on how well the domain model is identified and how well the commonality and the architect depicts variations among different instantiations of the system. Lack of domain knowledge in the architect can result into chaos during the project development. Problem frames [12], help in decomposing the problem into several sub problems but not help the architect in deriving the architecture of the system. Often the major concerns of the software architect vary from domain to domain. For example, for Telecommunication systems, that are distributed systems, solving the problems with distributed communication, configuration, session data storage, preserving data integrity etc. are the main concerns. Patterns such as cache proxy, Broker, Remote proxy, Client session state, Fine grained locking etc are used to solve such problems. There exist many similar types of relationships between problem domain and the patterns used. Similarly in the domain such as Aerospace and defense, quality requirements such as performance and reliability are the main concerns. We have identified such problem domains and the relationships with other architecture properties in our ontology. Often the architects are faced with queries like - can we retrieve the design documents with similar problem domain that is being worked upon? Such queries can be easily addressed while searching the repository if the architecture documents are annotated within this architecture property. The search will also be performed with the architecture properties related to this domain as explained above.

Technology components: The architecture includes hardware and software components that are not directly part of the actual design process. The design process helps in identifying various subsystems and the way they interact. These subsystems are thn mapped to technology components and they are related to each other in terms of interfaces they provide. The questions such as 1) can the architecture documents that use *MySql* database be retrieved on a Linux platform? 2) Can search be performed for the architecture documents that use the *Apache* webserver and *ODS* gateway with *CORBA* middleware and *Oracle* as the backend? Such queries can be easily answered if the architecture documents are annotated with all the technology components used in that system design. Often these technologies are related to best practices, for example the Yahoo UI library [10] is related to best practices like drag and drop, color picker, Image viewer. Similarly J2EE technology has some patterns such as Web Service Broker, Application Control, and Composite entity that are applicable in that domain. The same is the case with AJAX based applications. Many such relations are captured in our ontology; enabling search for architecture documents based on these technological issues as well as related best practices to these technologies.

Architecture Styles The architecture styles depicted in [13] are classified based on the characteristics such as data flow between components, call/return systems, data centered systems etc. Architecture styles in [27] are classified based on the views in which stakeholder is concerned. For example in an allocation view deployment and work assignment, styles are used. Several relationships exist between the architecture styles and the patterns used. For instance in layered architecture style, structuring the presentation logic, domain logic and the application logic are the main concerns. Several patterns such as Model view controller [26], packed abstraction controller, transaction script, operation script, virtual proxy are used to resolve such problems. Similarly several patterns classified under Message routing, message transformations, message channels and message end points used in information exchange mechanisms are used in several styles like event based systems, black board styles etc. Our explanation regarding these styles and the relationships to other architecture properties are necessarily brief. We have gathered different architecture styles and relationships with other architecture properties in our ontology. Queries such as can the architecture documents be retrieved with Interpreter and Rule based system styles? Can search be performed for architecture documents, which use Blackboard and Hypertext systems? Again such queries can be easily answered by annotating the documents with the corresponding styles used in the system.

Patterns: Application of best practices comes in the form of patterns. Several new patterns are evolving every year depending on the technologies developed. Gamma et.al [16] classified their patterns into three groups, Creational, Structural and behavioral. Tichy [17] gives a catalogue of over 100 patterns and arranged them under the categories like decoupling, state handling, virtual machine etc. Zimmer [18] analyzed the relationships between the patterns by Gamma. He introduced three kinds of relationships between patterns - X uses Y, X is similar to Y, X can be combined with Y. Architecture patterns catalogued by Buschman [19] also play a significant role in architecting the system. Also, a catalogue of 72 analysis patterns [20, 21] classified under Accountability, Association, Inventory and Accounting. There are also patterns associated with technologies such as CORBA, J2EE, AJAX [23, 24, 25] etc. Often the architects are left with the questions like - what is the consequence of using common interface and then wrapping it up to integrate it. Also, for example in Application Integration, how different applications should be integrated is the major concern. Patterns such as File transfer, Messaging, Remote procedure call, and shared database provide solutions to such design problems and annotating the documents with these patterns used in the system.

Architecture Tactics: An architecture tactic is a transformation of the system from one state to other that affects one of the parameters defined by quality attributes [15]. A large number of tactics have been identified and catalogued in Bass et al [15, 26]. The classified tactics are based on the quality attribute addressed. Their classification of patterns and tactics are based on the following relationship, Quality attributes → Tactics → Patterns. Consider an example of performance related tactics and patterns. Two patterns Flyweight and Thread pool pattern uses the same tactic, reduce computational overhead thereby meeting the quality requirement Such relationships between quality attributes, patterns and tactics are depicted in our ontology. Annotating the architecture documents with architectural tactics used while making architectural decisions helps to answer queries such as 1) did we use these tactics before and what was the result? The search will also be performed on the documents handling the quality attribute related with that tactic.

Quality Requirements: Quality requirements are the architecture drivers for any successful development of the system. The degree of quality achieved may vary from system to system. The Extended ISO model [28] depicts several quality attributes.

These are classified mainly based on reliability, usability, portability, efficiency and maintainability. Quality requirements and their attributes are defined in quality attribute theory [14, 28]. The purpose of the quality attribute theory is to enable the interpretation of software architecture in terms that are meaningful to quality attributes. Several relationships exist between quality attributes and tactics as described in the previous section. Also several relationships exist between patterns and quality requirements such as system performance patterns, and patterns for performance and reliability such as Fail over cluster, Load balancing cluster, Server cluster etc. We have defined such relations between quality attributes, patterns, tactics and problem domain in our ontology. Annotating the documents with the quality attributes handled in those systems helps the architect in full filling his queries like - can we retrieve the documents with throughput of the scenario between t1 and t2?

4. Ontology Relationships

Name	Description
Used-to	Denotes a means/ mechanism
Related-To	Denotes an association relationship
Is-A	Denotes a super, subclass relationship
Is – part - of / Has	Denotes an aggregation relationship
Is-Similar-To	Denotes an equivalence relationship
Requires	Denotes an association relationship

Table 1: Relationships between the Terms

Initially we gathered all the terms to software architecture from several architecture books and research papers. We also gathered some of the important index terms from some of the major architecture books [1, 15, 18, 19, and 21]. We manually went through the terms and refined the vocabulary. We found several relationships between the terms within the domain of each architecture property as well as the relationships between the architecture properties. We also included the relationships that are already between the terms like architecture tactics and patterns [15].

Currently our ontology consists of 1470 terms from all the architecture properties like problem domain, styles, patterns, tactics, technology components and several architecture frameworks. Identification of relationships between the terms is an ongoing effort, and we are augmenting and refining the relationships. More explanation of the relationship along with the Ontology is available to download from the URL http://www.cse.iitk.ac.in/~soft_arch/ontosoftarch. The viewer support tool for viewing this ontology is also available for download. We are trying to convert this ontology into an OWL based ontology, which allow users to load into any ontology editors like Protégé.

This ontology helps the architect in understanding the existing relationships between best practices thereby enabling him to construct a new system with existing best practices. Consider a scenario where an architect wants to develop a system for Financial and Accounting domain. This ontology helps him in understanding all the analysis accountability patterns related to that domain and the quality requirements to be considered for this system.

The ontology is useful for pedagogical purposes. This ontology helps the students to clearly understand all the terms and concepts in software architecture and the existing relationships between them.

A third application would be allowing the user to semantic search for the architecture documents which are annotated based on the above said architecture properties. Our previous work [29] talks about annotating the architecture documents with the architecture properties as an XML file, enabling the user to search for architecture documents based on the XML tags. In this search, the user will be giving one architecture property based on which search should be performed. The search will also be continued on the architecture properties related to the given architecture property. If the user gives the query like "Search for architecture documents, which used Client Server Architecture style", the search will also be performed on the architecture documents using the message exchange patterns in this architecture style. The related terms will be extracted from our ontology and searched in the repository.

5. Conclusions

An ontology of software architecture helps the architect in understanding the best practices used for documenting software architectures. Ontologies can also help in semantic annotations of architecture documents. Currently our ontology is populated with patterns, styles, tactics, domain concepts and different frameworks. The knowledge base contains the terms that are normally used in software architecture and relates them semantically, allowing effective searches and reuse of best practices. We plan to extend this ontology by adding more terms and more associations. This can make more inferences among the terms and a full-fledged ontology can be developed. Next step is the construction of CASE tool to allow semantic search for the architecture documents stored in the repository based on the semantic annotations.

References

1. Clements P, Bachmann F, Len Bass, David Garlan, James Ivers, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond.*
2. IEEE recommended practise for architectural description of software-intensive systems. *IEEE Standards*, pages 1–23, Sep 2000.
3. Hoffmesiter, C, Nord, R., Soni, D, Applied Software Architecture, Addison-Wesley, 2000 Kruchten, P
4. *Architectural Blueprints- The "4+1" View Model of Software Architecture*. Rational Software Corp., IEEE Software, November 1995.

5. RM-ODP Standards, ISO/IEC JTC1/SC21/WG7 Reference model of Open Distributed Processing available at http://archive.dstc.edu.au/AU/research_news/odp/refmodel/standards.html
6. Grady Booch's Handbook of Software Architecture
8. An Ontology-based Knowledge Base for the Representation and Reuse of Software Patterns Rosario Girardi and Alisson Neres Lindoso Federal University of Maranhão Campus do Bacanga, São Luís - MA, Brazi
9. Novak J.D., Cornell University *The Theory Underlying Concept Maps and How To Construct Them*, available at http://cmap.coginst.uwf.edu/info
10. http://developer.yahoo.com/yui/index.html.
11. Rambabu Duddukuri, Prabhakar T.V "On Archiving Architecture Documents'' In the Proceedings of APSEC 2005, Taipei, Taiwan.
12 Problem Frames: Analyzing and Structuring Software Development Problems by Michael Jackson
13 Shaw, M. "Making Choices: A Comparison of Styles for Software Architecture." *Carnegie Mellon University*, May 1994.
14 Barbacci, M.; Klein, M.; Longstaff, T.; & Weinstock, C. *Quality Attributes* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995.
15 Len Bass, Paul Clements, Rick Kazman, and Ken Bass. *Software Architecture in Practice*. 2nd Edition, Addison-Wesley
16 Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides; Addison-Wesley, 1995
17 A catalogue of General-Purpose Software Design Patterns, Tichy, Walter F , University of Karlsruhe, Karlsruhe,Germany. http://wwwipd.ira.uka.de/~tichy/publications/Catalogue.doc
18 Relationships between Design Patterns by Zimmer, Walter, in Pattern Languages of Program Design, James O. Coplien, Douglas C. Schmidt, Editors, Addison-Wesley, 1995
19 Frank Buschmann, Regine Meunier, Hans Rohnert,Peter Sommerlad, Michael Stal, Peter Sommerlad, and Michael Stal. Pattern-oriented software architecture.
20. Fowler, M. *Analysis Patterns*. Reading, Massachusetts: Addison-Wesley, 1997.
21. Fowler, M. *Patterns of Enterprise Application Architecture*. Reading, Massachusetts: Addison-Wesley, 2003.
22 William J. Brown, Raphael C. Malveau, Hays W. Skip McCormick (III), and Thomas J. Mowbray. Antipatterns: Refactoring software architectures and projects in crisis.
23 Alur, D., Crupi, J., Malks, D. *Core J2EE Patterns, 2nd ed.* Upper Saddle River, New Jersey: Prentice Hall, 2005.
24 Gross, C. *Ajax Patterns And Best Practices*. New York, New York: Springer-Verlag, 2006
25 Mowbray, T. & Malveau, R. *CORBA Patterns*. N6w York, New York: Wiley, 1997.
26 Design Patterns, Quality Attributes and Software Architectural Tactics, Felix Bachmann, Len Bass, Mark Klein.
27 Sun Microsystems. http://java.sun.com/blueprints/patterns/MVC-detailed.html, 2000.
28 Bob van Zeist, Paul Hendriks, Robbert Paulussen, and Jos Trienekens. Quality of software products — Experiences with a quality model. http://www.serc.nl/quint-book/index.htm.