The 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference

ISWC+ASWC 2007

Workshop 1

Service Matchmaking and Resource Retrieval in the Semantic Web

Workshop Organizers:

Tommaso Di Noia, Ruben Lara, Axel Polleres, Ioan Toma, Takahiro Kawamura, Matthias Klusch, Abraham Bernstein, Massimo Paolucci, Alain Leger, David Martin

11 Nov. 2007
BEXCO, Busan KOREA

ISWC 2007 Sponsor

Emerald Sponsor

Saltlux

Gold Sponsor

Google u-MTec KAIST VULCAN A Paul G. Allen Company

Kocb

Silver Sponsor

IBM Research NeOn EAST WEB

ASIA-LINK EUROPEAID CO-OPERATION OFFICE ETRI FRANZ INC. ADUNA IOS Press

We would like to express our special thanks to all sponsors

ISWC 2007 Organizing Committee

General Chairs

Riichiro Mizoguchi (Osaka University, Japan)
Guus Schreiber (Free University Amsterdam, Netherlands)

Local Chair

Sung-Kook Han (Wonkwang University, Korea)

Program Chairs

Karl Aberer (EPFL, Switzerland)
Key-Sun Choi (Korea Advanced Institute of Science and Technology)
Natasha Noy (Stanford University, USA)

Workshop Chairs

Harith Alani (University of Southampton, United Kingdom)
Geert-Jan Houben (Vrije Universiteit Brussel, Belgium)

Tutorial Chairs

John Domingue (Knowledge Media Institute, The Open University)
David Martin (SRI, USA)

Semantic Web in Use Chairs

Dean Allemang (TopQuadrant, USA)
Kyung-Il Lee (Saltlux Inc., Korea)
Lyndon Nixon (Free University Berlin, Germany)

Semantic Web Challenge Chairs

Jennifer Golbeck (University of Maryland, USA)
Peter Mika (Yahoo! Research Barcelona, Spain)

Poster & Demos Chairs

Young-Tack, Park (Sonngsil University, Korea)
Mike Dean (BBN, USA)

Doctoral Consortium Chair

Diana Maynard (University of Sheffield, United Kingdom)

Sponsor Chairs

Young-Sik Jeong (Wonkwang University, Korea)
York Sure (University of Karlsruhe, German)

Exhibition Chairs

Myung-Hwan Koo (Korea Telecom, Korea)
Noboru Shimizu (Keio Research Institute, Japan)

Publicity Chair: Masahiro Hori (Kansai University, Japan)

Proceedings Chair: Philippe Cudré-Mauroux (EPFL, Switzerland

Metadata Chairs

Tom Heath (KMi, OpenUniversity, UK)
Knud Möller (DERI, National University of Ireland, Galway)

Preface

One big challenge of service coordination in the Semantic Web is concerned with how to best connect the ultimate service requester with the ultimate service provider. Like intermediaries in the physical economy, a special kind of software agents, so called middle-agents, is supposed to solve this problem based on the declarative characterization of capabilities for both service requester and provider agents. In fact, the standard Web service interaction life cycle corresponds to a classical service matchmaking process. More generally, resource retrieval extends the notion of service matchmaking to the process of discovering any kind of resource (services, data, information, knowledge) for given settings, participating entities, and purposes. Such process is at the core of several scenarios in the Semantic Web area, going from Web-services, grid computing, and Peer-to-Peer computing, to applications such as e-commerce, human resource management, or social networks applications such as mating and dating services.

The papers accepted for the first edition of SMR^2 workshop try to cover several aspects related to semantic Web service (SWS) and resource retrieval. From SWS discovery to SWS composition – as done by Seiji Koide and Hideaki Takeda in **Formulation of Hierarchical Task Network Service (De)composition** – up to retrieval based on Natural Language queries (Serge Linckels et al., **Optimizing the Retrieval of Pertinent Answers for NL Questions with the E-Librarian Service**).

Many of the approaches presented, propose and analyze solutions and techniques mainly related to semantic Web service (SWS) discovery. Among various open issues related to the discovery phase of a SWS, a particular emphasis is given to: ***approximate retrieval*** – how to go over pure classification-based techniques to find SWS satisfying, to some extent, user requests?; ***ranking criteria*** – given a pull of retrieved SWS, how to evaluate the best ones?; ***efficiency*** – how to speed up the discovery phase?

Jacek Kpecký et al., in their **Semantic Web Service Offer Discovery**, describe a two phase discovery to find a Web service fulfilling user request and preferences. The first *static* phase uses coarse-grained semantic Web service descriptions to find services that approximatively match the users goal; the second *dynamic* one uses the Web service semantic description of the interface to find appropriate offers. Efficiency is the main concern of in **Efficient Discovery of Services Specified in Description Logics Languages** by Claudia d'Amato et al. Here a concept clustering approach – tailored for Description Logics based functional representation of a SWS – is proposed to speed up the retrieval process.

Alberto Fernandez et al. (**Towards Fine-grained Service Matchmaking by Using Concept Similarity**) analyze existing approaches for both concept similarity and Web service matching, based on the formal semantics behind ontology languages, and sketch some ideas on how to combine these techniques in a unified framework. Going beyond pure semantic-based approaches to matchmaking, in **Hybrid OWL-S Service Retrieval with OWLS-MX: Benefits and Pitfalls**, Matthias Klusch and Benedikt Fries mix logic- and IR-based matchmaking and present a thorough evaluation of their implemented system based on OWL-S SWS descriptions. A preliminary analysis of the performance for the hybrid retrieval system WSMO-MX is presented in **Performance of Hybrid WSML Service Matching with WSMO-MX: Preliminary Results** by Frank Kaufer and Matthias Klusch. Beside functional parts of a SWS description we do not have to forget non-functional ones such as Quality of Service (QoS). Kyriakos Kritikos and Dimitris Plexousakis (**OWL-Q fir Semantic QoS-based Web Service Description and Discovery**) focus on these properties by developing OWL-Q, an ontological specification for semantic QoS-based Web Service descriptions.

After all, we would like to see semantic Web services in action. In the direction of "real" semantic Web services Gennady Agre et al. (**Towards Semantic Web Service Engineering**) present a framework for semantic Web service engineering that covers the whole SWS life-cycle from creation to execution and monitoring. Results of an analysis on the current status of semantic Web services in the Web are in **Semantic Web Service in the Web: A Preliminary Reality Check** (by Matthias Klusch and Zhinguo Xing). In this paper the authors give an answer to the basic question: where are all the semantic Web services today?

As a last word, we are grateful to all authors for their valuable submissions, the members of the Program Committee and the external reviewers for their time and efforts, who all helped making this workshop a success. We are confident readers will find, through the papers, useful information on current state of the art on service matchmaking and resource retrieval in the semantic Web and useful hints to future research directions.

Abraham Bernstein, Tommaso Di Noia, Takahiro Kawamura, Matthias Klusch, Ruben Lara, Alain Leger, David Martin, Massimo Paolucci, Axel Polleres and Ioan Toma

SMR^2 PC chairs and organizers

Workshop Organization

Organizing Committee

Abraham Bernstein (University of Zurich, Switzerland)
Tommaso Di Noia (Technical University of Bari, Italy)
Takahiro Kawamura (Toshiba Research, Japan)
Matthias Klusch (DFKI, Germany)
Ruben Lara (AFI, Spain)
Alain Leger (France Telecom Research, France)
David Martin (SRI International, USA)
Massimo Paolucci (NTT DoCoMo Research Europe, Germany)
Axel Polleres (DERI Galway, Ireland)
Ioan Toma (DERI Innsbruck, Austria)

Programme Committee

Sudhir Agarwal (University of Karlsruhe, Germany)
Rama Akkiraju (IBM, USA)
Boualem Benatallah (The University of New South Wales, Australia)
Eugenio Di Sciascio (Technical University of Bari, Italy)
Stephan Grimm (FZI Karlsruhe, Germany)
Sung-Kook Han (Won Kwang University, Korea)
Frank Kaufer (University of Potsdam, Germany)
Uwe Keller (DERI Innsbruck, Austria)
Holger Lausen (DERI Innsbruck, Austria)
Freddy Lecue (Orange-France Telecom, France)
Terry Payne (University of Southampton, UK)
Euripides Petrakis (Technical University of Crete, Greece)
Fabio Porto (EPFL, Switzerland)
Christophe Rey (ISIMA, University of Clermont Ferrand, France)
Dumitru Roman (DERI Innsbruck, Austria)
Hideaki Takeda (National Institute of informatics, Japan)
Farouk Toumani (ISIMA, University of Clermont Ferrand, France)
Naohiko Uramoto (IBM Tokyo Research Laboratory, Japan)
Kunal Verma (University of Georgia, USA)
Takahira Yamaguchi (Keio University, Japan)

External Reviewers

Simona Colucci (Technical University of Bari, Italy)
Giuseppe De Giacomo (University of Rome "La Sapienza")
Michele Ruta (Technical University of Bari, Italy)

PROGRAM of the 1st International Joint Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web

[9:00 - 9:15] Opening

Session 1: Semantic Service Discovery

[9:15 – 10:00] Invited Talk by **Charles Petrie**

[10:00 – 10:30] Semantic Web Service Offer Discovery. *Jacek Kopecky, Elena Simperl, Dieter Fensel (DERI Innsbruck, Austria)*

[10:30 – 11:00] Coffee Break

[11:00 – 11:30] Efficient Discovery of Services specified in Description Logics Languages. *Claudia d'Amato (University of Bari, Italy), Steffen Staab (University of Koblenz-Landau, Germany), Nicola Fanizzi, Floriana Esposito (University of Bari, Italy)*
[11:30 – 12:00] Towards Fine-grained Service Matchmaking by Using Concept Similarity. *Alberto Fernandez, Axel Polleres, Sascha Ossowski (Universidad Rey Juan Carlos, Spain)*
[12:00 – 12:30] Hybrid OWL-S Service Retrieval with OWLS-MX: Benefits and Pitfalls. *Matthias Klusch (DFKI, Germany), Benedikt Fries*
[12:30 – 13:00] Performance of Hybrid WSML Service Matching with WSMO-MX: Preliminary Results. *Frank Kaufer (Hasso Plattner Institute, Germany), Matthias Klusch (DFKI, Germany)*

[13:00 – 14:00] Lunch

Session 2: Semantic Service Engineering and Applications

[14:00 – 14:15] Semantic Web Services in the Web: A Preliminary Reality Check. *Zhiguo Xing (Saarland University, Germany), Matthias Klusch (DFKI, Germany)*
[14:15 – 14:30] Towards Semantic Web Service Engineering. *Gennady Agre (IIT Bulgarian Academy of Sciences, Bulgary),Zlatina Marinova (OntoText Lab)*
[14:30 – 14:45] Formulation of Hierarchical Task Network Service (De)composition. *Seiji Koide, Hideaki Takeda (NII, Japan)*
[14:45 – 15:00] OWL-Q for QoS-based Semantic Web Service Description and Discovery. *Kiriakos Kritikos, Dimitri Plexousakis (Foundation of Research and Technology, Greece)*
[15:00 – 15:30] Optimizing the Retrieval of Pertinent Answers for NL Questions with the E-Librarian Service. *Serge Linckels (Hasso Plattner Institute, Germany), Harald Sack (University of Jena, Germany), Christoph Meinel (Hasso Plattner Institute, Germany)*

[15:30 – 16:00] Coffee Break

Session 3: Semantic Web Service Selection Tools: The S3 Competition 2007

[16:00 - 16:30] Short Demonstration & Results

Session 4: Panel & Open Discussion

[16:30 - 17:30] "Semantic Service Discovery - Quo Vadis?"

[17:30 - ...] Closing

Table of Contents

Semantic Web Service Offer Discovery

Jacek Kopecký, Elena Simperl, and Dieter Fensel

Digital Enterprise Research Institute (DERI)
Innsbruck, Austria
firstname.lastname@deri.at

Abstract. Semantic Web Services are a research effort to automate the usage of Web services, a necessary component for the Semantic Web. Traditionally, Web service discovery depends on detailed formal semantic descriptions of available services. Since a complete detailed service description is not always feasible, the client software cannot select the best service offer for a given user goal only by using the static service descriptions. Therefore the client needs to interact automatically with the discovered Web services to find information about the available concrete offers, after which it can select the best offer that will fulfill the user's goal. This paper shows when and why complete semantic description is unfeasible, it defines the role and position of offer discovery, and it suggests how it can be implemented and evaluated.

1 Introduction

The Semantic Web is not only an extension of the current Web with more semantic descriptions of data; it also needs to integrate services that can be used automatically by the computer on behalf of its user. A major technology for publishing services on the Web is the so-called Web services. Based on WWW standards HTTP and XML, Web services are gaining significant adoption in areas of application integration, wide-scale distributed computing, and business-to-business cooperation. Still, many tasks commonly performed in service-oriented systems remain manual (performed by a human operator).

In order to make Web services part of the Semantic Web, the research area of Semantic Web Services (SWS) aims to increase the level of automation of some of these tasks, e.g. discovering the available services and composing them to provide more complex functionalities. SWS automation is supported by machine-processible semantic Web service descriptions. Current state of the art in non-semantic service description is WSDL[1], which can describe the messages accepted and produced by a Web service, and the simple message exchanges (called operations) and all the necessary networking details. In effect, WSDL specifies a limited syntactical contract that the service adheres to. Semantic descriptions capture the important aspects of the meaning of operations and messages, generally in a formal language based on logics.

SWS descriptions are processed by a semantic execution environment (SEE, for instance WSMX [5]). A user can submit a concrete *goal* to the SEE, which then finds

[1] Web Service Description Language, `http://w3.org/TR/wsdl20`

and uses the appropriate Web services to accomplish the goal. SWS research focuses mainly on how the SEE “finds the appropriate Web service(s)”, as illustrated in Figure 1 with the first three SEE tasks.

In the figure, meant to be illustrative of the situation, rather than a real-world scenario, the user decides to lead a healthier life and wants to buy 2kg of fruit. The SEE will first discover any services that sell fruit (discarding the service that sells potatoes), then it will filter depending on the user’s constraints and requirements (the user doesn’t like peeling oranges), ranks the resulting services according to the user’s preferences (the user is a student and so prefers the cheaper options) and selects the one service that is invoked in the end. At any stage in the process, the user can be allowed to confirm the results.

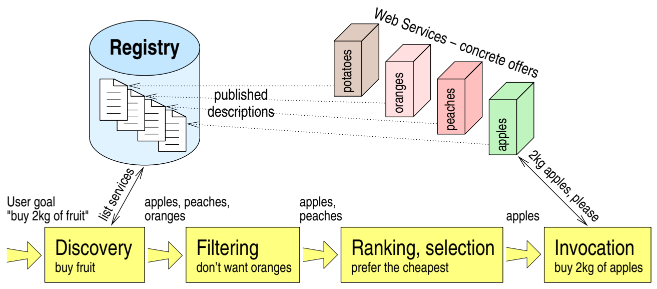


Fig. 1. Semantic Web Services automation tasks

We’ve chosen the simple fruit-shops example here to illustrate the situation in easily accessible terms, however, the general situation applies to any kinds of service: the existing services will publish their descriptions in a registry; then the SEE will *discover* the services applicable to the goal, *filter* and *rank* them according to any constraints and preferences specified by the user, and *invoke* the selected service to achieve the user’s goal.

The steps described above rely solely on the published Web service descriptions to find the best service that matches the user’s goal. Alas, in many cases it is not feasible to put all the relevant information in the service description, due to reasons detailed later in this paper. For instance, a grocery shop service would not list all the kinds of fruit they currently sell along with their up-to-date prices; instead, such a service would be described as “selling groceries”. This limits the scope of discovery based on static descriptions and introduces the need for an additional step, where the SEE will contact the discovered Web services (or their providers) to find out more about the service’s concrete offerings.

This additional step is called **offer discovery** (as opposed to Web service discovery). The objective of this step is to establish whether the discovered Web service can fulfill the user’s concrete goal and under what conditions. In our fruit shopping example, the SEE checks whether a grocery shop service carries any fruits, what sorts of fruits are available and at what prices, as shown in Figure 2. In this paper, we detail when and why static Web service discovery is not sufficient, we describe in detail what offer discovery needs to accomplish and how offer discovery approaches should be evaluated. We do not present a complete offer discovery solution in this paper, as our work is in an early stage.

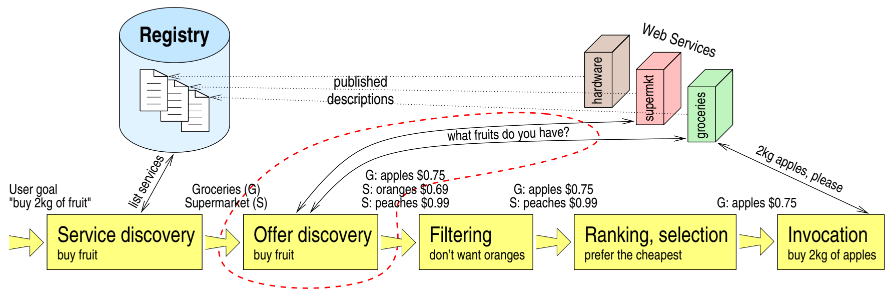


Fig. 2. Semantic Web Service offer discovery in SEE tasks

This paper is structured as follows: in Section 2 we detail the scenarios where offer discovery is necessary. Section 3 defines SWS offer discovery and relates it to other SEE tasks. Section 4 presents related work, both within Web services and in earlier research areas. In Section 5, we sketch the envisioned solution. Section 6 describes our expected evaluation methodology, and Section 7 contains concluding remarks.

2 Limitations of static Web service discovery

The best way to define service offer discovery is by describing the problems that it aims to solve. First, let us review the distinguishable functions of a semantic execution environment (SEE). The following steps are traditionally executed after a user submits their goal "buy 2kg of fruit", as shown in Figure 1.

1. **Web service discovery**[2] — using published descriptions, find all the available Web services that may sell fruits (the services may be more generic, like a supermarket with all kinds of products, or more specific, like an owner of a cherry-tree orchard, who naturally only offers cherries).
2. **Filtering** — filter out services that do not fit the user's constraints (for instance a service that sells oranges, because the user does not like them).
3. **Ranking, selection** — rank the remaining offers based on the user's preferences, for instance by price. The best-ranked service may be automatically selected, or the ranking may be presented to the user.
4. **Invocation** — use the selected service to achieve the goal (in our case, purchase the fruit).

[2] Sometimes, the term *discovery* is used to mean all the steps leading from a user's goal to a service that can fulfill it, i.e. everything but invocation. We choose a narrower definition of discovery which only does matchmaking on the available service descriptons.

There are also the additional steps of *mediation* and *service composition*, but they are not particularly relevant to offer discovery, even though they may interact with it.

The task sequence above is fully adequate when the service descriptions carry all the data relevant for the goal of the user. In a grid environment, a user might need processing services and storage services, and the descriptions will contain such classifications. In our fruit-buying scenario, the services need to advertise in their descriptions the particular kinds of fruit they sell and at their prices (for ranking).

A vast majority of currently available public Web services[3] provides only a limited and fixed number of offers (products, services): a fax service from `oneoutbox.com` has a single operation `S`endFax; the Amazon S3 service at `amazonaws.com` provides data storage and retrieval (two offers); or the typical stock quote services provide one offer, the current (delayed) price of any given stock. As they are described, the fax service works globally for any fax number, the S3 service works with any size and kind of data, and many stock quote services purport to know all stocks, therefore there are no discoverable limitations. The offers of these services are simple and generic.

On the other hand, there are services whose discoverable offers are of a finer granularity. The Amazon E-Commerce Service, for instance, gives access to all products sold on Amazon.com. Since Amazon cannot claim to sell all book and DVD titles, for instance, each book and DVD title and any other product becomes a separate offer. The service has operations for checking the availability or price of any given book title etc. On a similar note, a broadband internet provider only serves certain areas, and it provides operations to check availability at any given address.

Figure 3 illustrates how different kinds of services have widely different numbers of offers[4]. From the left side, a temperature conversion service has two operations for converting either way; a telephony service can have calling, voice mail, sms and a few other operations; a currency exchange service can recognize tens or hundreds of currencies; and on the higher end a hotel reservation service can offer tens of rooms a year ahead (each room on each day is an offer, making thousands of offers); and finally an online store easily offers upwards of a million products.

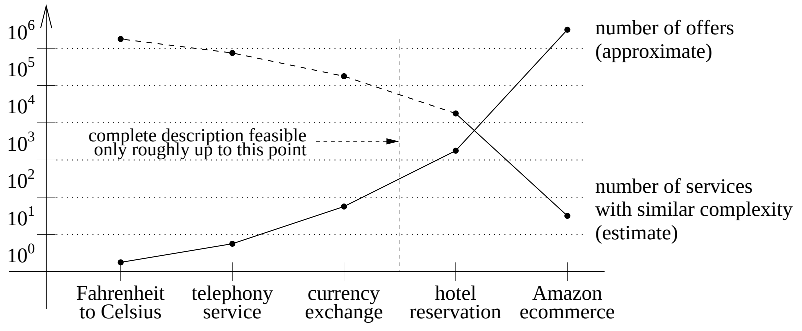


Fig. 3. Numbers of offers of various types of Web services

[3] Found through Web service registries `http://seekda.com` and `http://xmethods.net`
[4] The figure is only illustrative, it is not based on any concrete quantitative analysis.

The figure also shows an estimated potential number of distinct services with the indicated complexity (in terms of the number of offers). We can expect tens of online stores with millions of products, but already for the complexity of thousands of offers we can expect many more services — hotels and travel reservation services being prime examples. The large projected numbers of services with lower complexity can be justified by the vast variety and complexity of the Web and its diverse domains of discourse; nevertheless our uncertainty is higher on the left side of the curve of the potential number of services.

For simpler services, it is not a problem to publish all the relevant information in the semantic descriptions. Complete description becomes unwieldy for services on the right side of the graph: some currency exchange services do not bother to publish the list of supported currencies, hotels only publish up-to-date room availability to their close partners, and Amazon does not provide a "browse-all" functionality at all since it would be highly impractical. The dashed vertical line is a rough threshold above which complete description becomes infeasible.

The reasons against complete semantic description of all offers can be categorized as follows:

Processing performance: for a larger online store, the full product catalogue would make a Web service description impractical or impossible to process, considering current reasoning performance.

Description updates: updating the description in a service registry upon every inventory, availability or pricing change would lead to heavy resource utilization in the registries.

Trade secrets: a full description of service offers could even reveal sensitive stategy information or trade secrets.

While reasoning performance may improve, and registry updates can be optimized, sensitive information and trade secrets will not go away. For instance, a bank service description would have to detail all loan approval procedures in its complete offer description. For banks, the loan approval process with all its considerations is part of what makes some banks successful and others bankrupt. And sensitive information is not limited to such clear cases as banks. Even online retailers such as Amazon do not want to publish all offers, including bundle discounts (e.g., get Harry Potter 7 cheaper with other books in the series). Publishing all the prices and discounted offers in a single, easily accessible place, would provide the competition with insights into Amazon's strategy, and lower Amazon's competitive advantage.

In our experience, the complexity of complete service descriptions is a practical barrier to adoption of SWS technologies within the Web services industry. The need to maintain the complete descriptions, and to include possibly sensitive data, raises the barrier even higher. In other words, service discovery based solely on complete static semantic descriptions is of limited usefulness. On the other hand, less detailed "semantic-light" descriptions (for instance, Amazon would be described as selling books, movie DVDs, music CDs etc.) limit the SWS automation to simple but imprecise matchmaking and ranking.

These limitations of static semantic Web service discovery affect adversely the adoption of semantic technologies, and the lack of automation without semantics in turn lowers the adoption of Web service technologies themselves. While it may seem from Figure 3 that only a relative minority of Web services cannot be described completely, these complex services represent a significant economical value among the (potentially) available Web services.

3 Semantic Web Service offer discovery

As we have shown, static service discovery based on complete descriptions is, in many important cases, not feasible. Therefore, we split the task of finding the most appropriate offer from all the available Web services into static *Web service discovery* followed by dynamic *offer discovery*. The static Web service discovery uses coarse-grained semantic Web service descriptions to find services that potentially match the user's goal, and the dynamic offer discovery uses the semantic description of the Web service interface to automatically find any appropriate offers. With offer discovery, the set of steps can be rephrased as follows:

1. Web service discovery — find all the available Web services that may be able to fulfill the user's goal (i.e. discard those which, based on their description, cannot fulfill the user's goal).
2. Offer discovery — by interacting with the discovered services, find all their offers relevant for the goal.
3. Filtering — filter out offers that do not fit the user's constraints.
4. Ranking, selection — rank the remaining offers based on the user's preferences, and select one to be invoked.
5. Invocation — use the selected service.

Offer discovery can be seen as information retrieval (search) or as negotiation, as discussed in Section 4. Semantic offer discovery should be able to communicate with any Web service and find information about offers relevant to the user's goal. For communicating with the Web services, the offer discovery engine needs a description of the service interface (what operations it contains that can be used to gather offer information) and a description of the exchanged data, to understand the offers and be able to compare it with the goal. In other words, offer discovery needs different semantic description than Web service discovery; the latter needs to know what the service offers, whereas offer discovery needs to know how to talk to the service to get the information.

Seen as a black box, an offer discovery engine has as its inputs the user goal and the set of discovered Web services, and the output is the set of offers which should be of the same granularity as the user goal, even though the semantic description of the service is on a higher level of abstraction (more coarse-grained). For instance, the semantic description for the Amazon e-commerce service could say "this service sells books". For a concrete user goal "buy the last Harry Potter book", the offers could be "Harry Potter 7, Hardback, \$12.99" and "Harry Potter 7, Paperback, \$8.99".

Offer discovery complements Web service discovery in situations where the latter alone is not feasible. Our main hypothesis is that *the semantic description necessary for automated offer discovery is significantly easier to create and manage (and more acceptable) than the complete semantic description of all the offers*. A further hypothesis is that *the process of offer discovery is more efficient or on par with the processes of managing the complete descripton and reasoning with it in Web service discovery*. The terms “easy”, “acceptable” and “efficient” are defined more clearly in Section 6, where we propose evaluation criteria for offer discovery solutions.

We should note that what we call *offer discovery* is elsewhere in literature (e.g. [6]) called *service discovery*, making the distinction between a Web service and the service it actually provides. We prefer the term “offer” to avoid causing confusion due to overloading of the common word “service”.

4 Related work

Semantic Web service offer discovery, as defined in the preceding section, is related to earlier research in automated negotiation, and to the related areas of query processing and information gathering.

The term *negotiation* has been used for different purposes in a variety of computer science fields, e.g. electronic commerce, grid computing, distributed artificial intelligence and multi-agent systems. In electronic commerce, Beam and Segev [2] define negotiation as “the process by which two or more parties multilaterally bargain resources for mutual intended gain”. There are several different types of negotiations in e-commerce: auctions (multiple buyers bid for price), double auctions (both buyers and sellers bid for price, e.g. stock exchanges), one-to-one bargaining, and even catalogue provision (price fixed by seller). Offer discovery is similar to catalogue provision (offer discovery accesses and retrieves the relevant parts of the offer catalogue), but it could be extended in the direction of bargaining as well.

Research in query answering and information retrieval has dealt, among others, with using multiple information sources to gather the requested (or relevant) information (cf. [7]), based on a user query. In Semantic Web services, a user goal can be seen as a form of query, and the discovered Web services (or their individual operations) as information sources. The particular problem in SWS offer discovery is the description of the services and their operations so that information retrieval techniques would be applicable.

We can see that offer discovery is not a problem specific to SWS. However, earlier efforts on similar automation (e.g. in multi-agent systems) have generally presumed a controlled environment with a predefined set of interaction protocols for various tasks; for instance, a marketplace would dictate a bargaining and auctioning protocol. Such an approach can be applied to Web services, however, a bargaining/auctioning protocol or a common query language would need to be standardized and adopted by most service providers. Any SWS offer discovery mechanism, together with any necessary semantic annotations mechanisms, would be different and novel because SWS offer discovery aims to be generic, independent of the domain of the service offers. Indeed, the semantic annotations should make the offer discovery algorithm adapt to any available negotiation or query protocol.

Apart from related work described above, we know of only one published attempt that involves dynamic offer discovery in Semantic Web Services: Zaremba *et al.* [10, 11] talk about a so-called “contracting interface” with a described choreography. In their case, the SEE client follows the predefined choreography to find out the concrete price offered by a discovered Web service. The contracting interface can be likened to a prescribed protocol for offer discovery.

5 Envisioned solution components

While we do not have a working solution at this time, we can describe the major components necessary for any solution for semantic Web service offer discovery. Offer discovery takes a number of discovered Web services (or their descriptions, to be more precise) together with a user goal and returns a set of offers from these Web services. Any of the discovered Web services can provide any number of relevant offers, or none at all, and at least initially we can assume that offers from different services are independent. Therefore, we describe offer discovery in terms of dealing with a single service; if multiple services have come from Web service discovery, we can deal with each one of them separately. Offer discovery can be seen as a three-step process:

1. selecting the "offer-inquiry" operations from among all the operations of the discovered Web service;
2. planning the execution of some or all of these operations, based on what data is available in the user's goal and what data the operations return;
3. invocation of the selected operations according to the plan, translating the appropriate goal data into the appropriate XML messages.

The first step is necessary because we cannot assume that all the operations of any Web service can be used in offer discovery. Indeed, Web service interfaces often intermix operations for offer inquiry with operations that actually provide the resulting product or service, for instance a hotel service would mix the availability inquiry operations with the operations for booking rooms. For purpose of automatic invocation, we must select operations that are *safe* in the same sense in which the Web architecture [1] defines "safe interactions":

> A *safe interaction* is one where the agent does not incur any obligation beyond the interaction. An agent may incur an obligation through other means (such as by signing a contract). If an agent does not have an obligation before a safe interaction, it does not have that obligation afterwards.[5]

To recognize the "offer-inquiry" operations, we need semantic annotations about the nature of operations. In step with the Web architecture, WSDL 2.0 has a mechanism for annotating Web service operations as safe; it is unclear whether more information would be necessary for selecting "offer-inquiry" operations; if so, they can be added

[5] A canonical example of a safe interaction is information retrieval — the client may query a service about the availability of a hotel room, yet by issuing the query the client makes no commitment to book the room. Note that safety is not the same as idempotency: safe operations are generally idempotent, but idempotent operations need not be safe — for example a delete operation on a data store is idempotent but not safe.

using SAWSDL[6] (Semantic Annotations for WSDL and XML Schema), a specification from the W3C; specifically using `modelReference` on WSDL operations. It remains to be seen whether all safe operations can be seen as “offer-inquiry” operations, but due to their safety, there is no harm in invoking such operations even if they do not actually help get information about the service offers.

When we have selected the suitable operations, we can use information gathering and planning techniques in the second step to plan an execution that will return relevant information about the offers pertinent to the user’s goal. We do not have a firm definition of *relevant* at the moment, and we expect that this process can even involve some heuristics for optimizing the interactions with the Web service. Retrieving too little information will give us an incomplete view of the offers, whereas retrieving too much information would be overhead and a potential performance problem. This step requires semantic annotations of the operation inputs and outputs; in other words, what parameters the operations require and what kind of information they return. Such annotations can be added as SAWSDL `modelReference`s on the XML schema elements that are the input and output messages of the Web service operations.

The third step actually executes the plan and invokes the operations. Its role is to ground the goal data (presumably in a Semantic Web language, e.g. RDF or in WSML [4]) to the XML messages expected by the operations, and interpret the returned XML messages as semantic data about offers. This step can probably be implemented with simple reuse of some automatic Semantic Web service invocation mechanism that implements the grounding (cf. [8]), and it needs annotations that specify the data grounding transformations.

6 Evaluating SWS offer discovery approaches

The expected end contribution of our work is an efficient approach to automatic offer discovery that complements Web service discovery based on static descriptions. Even though we do not yet have a concrete solution, we can sketch the ways in which we expect to evaluate it. The evaluation criteria listed below are independent of the details of any proposed solution.

The efficiency of an offer discovery approach is evaluated in two dimensions: volume and complexity of the necessary semantic description, and the performance and scalability of the discovery process, as described below. The offer discovery engine will form a part of a semantic execution environment (SEE). The use cases for a SEE include the common scenarios of electronic shopping and travel scheduling[7], but also existing real-world applications such as e-Government Emergency Planning, as described in [3], and any such use cases should be helpful in testing offer discovery.

The major benefit of the presence of offer discovery in a SEE is that the semantic descriptions of Web services need not be complete and detailed (e.g. the whole catalogue of an online store). This guides the first evaluation criterion: *the semantic description necessary to enable automated offer discovery must be significantly simpler than a complete and detailed static semantic description of the service.* The relative simplicity of

[6] http://w3.org/TR/sawsdl
[7] See http://sws-challenge.org/

the semantic descriptions can be tested using reasoning performance comparisons, and user evaluations of the process of creating and managing the full static description vs. the descriptions necessary for automatic offer discovery. As these evaluations require experts, instead of a simple survey with many participants who may not be so expert, we suggest to use the Delphi method [9] which is shown to have good results with fewer participants, even though the process is more time consuming.

Apart from the complexity of the semantic descriptions, offer discovery requires interaction with the Web services, whereas static service discovery based on complete semantic-heavy descriptions requires that the service provider updates the description on every change. Therefore, the second evaluation criterion is: *the reasoning and networked interaction during offer discovery should have better performance and scalability than the combination of reasoning with complete descriptions and network interactions for description updates*. Performance can be compared on specific test cases, and scalability needs to evaluate how the approaches can deal with many services and many service offers. Further, the comparison combines reasoning tasks with network interactions; therefore it is crucial to evaluate different settings of reasoning power vs. networking setup.

In short, the evaluation of SWS offer discovery approaches is in comparison to static service discovery with complete descriptions (complete enough to get comparable results), and it involves experiments in controlled environments for comparing the performance, and expert surveys for comparing the relative simplicity and maintainability of the involved semantic descriptions.

7 Conclusions

Since Web service discovery cannot always be based on complete and detailed semantic description, it needs to be complemented with automatic offer discovery. In this paper, we have described the problem, sketched the components of a solution and the evaluation methodology. Eventually, we intend to develop an approach to SWS offer discovery that will significantly simplify the needed semantic descriptions and thus help ease the adoption of SWS technologies in the industry.

Any offer discovery approach needs to answer the following major questions: how should the user goal and concrete offers be modeled semantically to enable a generic algorithm for offer discovery; and how to select Web service operations that can be used for retrieving information about concrete offers pertaining to the user goal, plus how to sequence the invocations of these operations. There are two concrete steps ahead of us now: we intend first to work on a prototype whose function will help us understand and formalize offers and goals; when these terms are formalized, we can proceed to specify and evaluate a concrete fully-fledged approach to offer discovery.

References

1. Architecture of the World Wide Web. Recommendation, W3C, December 2004. Available at http://www.w3.org/TR/webarch/.

2. C. Beam and A. Segev. Automated negotiations: A survey of the state of the art. *Wirtschafts-informatik*, 39(3):263–268, 1997.
3. R. Davies. Summative report on potential applications of SWS in eGovernment, 2006. Deliverable D9.16, project DIP (FP6 - 507483), available at `http://dip.semanticweb.org/documents/D916SummativeRpt_potentialAppssSWS_EGov_final.pdf`.
4. J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The Web Service Modeling Language WSML: An Overview. In *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *Lecture Notes in Computer Science, LNCS*. Springer, 6 2006.
5. A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX – A Semantic Service-Oriented Architecture. *International Conference on Web Services (ICWS 2005)*, July 2005.
6. U. Keller, R. Lara, H. Lausen, and D. Fensel. Semantic Web Service Discovery in the WSMO Framework. In J. Cardoses, editor, *Semantic Web: Theory, Tools and Applications*. Idea Publishing Group, 2006.
7. C. A. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proc. of the 14th Int'l Joint Conference on Artificial Intelligence*, pages 1686–1693, 1995.
8. J. Kopecký, D. Roman, M. Moran, and D. Fensel. Semantic Web Services Grounding. In *Proc. of the Int'l Conference on Internet and Web Applications and Services (ICIW'06), Guadeloupe*, February 2006.
9. H. A. Linstone and M. Turoff, editors. *Delphi Method: Techniques and Applications*. Addison-Wesley, 1975.
10. T. Vitvar, M. Zaremba, and M. Moran. Dynamic service discovery through meta-interactions with service providers. In E. Franconi, M. Kifer, and W. May, editors, *ESWC*, volume 4519 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2007.
11. M. Zaremba, T. Vitvar, M. Moran, and T. Hasselwanter. WSMX Discovery for SWS Challenge. SWS Challenge Workshop, Athens, Georgia, USA, November 2006.

Efficient Discovery of Services Specified in Description Logics Languages

Claudia d'Amato[1], Steffen Staab[2], Nicola Fanizzi[1], and Floriana Esposito[1]

[1] Department of Computer Science, University of Bari, Italy
{claudia.damato, fanizzi, esposito}@di.uniba.it
[2] ISWeb, University of Koblenz-Landau, Germany
staab@uni-koblenz.de

Abstract. Semantic service descriptions are frequently given using expressive ontology languages based on description languages. The expressiveness of these languages, however, often implies problems for efficient service discovery, especially when increasing numbers of services become available in large organizations and on the Web. To remedy this problem, we propose an efficient service discovery/retrieval method grounded on a conceptual clustering approach, where services are specified in Description Logics as class definitions [10] and they are retrieved by defining a class expression as a query and by computing the individual subsumption relationship between the query and the available descriptions. We present a new conceptual clustering method that constructs tree indices for clustered services, where available descriptions are the leaf nodes, while inner nodes are intensional descriptions (generalization) of their children nodes. The matchmaking is performed by following the tree branches whose nodes might satisfy the query. The query answering time may strongly improve, since the number of retrieval steps may decrease from $O(n)$ to $O(log\ n)$ for concise queries. We also show that the proposed method is sound and complete.

1 Motivation

First research efforts in the fields of service discovery and service matchmaking have been concentrated on setting up methods and formalisms for describing the service semantics, with the goal of making service retrieval an automatic or, at least semi-automatic, task. Expressive ontology languages such as OWL-DL[3] are increasingly used to describe the ontology of a domain as well as specific resources available in such a domain such as Web services [16, 10]. Thereby such resources may be described as parts of an ABox, i.e. as instances of concept expressions [16]. Frequently, however, it is also useful to fully exploit Description Logics (DLs) subsumption capabilities using concept expressions in a TBox, e.g. to describe all the possible concretizations of a specific Web service resource (as proposed in [10]) using TBox concept expressions to represent the generic Web service resources (and their instance to represent Web service instantiations). Eventually, the services are retrieved by formulating a request as a concept expression and checking each service description with regard to instantiation or

[3] www.w3.org/2004/OWL/

subsumption of the query. However, these approaches suffer heavily from runtime inefficiency when the number of available services grows, since the number of instantiation or subsumption checks grows linearly in the number of the available resources.

In spite of such limitations, most of the current researches that concentrate on the automation of the service discovery and retrieval focus on the improvement of the effectiveness of the matchmaking process rather than on the efficiency of the whole retrieval process. Specifically, central to the majority of the current service matchmaking approaches is that the formal semantics of service descriptions is explicitly defined in an ontology language such as OWL-DL [12] based on some decidable DLs [1]. In this way service matchmaking can be performed by exploiting standard DL inferences [16, 22, 7, 13] sometimes jointly with the use of syntactic similarity measures [14].

Less attention has been dedicated to the improvement of the efficiency of the service discovery task. In this paper we address such a problem, by proposing an *original tree indexing method, **DL-tree**, for services described as DLs concept expressions* and referring to an ontology acting as common knowledge base. The DL-tree is obtained by exploiting *a novel conceptual clustering algorithm for concept expressions*, **DL-link**, grounded on the use of *a new, truly semantic similarity measure*, **GCS-based similarity**.

In the DL-tree, available service descriptions are found at the leaf nodes, while inner nodes are generalizations of their children nodes. Germane to the heuristic construction of DL-tree are non-standard inference procedures for computing the Good Common Subsumer [3] of $\mathcal{ALE}(\mathcal{T})$ concept expressions (see Sect. 2). Hence, in this paper, we focus on the description and evaluation of services specified in this ontology language[4]. Since the DL-tree is computed on the ground of the GCS-based similarity, it is different from the subsumption-based taxonomy representing the ontology.

Once that the DL-tree for a set of service description is obtained, its structure may be used to focus subsumption-based matching to the branches of the tree where nodes satisfy subsumption (or instantiation) checks. By this way, we achieve a drastic reduction of the size of the retrieval space. Query answering time may strongly improve, since the number of retrieval steps may decrease from $O(n)$ to $O(log\ n)$ for concise queries and n resources. Because of the way that the DL-tree is constructed it heuristically achieves a good covering of the retrieval space while maintaining soundness and completeness of the retrieval method.

In the following, we first summarize some important definitions of the DLs representation languages, $\mathcal{ALC}$ and $\mathcal{ALE}$, and some established reasoning procedures that we use subsequently (Sect. 2). We define a semantic similarity measure, *GCS-based similarity*, for $\mathcal{ALE}(\mathcal{T})$ in Sect. 3. The measure combines extensional size of concept expressions to reflect their model semantics and an intensional generalization of two concepts, their *Good Common Subsumer* (cf. [3]) to consider also the structure of the given ontology. The GCS-based similarity is used to cluster service descriptions into the new indexing structure, DL-tree (instead of the DAG given by the ontology), using a modified agglomerative clustering mechanism in Sect. 4. The DL-tree is exploited in the service retrieval procedure defined in Sect. 5. In Sect. 6 a preliminary experi-

[4] In the conclusion we will indicate some ways to generalize DL-tree to indexing of OWL-DL concept expressions and instances.

mental evaluation of the DL-tree indexing method is presented. Some related works are examined in Sect. 7 while conclusions and future extensions of DL-Tree indexing are presented in Sect. 8.

2 The Reference Representation Language

We recall the basics of $\mathcal{ALC}$ and $\mathcal{ALE}$ [1] logics which adopt constructors supported by the standard ontology languages (see the DL handbook [1] for a thorough reference). In DLs, descriptions are inductively defined starting with a set N_C of *primitive concept* names and a set N_R of *primitive roles*. The semantics of the descriptions is defined by an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set representing the *domain* of the interpretation, and $\cdot^{\mathcal{I}}$ is the *interpretation function* that maps each $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each $R \in N_R$ to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The *top* concept $\top$ is interpreted as the whole domain $\Delta^{\mathcal{I}}$, while the *bottom* concept $\bot$ corresponds to $\emptyset$. Complex descriptions can be built in $\mathcal{ALC}$ using primitive concepts and roles and the following constructors whose semantics is also specified. The language supports *full negation*, denoted $\neg C$ (given any description C), it amounts to $\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$. The *conjunction* of concepts, denoted $C_1 \sqcap C_2$, yields an extension $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$ and, dually, concept *disjunction*, denoted $C_1 \sqcup C_2$, yields the union $C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$. Finally, the *existential restriction*, denoted $\exists R.C$, is interpreted as the set $\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}((x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$ and the *value restriction* $\forall R.C$, has the extension $\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}((x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}})\}$.

$\mathcal{ALE}$ logic is a sub-language of $\mathcal{ALC}$ as only a subset of $\mathcal{ALC}$ constructors is allowed. Specifically, concept disjunction is not allowed and only the *atomic negation* can be used, namely complex concept descriptions cannot be negated.

The main inference in DLs is *subsumption* between concepts:

Definition 1 (subsumption). *Given two descriptions C and D, C* subsumes *D, denoted by $C \sqsupseteq D$, iff for every interpretation $\mathcal{I}$ it holds that $C^{\mathcal{I}} \supseteq D^{\mathcal{I}}$. When $C \sqsupseteq D$ and $D \sqsupseteq C$ then they are equivalent, denoted with $C \equiv D$.*

A *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ contains a *TBox* $\mathcal{T}$ and an *ABox* $\mathcal{A}$. $\mathcal{T}$ is the set of definitions $C \equiv D$, meaning $C^{\mathcal{I}} = D^{\mathcal{I}}$, where C is the concept name and D is its description. $\mathcal{A}$ contains assertions on the world state, e.g. $C(a)$ and $R(a, b)$, meaning that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. Subsumption based axioms ($D \sqsubseteq C$) are also allowed in the TBoxes as partial definitions. Indeed, $C \equiv D$ amounts to $D \sqsubseteq C$ and $C \sqsubseteq D$.

A related inference used in the following is *instance checking*, that is deciding whether an individual is an instance of a concept [1]. Concept subsumption and instance checking are examples of standard inference services in DLs. Besides of these, we also use non-standard inference services.

The most used non-standard inference services are the *Most Specific Concept* of an individual and the *Least Common Subsumer* of a given collection of concepts.

Definition 2 (Most Specific Concept). *Given an ABox $\mathcal{A}$ and an individual a, the most specific concept of a w.r.t. $\mathcal{A}$ is the concept C, denoted $MSC_{\mathcal{A}}(a)$, such that $\mathcal{A} \models C(a)$ and $\forall D$ such that $\mathcal{A} \models D(a)$, it holds: $C \sqsubseteq D$.*

In the general case of a cyclic ABox expressed in an expressive DL endowed with existential or numeric restriction, the MSC cannot be expressed as a finite concept description [1], thus it can only be approximated. Generally an approximation of the MSC is considered up to a certain depth k. The maximum depth k has been shown to correspond to the depth of the considered ABox, as defined in [17]. We will indicate generically an approximation to the maximum depth with MSC^*.

Definition 3 (Least Common Subsumer). *Let $\mathcal{L}$ be a description logic. A concept description E of $\mathcal{L}$ is the **least common subsumer (LCS)** of the concept descriptions $C_1, \cdots, C_n$ in $\mathcal{L}$ ($LCS(C_1, \cdots, C_n)$ for short) iff it satisfies:*

1. *$C_i \sqsubseteq E$ for all $i = 1, \cdots, n$ and*
2. *E is the least $\mathcal{L}$-concept description satisfying (1), i.e. if E' is an $\mathcal{L}$-concept description satisfying $C_i \sqsubseteq E'$ for all $i = 1, \cdots, n$, then $E \sqsubseteq E'$.*

Depending on the DL language, the LCS need not always exist. If it exists, it is unique up to equivalence. In the case of $\mathcal{ALC}$ and $\mathcal{ALE}$ logic, the LCS always exists [2, 1]. Specifically, in the case of $\mathcal{ALC}$ (as in every DL allowing for concept disjunction) the LCS is given by the disjunction of the considered concepts, thus it is also "uninteresting" as it does not reflect any ontological modeling decisions. In the case of $\mathcal{ALE}$ logic, where disjunction is disallowed, the LCS is computed by taking the common concept names in the concept descriptions (also in the concepts scope of universal and existential restrictions w.r.t. the same role), without considering the TBox (see [2] for more details).

The $\mathcal{ALE}$ LCS computed using such an approach often results to be very general. For this reason the notion of LCS computed w.r.t. the TBox[5] to which the concept definitions refer has been introduced [3].

Definition 4 (LCS w.r.t. a TBox). *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be DLs such that $\mathcal{L}_1$ is a sub-DL of $\mathcal{L}_2$. i.e., $\mathcal{L}_1$ allows for less constructors. For a given $\mathcal{L}_2$-TBox $\mathcal{T}$, let $\mathcal{L}_1(T)$*-concept descriptions *be those $\mathcal{L}_1$-concept descriptions that may contain concepts defined in $\mathcal{T}$. Given an $\mathcal{L}_2$-TBox $\mathcal{T}$ and $\mathcal{L}_1(T)$-concept descriptions $C_1, \ldots, C_n$, the least common subsumer (LCS) of $C_1, \ldots, C_n$ in $\mathcal{L}_1(T)$ w.r.t. $\mathcal{T}$ is the most specific $\mathcal{L}_1(T)$-concept description that subsumes $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$, i.e., it is an $\mathcal{L}_1(T)$-concept description D such that:*

1. *$C_i \sqsubseteq_{\mathcal{T}} D$ for $i = 1, \ldots, n$*
2. *if E is an $\mathcal{L}_1(T)$-concept description satisfying $C_i \sqsubseteq_{\mathcal{T}} E$ for $i = 1, \ldots, n$, then $D \sqsubseteq_{\mathcal{T}} E$*

Specifically, in [3], the case of $\mathcal{L}_2 = \mathcal{ALC}$ and $\mathcal{L}_1 = \mathcal{ALE}$ has been focused and it has been proved that the $\mathcal{ALE}$ LCS w.r.t. an *acyclic* $\mathcal{ALC}$ TBox[6] always exists, while it cannot exist in case of *cyclic* or *general*[7] TBoxes. A brute force algorithm for computing the LCS w.r.t. a TBox has been also shown. Anyway, such an algorithm cannot be usable

[5] The TBox can be described by a DL that is more expressive than $\mathcal{ALE}$.

[6] A TBox is called acyclic if it does not contain any concept definition in which the concept name to define is also used in the concept definition. If this happens, the TBox is called cyclic.

[7] A TBox in which inclusion axioms are defined is called general TBox.

in practice. For this reason, an algorithm for computing an approximation of the $\mathcal{ALE}$ LCS w.r.t. an $\mathcal{ALC}$ TBox has been presented. The computed approximation is called *Good Common Subsumer* (GCS) w.r.t. a TBox and it can exist also when a general TBox is considered. The GCS is computed by determining the smallest conjunction of concept and their negations that is able to subsume the conjunction of the top level concept names of each considered concept, and the same for the concepts that constitute the range of existential and universal restrictions w.r.t. the same role. The GCS is more specific than the LCS computed by ignoring the TBox, though in general it subsumes (or is equivalent to) the least common subsumer w.r.t. the TBox (see [3] for more details).

3 GCS-based Similarity: A Semantic Similarity Measure for $\mathcal{ALE}(\mathcal{T})$ Concept Descriptions

In the last few years, several measures for assessing the similarity value between concepts have been proposed in the literature. From them, two main approaches can be distinguished: 1) measures based on semantic relations (also called path distance measures); 2) measures based on concept extensions and *Information Content*.

In the former approach all concepts are in an is-a taxonomy, and the similarity value between two concepts is computed by counting the (weighted) edges in the paths from the considered concepts to their most specific ancestor. Concepts with a few links separating them are similar; concepts with many links between them are less similar [23, 15, 6, 18].In the latter approach the similarity value is computed by counting the common instances of the concept extensions [8] or by measuring the variation of the *Information Content* between the considered concepts [9, 24, 5].

Since the ontology does not have the simple structure of a taxonomy, but it is rather an elaborated DAG, similarity measures based on distances in the taxonomy cannot be used. Furthermore, in the considered application domain, the typical scenario consists of having a set of concept descriptions (the service descriptions) with no one necessarily overlapping the others. Consequently also measures based on overlap of concept extensions as well as measures based on *Information Content* cannot be used, since also semantically similar concept could result to be totally different.

Hence, we define a new semantic similarity measure, the **GCS-based similarity**, able to cope with the presented scenario. Moving from the principles of the measures based on concept extension and Information Content, we propose a similarity measure exploiting the notion of concept extension, but instead of counting the common instances between two considered concepts, the similarity value is given by the variation of the number of instances in the concept extensions w.r.t. the number of instances in the extension of their common super-concept. The common super-concept is given by the GCS of the concepts (see Sect. 2). The measure is formally defined in the following.

Definition 5 (GCS-based Similarity Measure). *Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox. For all C and D $\mathcal{ALE}(\mathcal{T})$-concept descriptions, the* Semantic Similarity Measure *s is a function $s : \mathcal{ALE}(\mathcal{T}) \times \mathcal{ALE}(\mathcal{T}) \to [0, 1]$ defined as follow:*

$$s(C, D) = \frac{\min(|C^I|, |D^I|)}{|(GCS(C, D))^I|} \cdot (1 - \frac{|(GCS(C, D))^I|}{|\Delta^I|} \cdot (1 - \frac{\min(|C^I|, |D^I|)}{|(GCS(C, D))^I|}))$$

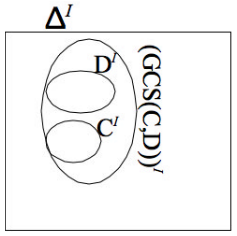


Fig. 1. Concepts C ≡credit-card-payment, D ≡debit-card-payment are similar as the extension of their GCS≡card-payment does not include many other instances besides of those of their extensions.

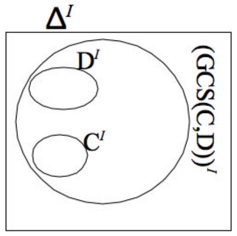


Fig. 2. Concepts C ≡car-transfer, D ≡ debit-card-payment are different as the extension of their GCS≡service includes many other instances besides of those of the extension of C and D.

where $(\cdot)^I$ computes the concept extension w.r.t. the interpretation I (canonical interpretation).

The canonical interpretation adopts the set of individuals mentioned in the ABox as its domain and the identity as its interpretation function [1, 17].

The rationale of the presented measure is that if two concepts are semantically similar, such as credit-card-payment and debit-card-payment, then they should have a good common superconcept, e.g. card-payment, that is so close to the two concepts, namely the extensions of the superconcept and even the lesser-sized input concept share many instances, consequently the similarity value will be close to 1. On the contrary, if the considered concepts are very different, e.g. car-transfer and debit-card-payment, their GCS, e.g. service, will be high up in the TBox, and this superconcept will have many instances not contained in the two compared concepts, consequently the similarity value will be next to 0. In Fig. 1 and Fig. 2 this rationale is illustrated. Opposite to existing semantic similarity measures, this rationale does not require overlap of compared concepts, and it does not take into account semantic path distance. Moreover, the minimum extension of the concepts is considered in the measure definition, in order to avoid to have an incorrect similarity value (high similarity value) in the case in which one of the concepts is very similar to the super-concept but very different from the considered one.

It is trivial to verify that the function s of Def. 5 is really a *similarity measure*, namely that (following the formal definition in [4]) it satisfies the following properties: 1) s is definite positive; 2) s is symmetric; 3) s satisfies the maximality property ($\forall\, C, D\ :\ s(C, D) \leq S(C, C)$).

4 *DL-Link*: A Conceptual Hierarchical Agglomerative Algorithm for Clustering Description Logic Resources

The main goal of Cluster Analysis is to set up methods for the organization of a collection of unlabeled resources into meaningful clusters exploiting a similarity criterion. Specifically, clusters (classes) are collections of resources whose intra-cluster similarity is high and inter-cluster similarity is low. The methods that focus also on techniques for generating intensional cluster description are called *Conceptual Clustering methods*.

A prominent conceptual clustering algorithm is the *average-link* algorithm [26]. It starts by assigning each resource to a distinct cluster and iteratively merges the two most similar clusters until only one cluster remains. The output of the algorithm is a dendrogram, namely a tree structure representing a nested grouping of resources. We modify average-link in several ways, in particular we substitute the typical Euclidean distance measure by the GCS-based similarity and we substitute the computation of the Euclidean average of each cluster in the computation of the GCS of the merged clusters.

Resources are assumed to be described as $\mathcal{ALE}(\mathcal{T})$ concepts, exploiting a common vocabulary $\mathcal{T}$ (mainly a shared ontology) written in $\mathcal{ALC}$ logic. They are clustered by a batch process by the use of the **DL-Link** algorithm detailed in the following.

Let $S = \{S_1, \dots, S_n\}$ be a set of available resources.

```
1. Let $\mathcal{C} = \{C_1, \dots, C_n\}$ be the set of initial clusters obtained by considering each resource as a single cluster
2. $n := |\mathcal{C}|$
3. For $i := 1$ to $n - 1$ consider cluster $C_i$
   – For $j := i + 1$ to $n$ consider cluster $C_j$
      • compute the similarity values $s_{ij}(C_i, C_j)$
4. compute $\max_{hk} = \max_{i,j=1,\dots,n} s_{ij}$ where $h$ and $k$ are the clusters with maximum similarity
5. create the intensional cluster description $C_m = GCS(C_h, C_k)$
6. set $C_m$ as parent node of $C_h$ and $C_k$
7. insert $C_m$ in $\mathcal{C}$ and remove $C_h$ and $C_k$ from $\mathcal{C}$
8. if $|\mathcal{C}| \neq 1$ go to 2
```

DL-Link algorithm starts by considering each service description in a single cluster (list of available clusters), hence the similarity value[8] between all couples of clusters is computed and the couple having the highest similarity is selected. Then a new description, generalizing the chosen clusters, is created by computing the GCS (see Sect. 2). The new description is first set as parent node of the chosen clusters, then it is inserted in the list of the available clusters while the selected ones are removed. The generated description represents a cluster made by a single element while the resources that it describes are represented as its children in the dendrogram under construction. We call such a dendrogram **DL-Tree**. The same steps are iteratively repeated, until a unique cluster (describing all resources) is obtained.

An example of the DL-Tree returned by the DL-link algorithm is shown in Fig.3. It is a binary tree and this is because, at every step, only two clusters are merged[9]. Anyway, it can happen that two children nodes (or more than two) of the DL-Tree have the same intentional description as well as it can happen that a parent node has the same description as a child node. Since such redundant nodes do not add any information, a

[8] The GCS-based similarity measure presented in Sect. 3 is used.

[9] The clustering process could be speeded up by finding a way for merging more than two clusters at every step.

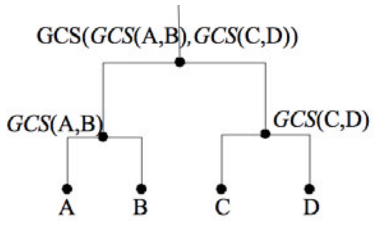


Fig. 3. DL-Tree index (dendrogram) returned by the hierarchical clustering algorithm presented in Section 4 applied to the four service descriptions respectively labeled by A, B, C, and D. *GCS* is the *Good Common Subsumer* of two $\mathcal{ALE}(\mathcal{T})$ descriptions.

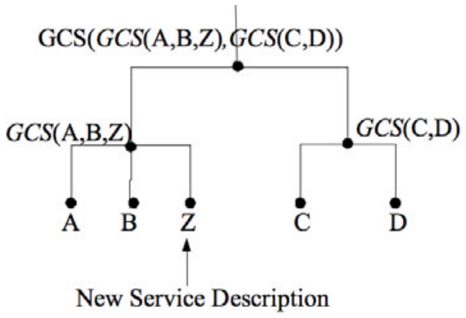


Fig. 4. **Z** is a new service description occurring after the DL-Tree construction. Once that the most similar service description **B** is found, **Z** is added has sibling node of **B** and the parent node of **B** is recomputed, as well as all parent node in the path, until the root node.

post-processing step is applied to the DL-Tree. If a child node is equal to another child node then one of them is deleted and their children nodes are assigned to the remaining node. If a child node is equal to a parent node then the child node is deleted and its children nodes are added as children of its parent node. The result of this flattening process is an n-ary DL-Tree.

It is important to note that, in case of a new service description occurs after performing the clustering process, the DL-Tree has not to be entirely re-computed. Indeed, the similarity value between the new service description and all available service descriptions (leaf nodes of the DL-Tree) can be computed and the most similar available service is selected. Hence the new service description can be added as sibling node of the most similar service while the parent node is re-computed as the GCS of the old child nodes plus the new child. In the same way all the ancestor nodes of the new generated parent node are computed. In this way a single path of the DL-Tree is updated rather than the entire tree structure. In Fig. 4 an example of updating of the DL-Tree illustrated in Fig. 3 is reported, in case of a new service description **Z** occurs. Obviously, after a certain number of changing of the DL-Tree, the clustering process have to be recomputed.

5 Resource Retrieval exploiting DL-Tree

Resource retrieval is performed by matching a given request with available resource descriptions. In this section we present an algorithm that, by exploiting the DL-Tree returned by DL-Link algorithm (see Sect. 4), is able to accelerate the resource retrieval task as well as to outperform the service discovery task. The rationale of the algorithm is to cut the search space in order to drastically reduce the number of necessary comparisons (matches) for finding resources satisfying a given request. The algorithm is presented in the following.

Let R be a request and let C the root of the DL-Tree $\mathcal{C}$

```
**retrievalProcedure**$(R, C)$

1. returnedResource := null
2. if **matchTest**$(R, C)$ = false then
     – return returnedResource
3. else if C is leaf node then
     – returnedResource.add($C$)
4. else
   (a) if $C$ has left child node $C_l$ then
         i. returnedLeftResource = **retrievalProcedure**$(R, C_l)$
        ii. if returnedLeftResource != null then
              – returnedResource.add(returnedLeftResource)
   (b) if $C$ has right child node $C_r$ then
         i. returnedRightResource = **retrievalProcedure**$(R, C_r)$
        ii. if returnedRightResource != null then
              – returnedResource.add(returnedRightResource)
5. return returnedResource
```

```
**matchTest**$(R, C)$

1. if ($R \sqsubseteq C$) then
     – return *true*
2. else
     – return *false*
```

The **retrievalProcedure** is performed by exploiting the DL-Tree whose leaf nodes are the available resource descriptions and every inner node (included the root node) is an intensional description of the child nodes. Given a request R, it is matched with the root node of the DL-Tree, in order to test if it can be satisfied by the available resources or not. Indeed, since the root node is the intensional description of all available resources, then if the match test is not satisfied (false returned value), this means that there are no available resources able to satisfy the request. On the contrary, if the test is satisfied, the match test is performed for each child node. If a child node does not satisfy the match subsumption condition, then the branch rooted in this node is discarded. Otherwise all

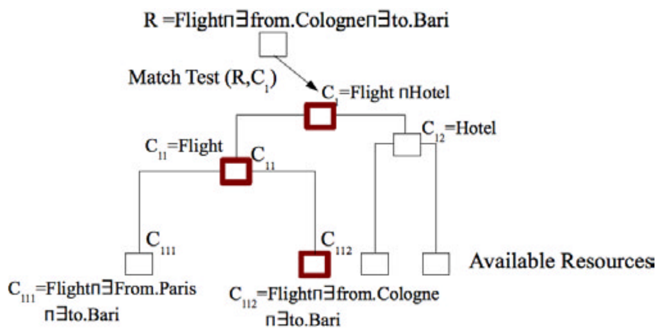

R =Flight⊓∃from.Cologne⊓∃to.Bari
Match Test (R,C1)
C1=Flight ⊓Hotel
C12=Hotel
C11=Flight
C11
C111
C112
Available Resources
C111=Flight⊓∃From.Paris ⊓∃to.Bari
C112=Flight⊓∃from.Cologne ⊓∃to.Bari

Fig. 5. Retrieval of an available service able to satisfy the request *R*. Bold boxes represent nodes satisfying the *MatchTest*.

the children nodes are recursively explored, until a leaf node is reached or until there are no child nodes satisfying the match condition.

As *MatchTest*, the *Entailment of Concept Subsumption* [16, 22] is used. It checks for subsumption, either of the requestor's description by the provider's or vice versa. Specifically, we check for the subsumption of description w.r.t. the request since we are interested in resources able to fully satisfy the request, while considering the vice versa, resources that only partially satisfy the request could be found.

It is important to note that, at the same level, more than one node could satisfy the match test. In this case all paths rooted in such nodes will be explored. As an alternative to such an exhaustive search, an heuristic could be adopted for suggesting the path to follow. As a possible heuristic, the path rooted in the node that is most similar to the request can be chosen. Differently from the previous case, in this way only one resource will be found, but it will be probably the most proper one.

The proposed method drastically reduces the size of the search space, since the search is restricted only to the branches of the DL-Tree whose nodes satisfy the match test. A good clustering of n available resources (i.e. a balanced hierarchy) may reduce the number of necessary matches for finding the right resource from $O(n)$ to $O(log\ n)$ (if the request is specific enough), thus strongly outperforming the response time for a request. An example of application of *retrievalProcedure* procedure is reported in Fig. 5.

Here we conclude with the following lemma.

Lemma 1. *Resource retrieval using DL-Link algorithm and DL-Tree indexing is sound and complete.*

6 Experimental Evaluation

In this section some preliminary experiments are illustrated in order to show that the method proposed in Sect. 5 really improves the resource retrieval task w.r.t. to the traditional approaches based on linear matching.

6.1 Data sets

For measuring the efficiency of the retrieval method, it has been applied to the SEMANTIC WEB SERVICE DISCOVERY DATA SET (SWS DISCOVERY DATA SET for brevity). It is a set of Semantic Web Services described by means of the DL-based framework presented in [10]. Since no existing data sets using such a framework were available, we have created a new one. Moving from the experience of OWLS-TC[10] that is a service retrieval test collection consisting of services specified by OWL-S 1.1, the SWS DISCOVERY DATA SET has been built. It consists of an $\mathcal{ALC}$ ontology representing the knowledge base of reference and a set of $\mathcal{ALE}(\mathcal{T})$ services described using such an ontology. The ontology models broad domains: bank domain, post domain, means of communication domain and geographical information. On the ground of such an ontology, 96 complex concept descriptions acting as service descriptions have been built. The resulting test set can be downloadable from https://www.uni-koblenz.de/FB4/Institutes/IFI/AGStaab/Projects/xmedia/dl-tree

6.2 Evaluation Methodology

The available service descriptions in SWS DISCOVERY DATA SET has been clustered by means of the DL-Link algorithm (see Sect. 4), obtaining as result a DL-Tree having, after the flattening post-processing, a depth equal to 7. This represents the maximum depth, while most of the leaf nodes in the obtained DL-Tree are at level 4. The average branching factor is equal to 5. In order to measure the efficiency of the retrieval method, different kinds of queries (requests) have been generated: 1) $\#\mathbf{96}$ queries corresponding to the leaf nodes of the tree (the actual service descriptions); 2) $\#\mathbf{20}$ queries corresponding to some inner node descriptions of the tree; 3) $\#\mathbf{116}$ queries randomly generated by disjunctions/conjunctions of primitive and/or defined concepts of the reference ontology and/or available service descriptions.

The efficiency of the method has been measured by counting the number of necessary comparisons (matches) in the tree for finding all available resources able to satisfy a request. Specifically, the average number of matches for each kind of query has been considered as well as the minimum and maximum number of matches. These values have been then compared with the number of checks in the linear retrieval case. Moreover, in both approaches, the average execution time for each kind of query has been measured, where the experiments have been performed on a laptop PowerBook G4 1.67 GHz 1.5 GB RAM.

6.3 Evaluation Results

Considering the queries generated as explained in the previous section and the DL-Tree obtained by clustering the SWS SERVICE DISCOVERY DATA SET, the retrieval procedure presented in Sect. 5 has been applied. The mean number of matches (subsumption checks) and the retrieval execution time have been measured. Specifically, for each query, the number of nodes visited (namely to which the subsumption check has

[10] http://projects.semwebcentral.org/projects/owls-tc/

Table 1. Number of comparison (average and range) and mean execution time for finding all the services satisfying a request w.r.t. the different kinds of requests both in the linear matching and in the DL-Tree based retrieval.

DATA SET	*Algorithm*	*Metrics*	*Leaf Node*	*Inner Node*	*Random Query*
SWS DISCOVERY DATA SET	**DL-Tree based**	*avg.*	**41.4**	**23.8**	**40.3**
		range	13 - 56	19 - 27	19 - 79
		avg. exec. time	266.4 ms.	180.2 ms.	483.5 ms.
	Linear	*avg.*	**96**	**96**	**96**
		avg. exec. time	678.2 ms.	532.5 ms.	1589.3 ms.

been applied) for finding all the services able to satisfy a request has been counted as well as the execution time has been measured. Hence the mean number w.r.t. all queries for a given kind (randomly, inner node, leaf node) has been computed.

The outcomes of the experiments are reported in Tab. 1. Looking at the table it is straightforward to note the our method requires a very low number of matches w.r.t. the linear approach. Specifically, independently to the kind of considered request, the DL-Tree based retrieval decreases the number of comparisons more than 50%. Since a lower number of comparisons means a decreasing of the response time, then our method is really able to improve the efficiency of the retrieval and discovery process. This is also evident looking at the average execution time of the two methods. Focussing on the experimental result and specifically on the mean execution time for the different kinds of queries, it is possible to note that querying for inner nodes requires the lowest execution time. This is because most of the resources satisfying the queries are at the highest levels of the DL-Tree, so finding them requires less execution time. Moreover the structure of such queries is very simple. Since the adopted match test is based on subsumption test (see Sect. 5), consequently the time necessary for checking for subsumption decreases when the query is simple. Conversely, querying for randomly generated queries requires the highest execution time. This is because the structures of such queries are more complex then those of leaf node queries and inner node queries (that are the simplest ones). Particularly, the benefits of our approach increase with the increasing of available services.

From the presented initial experiments it is possible to assert that our method really improve the efficiency of the resource retrieval task.

7 Related Work

Service discovery is the task of locating service providers that can satisfy the requester's needs. Generally, it is performed by matching a request against available service representations - implying linear query performance.

Most of the current works concentrating on the automation of the service discovery task, focus on the improvement of the effectiveness of the service matchmaking, i.e. on engineering the service description. Central to the majority of the current SWS matchmaking approaches is that the formal semantics of service descriptions is explicitly defined in an ontology language such as OWL-DL [12]. In this way, service matchmaking can be performed by exploiting standard DL inferences [16, 22, 7, 13] sometimes jointly with the use of syntactic similarity measures [14].

Less attention has been dedicated to the improvement of the efficiency of the service discovery task, that, on the contrary, is the focus of our work. In [21], a way for turning efficient resource retrieval is proposed, by abstracting from a more expressive to a less expressive language, e.g. from OWL to DL-lite. Even if this approach is semantically sound, differently from our method, it looses completeness.

Most of the efforts have been employed for the optimization of reasoning and query answering. In [11], a set of optimization techniques for improving tableaux decision procedures for DLs are presented. They can be effectively used for performing service matchmaking. In [20], an algorithm for optimizing query answering of $\mathcal{SHIQ}$ knowledge bases extended with DL-safe rules is proposed, by exploiting the reduction to disjunctive programs. A combination of DL-tree retrieval with the optimizations defined in [20] could be more helpful than either on its own. Möller et al. [19] propose optimization techniques for improving the scalability of the instance retrieval task. This is orthogonal to our work as our method could be used also for performing instance retrieval by firstly clustering the MSCs of the considered knowledge base and then querying for the concept of interest, by checking for nodes of the DL-Tree that are subsumed by the query concept until leave nodes are found.

Another recent approach aiming at a scalable discovery process is *Semantic Discovery Caching* (SDC) [25]. It is based on an index structure, the SDC graph, that is a subsumption hierarchy made up of goal templates and usable Web services. Templates are organized w.r.t. their semantic similarity. The lower layer is the cache that captures knowledge on the usability of the available Web services. Based on this structure, the discovery process uses inference rules between the similarity degree of goal templates and the usability degree of Web services.

8 Conclusion and Future Work

We have presented a sound and complete method for improving the efficiency of the resource retrieval task and its validity has been experimentally shown. It is based on the exploitation of a tree-index (DL-Tree) that is built by applying a new conceptual clustering algorithm to available resource descriptions. For clustering resources, a new semantic similarity measure has been presented, while intentional cluster descriptions are built exploiting the Good Common Subsumer for $\mathcal{ALE}(\mathcal{T})$ concept descriptions.

Further work for improving the effectiveness of the similarity measure has been planned. Moreover, the validity of the method applied using different matching procedures will be verified. Furthermore, an incremental clustering algorithm will be developed, in order to cope with new available services avoiding the recomputation of a new clustering.

Our approach has been restricted to $\mathcal{ALE}(\mathcal{T})$ concept expressions and instances. However, the DL-Tree indexing procedure actually works with approximations: non-standard inference procedure like the Good Common Subsumer and the Most-Specific Concept. By approximating more expressive DLs expressions (i.e. OWL-DL expressions) to weaker languages, such as $\mathcal{ALE}(\mathcal{T})$ and $\mathcal{ALC}$, it is still possible to use the DL-Tree indexing procedure. Actual empirical evaluations of whether typical OWL-DL ontologies benefit from DL-tree indexing will however still have to be performed.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
2. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In T. Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 96–101. Morgan Kaufmann, 1999.
3. F. Baader, R. Sertkaya, and Y. Turhan. Computing least common subsumers w.r.t. a background terminology. In V. Haarslev and R. Möller, editors, *Proceedings of Proceedings of the 2004 International Workshop on Description Logics (DL2004)*. CEUR-WS.org, 2004.
4. H.H. Bock. *Analysis of Symbolic Data: Exploratory Methods for Extracting Statistical Information from Complex Data*. Springer-Verlag, 1999.
5. A. Borgida, T. Walsh, and H. Hirsh. Towards measuring similarity in description logics. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL2005)*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.
6. M. W. Bright, A. R. Hurson, and Simin H. Pakzad. Automated resolution of semantic heterogeneity in multidatabases. *ACM Transaction on Database Systems*, 19(2):212–253, 1994.
7. S. Colucci, T. D. Noia, E. Di Sciascio, F. Donini, and M. Mongiello. Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. In *In Proc. 6th Int. Conference on Electronic Commerce (ICEC 2004)*. ACM Press, 2004.
8. C. d'Amato, N. Fanizzi, and F. Esposito. A semantic similarity measure for expressive description logics. In A. Pettorossi, editor, *Proceedings of Convegno Italiano di Logica Computazionale, CILC05*, Rome, Italy, 2005. `http://www.disp.uniroma2.it/CILC2005/downloads/papers/15.dAmato_CILC05.pdf`.
9. C. d'Amato, N. Fanizzi, and F. Esposito. A dissimilarity measure for $\mathcal{ALC}$ concept descriptions. In *Proc. of the 21st Annual ACM Symposium of Applied Computing, SAC2006*, 2006.
10. S. Grimm, B. Motik, and C. Preist. Variance in e-business service discovery. In *Proceedings of the ISWC Workshop on Semantic Web Services*, 2004.
11. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.
12. I. Horrocks, P. Patel-Schneider, and F. van Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Web Semantics*, 1(1), 2004.
13. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *In Proceedings of the 2nd European Semantic Web Conference (ESWC 2005)*, volume 3532 of *LNCS*, 2005.
14. M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with owls-mx. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA, 2006. ACM Press.
15. J. Lee, M. Kim, and Y. Lee. Information retrieval based on conceptual distance in is-a hierarchies. *Journal of Documentation*, 2(49):188–207, 1993.
16. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 331–339, New York, NY, USA, 2003. ACM Press.
17. T. Mantay. Commonality-based ABox retrieval. Technical Report FBI-HH-M-291/2000, Department of Computer Science, University of Hamburg, Germany, 2000.
18. D. Maynard, W. Peters, and Y. Li. Metrics for evaluation of ontology-based information extraction. In *Proceeding of the EON 2006 Workshop*, 2006.

19. R. Möller, V. Haarslev, and M. Wessel. On the scalability of description logic instance retrieval. In In B. Parsia, U. Sattler, and D. Toman (Eds.), editors, *Proceedings of the International Workshop on Description Logics (DL2006)*, volume 189. CEUR, 2006.
20. B. Motik, U. Sattler, and R. Studer. Query answering for owl-dl with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.
21. J. Z. Pan, E. Thomas, and D. Sleeman. Ontosearch2: Searching and querying web ontologies. In *Proc. of the IADIS International Conference*, 2006.
22. M. Paolucci, T. Kawamura, T.R. Payne, and K.P. Sycara. Semantic matching of web services capabilities. In *International Semantic Web Conference*, pages 333–347, 2002.
23. R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on System, Man, and Cybernetics*, 19(1):17–30, 1989.
24. P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999.
25. M. Stollberg and M. Hepp. Semantic discovery caching (sdc): Prototype and use case evaluation. Technical report, DERI, 07 April 2007.
26. E. M. Voorhees. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. *Information Processing and Management*, 22(6):465–476, 1986.

Towards Fine-grained Service Matchmaking by Using Concept Similarity

Alberto Fernandez[1], Axel Polleres[2], and Sascha Ossowski[1]

[1] Universidad Rey Juan Carlos, Móstoles, Spain
{alberto.fernandez,sascha.ossowski}@urjc.es
[2] Digital Enterprise Research Insitute (DERI), National University of Ireland, Galway
{axel.polleres}@deri.org

Abstract. Several description frameworks to semantically describe and match services on the one hand and service requests on the other have been presented in the literature. Many of the current proposals for defining notions of match between service advertisements and requests are based on subsumption checking in more or less expressive Description Logics, thus providing boolean match functions, rather than a fine-grained, numerical degree of match. By contrast, concept similarity measures investigated in the DL literature explicitely include such a quantitative notion. In this paper we try to take a step forward in this area by means of an analysis of existing approaches from both semantic web service matching and concept similarity, and provide preliminary ideas on how to combine these two building blocks in a unified service selection framework.

1 Introduction

In the quest to provide the underpinnings for Service-Oriented Architectures, proper methods to enable the automatic location and selection of suitable services in order to solve a given task or user request are an essential ingredient. To this end, several description frameworks to semantically annotate provided services on the one hand and express service requests on the other, both based on the same shared, formal ontologies, have been presented in the literature. Complementarily, numerous proposals for defining notions of match between such semantic descriptions of service advertisements and requests have been developed over the last few years.

Many of these are based on subsumption checking in more or less expressive Description Logics, thus providing boolean match functions, but not a fine-grained, numerical degree of match. On the contrary, concept similarity measures investigated in the DL literature provide precisely this missing piece but their application to the concrete domain of service matching is very limited. This is not as surprising as it may seem because, as pointed out in this work, the combination of service matching notions and concept similarity in a unified framework is not as straightforward as could be expected.

The objective of this paper is to take a step forward in this area by a systematical analysis of existing approaches from both semantic web service matching and concept similarity in order to combine these two building blocks into a unified service selection framework.

The rest of the paper is organized as follows. In section 2 we provide a review of notions of concept similarity in ontologies. Then, we survey current approaches found in literature to semantic service descriptions by means of ontologies and notions of match between formally defined requests and service advertisements, and discuss how concept similarity can be used to refine them. We provide also preliminary ideas where concept similarity could be beneficial to refine the notions of service matchmaking, aiming at a framework for a numerical notion of service match aimed at refineing the notions defined in literature. We conclude with an outlook and future work.

2 Preliminaries

In order to enable semantic matchmaking, it is necessary that possible communication partners, say service providers and requesters, agree on a certain *specification of a conceptualization*[13] of the domain, i.e. a shared, formal *ontology*. In the context of the Semantic Web and Semantic Web Services, this term which originally sets from Philosophy, is usually conceived by Computer Scientists as a logical theory defining and axiomatizing the concepts and properties used to describe the domain. Common to almost all ontology languages (like DAML+OIL[3] ,OWL [6], KIF [12], WSML [5], Common Logic [7]) is that in principle they are based on first-order languages, usually representing concepts as unary predicates and properties (i.e., relations between concepts) as binary predicates. In such a language a subclass-hierarchy (or taxonomy) of concepts can be expressed simply by a set of implications, where e.g.

$$\forall x OnlineBankingService(x) \rightarrow FinancialService(x) \tag{1}$$

expresses that concept $OnlineBankingService$ is a subclass of *FinancialService* and simple facts like $FinancialService(myDepotService)$ denote membership of certain instances in classes. With these basic ingredients, it is already possible to describe simple taxonomies of concepts.

2.1 Description Logics

In the context of conceptual and ontological reasoning especially the Description Logics (DL) fragments [1] of first order logics have gained momentum, due to their desirable features such as decidability of core reasoning tasks such as concept subsumption and concept membership. Among these, especially SHIQ, SHIF, and SHOIN deserve attention, being the logical foundations of DAML+OIL, OWL Light, and OWL DL, respectively. As opposed to simple subclass hierarchies expressible with formula like (1), DLs allow more sophisticated definitions of concept hierarchies by relating concepts by roles (binary relations) and defining subclass relations via these roles. Roles may also be viewed as object attributes or predicate-value pairs assigned to objects, respectively, and are usually modelled via binary predicates, where e.g.

$$\forall x.CreditCardAccountService(x) \rightarrow (\exists y.input(x,y) \rightarrow CredidCardNumber(y)) \tag{2}$$

expresses that the class of $CreditCardAccountService$ is a subclass of the "services which have a CreditCardNumber as input", or, in other words that all $CreditCard$-$AccountService$s have a $CreditCardNumber$ as $input$. In order to express such complex subclass relations, DLs provide an easier to read syntax to define complex class descriptions as follows (we take here the syntactic constracts of SHOIN, the base language of OWL DL, as a basic example), where C,D are class descriptions and R is a role name:

DL Syntax	First-order Syntax
C	$C(x)$
$\exists R.C$	$\exists y.R(x,y) \wedge C(y)$
$\forall R.C$	$\forall y.R(x,y) \rightarrow C(y)$
$C \sqcap D$	$C(x) \wedge D(x)$
$C \sqcup D$	$C(x) \vee D(x)$
$\geq nR$	$\exists y_1, y_2, \ldots y_n. \bigwedge_{1 \leq i \leq n} R(x, y_n) \wedge \bigwedge_{i \neq \mathrm{J}} y_i \neq y_j$
$\leq nR$	$\neg\exists y_1, y_2, \ldots y_{n+1}. \bigwedge_{1 \leq i \leq n} R(x, y_{n+1}) \wedge \bigwedge_{i \neq \mathrm{J}} y_i \neq y_j$

The above subclass statment (2) would then be written *CreditCardAccountService* $\sqsubseteq \exists input.CreditCardNumber$ in DL notation. Now, if you had for instance additional information that $CreditCardNumber \sqsubseteq PaymentCredential$, and that everything which has a payment credential as input, is in the class $CommercialService$, i.e. $\exists input.PaymentCredential \sqsubseteq CommercialService$ we could additionally infer that $CreditCardAccountService$ is a subclass of $CommercialService$, i.e. $Credit$-$CardAccountService \sqsubseteq CommercialService$ Commercial Services are not necessarily only ones dealing with credit card account management, another subclass of $CommercialService$ is for instance *CarRentalService*. Figure 1 shows a simple concept hierarchy for the concepts mentioned here showing explicit (arrows) and some inferred (dashed arrows) subclass relationships.

By complex concept definitions and inferred concepts, terms like *least common subsumer (lcs)*[3], depth or *distance* in the concept lattice, which are quite intuitive for simple taxonomies, become a bit blurry. In fact, we can observe that literature which talks about measures of distance in description logics such as [2] usually only consider very simple description logics. We will get back to this point later on and leave the reader at the moment with the question to intuitively try to assess whether $CarRentalService$ is more "similar" to $CommercialService$ than $CreditCardAccountService$?

2.2 Concept Similarity

Getting back to our goal to find measures for "matches" we find several similarity measures having been proposed in the Literature. Some authors define the *similarity* while others use *distance*. Both measures are inversely proportional and usually are taken as the inverse of each other.

Simple (atomic) concepts In this section we describe some of the approaches proposed in the literature to measure similarity between two simple concepts.

[3] also known as the *most specific ancestor (msa)*

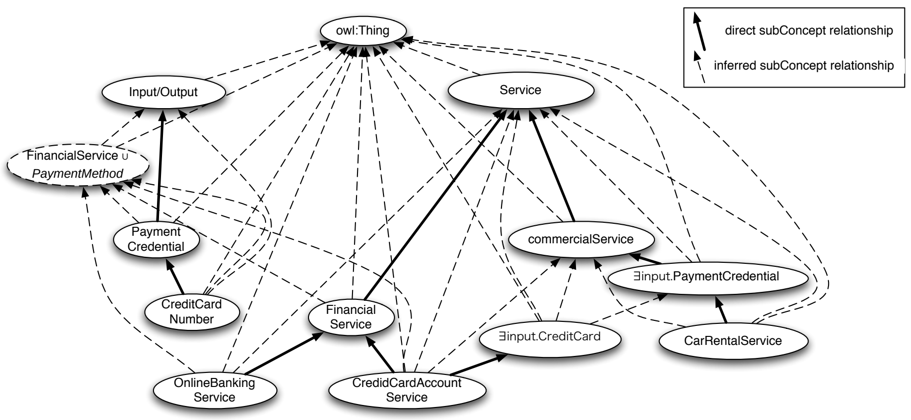


Fig. 1. A simple concept hierarchy for services

One of the most well known *distance* measures between concepts is the **length of the shortest path** between them in the taxonomy, proposed by Rada et al. [24]. As both concepts might not be along the same branch of the taxonomy tree, it can be calculated as the sum of the path length from each concept to their *lcs*.

$$dist(c_1, c_2) = depth(c_1) + depth(c_2) - 2 \times depth(lcs(c_1, c_2)) \tag{3}$$

where $depth(c)$ is the number of edges from c to the root concept.

Leacock & Chodorow [17] define the similarity between two terms as the *relatedness*, which they define as the inverse of the semantic distance.

$$relatedness(t_1, t_2) = -\log \frac{dist(t_1, t_2)}{2D} \tag{4}$$

where $dist(t_1, t_2)$ is the same as (3), and D is the maximum depth of the structure.

In their role-based[4] service matchmaking approach Fernández et al. [11] consider similarity (*degree of match)* as asymmetric. They consider some degree of similarity between concepts if there is a subsumption relation between them in the taxonomy. They define the following function, which is also based on the path length between them.

$$sim(c_1, c_2) = \begin{cases} 1 & if\ c_1 = c_2 \\ \frac{1}{2} + \frac{1}{2 \cdot e^{\|c_1, c_2\|}} & \text{if } c_2 \text{ subsumes } c_1 \\ \frac{1}{2} \cdot e^{\|c_1, c_2\|} & \text{if } c_1 \text{ subsumes } c_2 \\ 0 & otherwise \end{cases} \tag{5}$$

[4] Here we refer to roles of an actor or agent, as opposed to the roles related to ontologies that we mentioned above.

Here, $\| \cdot, \cdot \|$ is the distance $(depth(c_2) - depth(c_1))$ in the taxonomy tree.

The aforementioned measures (3) and (5) are independent of the absolute location of concepts in the taxonomy tree. Other proposals further refine these approaches by taking into account the **depth of the concepts in the taxonomy**. This makes sense under the assumption that concepts at upper layers have more general semantics and less similarity between them, while concepts at lower layers have more concrete semantics and thus stronger similarity.

In this line, Wu & Palmer [28] use the terminology *score* to define the similarity between two terms:

$$score(t_1, t_2) = \frac{2N_3}{N_1 + N_2 + 2N_3} \tag{6}$$

Where N_1 and N_2 are the length sof the shortest path from t_1 and t_2 (respectively) to the *lcs*, and N_3 is the length of the shortest path from the *lcs* to the root.

Li et al. [19] define the similarity between two concepts as:

$$sim(c_1, c_2) = \begin{cases} e^{-\alpha l} \cdot \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} & si\ c_1 \neq c_2 \\ 1 & otherwise \end{cases} \tag{7}$$

where $\alpha \geq 0$ and $\beta \geq 0$ are parameters scaling the contribution of the shortest path length (l) between the two concepts and the depth of the *lcs* (h) in the concept hierarchy, respectively.

Other authors do not base concept similarity on the distance between the concepts in the taxonomy.

Tversky [27] proposes an approach in which a concept C is characterized by a set of features, $ftrs(C)$. He introduces two kind of measures:

1. *contrast model*

   $$contrast(C, D) = \theta f(ftrs(C) \cap ftrs(D)) - \alpha f(ftrs(C) \backslash ftrs(D)) - \beta f(ftrs(D) \backslash ftrs(C)) \tag{8}$$

   where $\backslash$ is set difference, θ, α and β are non-negative constants, and $f(\cdot)$ is usually the count of features in the set. That is, the number of common minus the number of non-common features.
2. *ratio model*

   $$sim(C, D) = \frac{f(ftrs(C) \cap ftrs(D))}{f(ftrs(C) \cap ftrs(D)) + \alpha f(ftrs(C) \backslash ftrs(D)) + \beta f(ftrs(D) \backslash ftrs(C))} \tag{9}$$

   When asymmetry of similarity is not desired, $\alpha = \beta = 0.5$ can be chosen, and under the assumption that f is distributive over disjoint sets $(f(V \cup W) = f(V) + f(W))$, similarity is commonly taken as:

   $$dist(C, D) = \frac{2 \times f(ftrs(C) \cap ftrs(D))}{f(ftrs(C)) + f(ftrs(D))} \tag{10}$$

Resnik [25] proposes an information-content based model in which there are information about the probability of an individual being described by a specific concept C ($pr(C)$). He uses the $lcs(c_1, c_2)$ as the representative of the similarity of c_1 and c_2, and proposes the *information content* as similarity measure:

$$sim(c_1, c_2) = IC(lcs(c_1, c_2)) = -\log pr(lcs(c_1, c_2)) \tag{11}$$

This approach has the advantage of not being transparent to changes in the hierarchy.

Jiang & Conrath [14] refine Resnick's measure:

$$sim(c_1, c_2) = IC(c_1) + IC(c_2) - 2 \times IC(lcs(c_1, c_2)) \tag{12}$$

Lin [20] proposes:

$$sim(c_1, c_2) = \frac{2 \times IC(lcs(c_1, c_2))}{IC(c_1) + IC(c_2)} \tag{13}$$

Borgida et al. [2] apply some of the previous approaches to a very simple DL (A), involving only conjunctions. Di Noia and colleagues [22] focus on DL and propose a ranking function for what they call *potential match* (some requests in demand D are not specified in supply S). The ranking function *rankPotential(S,D)* counts:

– the number of concepts names in D not in S,
– the number of number restrictions of D not implied by those of S,
– add recursively *rankPotential* for each universal role quantification in D,

assuming 0 to be the best ranking.

Fanizzi & d'Amato [9] define a similarity measure between concepts in ALN DL. They decompose the normal form of the concept descriptions and measure the similarity of the subconcepts:

– Primitive concepts: ratio of the number of common individuals with respect to the number of individuals belonging to either conjunct.
– Value restrictions: computed recursively, the average value is taken.
– Numeric restrictions: ratio of overlap between the two intervals and the larger interval (whose extremes are minimum and maximum), the average value is taken.

In the OWLS-MX [15] semantic Web service matching approach, logic-based reasoning is complemented by IR (Information Retrieval) based similarity computation. In particular, they allow four different token-based string metrics: the *cosine*, the *loss of information*, the *extended Jacquard* and the *Jensen-Shannon information divergence* similarity metrics. This metrics are applied to unfolded concepts, e.g. the unfolded expression *(and C (and B (and A)))* corresponds to the concept C ($C \sqsubseteq B \sqsubseteq A$).

Table 1 summarizes the different approaches to concept similarity described in this section.

The first characteristic determines whether a taxonomy tree based (*structural*) model, a *feature* based model, or a *DL* based model is applied (including the DL language used).

	Approach	focus	symmetry	distance	depth	measure	range
Rada et al.	structural	concept	yes	yes	no	distance	0..2H
Leacock-Chodorow	structural	concept	yes	yes	no	similarity	$0..\infty$
Fernndez et al.	structural	concept	no	yes	no	similarity	0..1
Wu-Palmer	structural	concept	yes	yes	yes	similarity	0..1
Li et al.	structural	concept	yes	yes	yes	similarity	0..1
OWLS-MX	structural	concept	yes	yes*	yes*	similarity	0..1
Tversky-contrast	feature	concept	yes	no	no	similarity	0..N
Tversky-ratio	feature	concept	yes/no	no	no	similarity	0..1
Resnick	structural	instance	yes	no	no	similarity	$0..\infty$
Jiang-Conrad	structural	instance	yes	no	no	distance	$0..\infty$
Lin	structural	instance	yes	no	no	similarity	0..1
Borgida et al.-feature	DL A	concept	yes	no	no	similarity	0..1(N)
Borgida at al.-struct.	DL A	concept	yes	yes	no	distance	0..2H
Borgida et al.-IC	DL A	instance	yes	no	no	distance/sim	$0..\infty(1)$
Di Noia et al.	DL	concept	no	no	no	distance	≥ 0
Fanizzi-d'Amato	DL ALN	concept	yes	no	no	similarity	0..1

Table 1. Summary of concept similarity approaches

Most approaches described here make use of *concept* definitions but others base their similarity measures on the number of instances of concepts, which are less affected by changes in the taxonomy.

Although *symmetry* has been defined by several authors as a desirable property of similarity functions, not all approaches readily comply with it. Consider, for instance, a semantic service matching scenario where it is important whether an input concept in the query subsumes or is subsumed by an input concept of an advertised service (this determines if the service can, at least, be invoked). Note that the Tversky-ratio approach allows both, symmetric and asymmetric options, depending on the values of some parameters in its similarity function.

Distance between concepts in the taxonomy is the main parameter used by structural approaches (including the Borgida et al. DL based on Rada's function). This measure makes sense under the assumption of equally distributed instances over concepts; otherwise pairs of concepts in a fine grained part of the taxonomy would be ranked with lower similarity than concepts at the same distance in another part. In the case of DL, distance is difficult to be used unless some kind of "canonical representation" is adopted. However, as we illustrated by the example of Figure 1, such a canonical representation is hard to find for expressive DLs. Common DL reasoners allow to "pre-classify" the TBox of an ontology, thus computing all subclass relations of named concepts. One could take the spanning tree of such a pre-classification, removing all transitive edges as a starting point for distance measures, but would miss the difference then for concepts defined by restrictions. Another possible way to circumvent this and maybe arriving at a more precise canonical representation would be to recursively introduce new atomic concept names for all atomic restrictions $\exists R.C$, $\forall R.C$, such that R is a role and C is an atomic concept occurring in the TBox and use these as well in the pre-classification.

DLs allowing disjunction ($\sqcup$) are hard to grasp by such approaches, we will mention more on complex compound concepts below.

As for assumptions, we mentioned above that often one assumes that in a taxonomy tree, higher nodes represent more general concepts (less similar semantically), while lower levels contain more specific concepts. For this reason, some approaches also take into account the depth of the concepts in the taxonomy. However, we note that this assumption only makes sense if we additionally assume equal distribution among instances among subclasses in the ontology in the general case.

Note that, the way OWLS-MX applies IR techniques by unfolding concept names, indirectly uses the distance and depth of concepts, but could also be viewed as kind of feature-based similarity measure mentioned above.

We also detail whether they define a *similarity* function or a *distance* function. Although both measures can be easily obtained from each other (e.g. $sim = \frac{1}{dist}$) we prefer keeping their original definition, as they vary on the *range* of the returned value (last column). A unified range of values is convenient in order to make it easier to combine/aggregate similarity values in case of complex expressions involving several concepts.

Complex (compound) concepts Rada et al. also extend the definition of *distance* to handle compound concepts represented by a set of concepts. Concepts in those sets can be interpreted as conjunctions or disjunctions. In the case of a disjunction of concepts the distance is defined as:

$$dist(C_1 \sqcup \ldots \sqcup C_k, C) = min_i dist(C_i, C) \tag{14}$$

where C_i and C represent concepts (elementary or compound). When C itself is a disjunctive concept, the same function ($dist(C_i, C)$) is in turn applied.

The distance between conjunctive concepts is defined as:

$$dist(V_1, V_2) = \begin{cases} 0 & if\ V_1 = V_2 \\ \frac{1}{|V_1||V_2|} \sum\limits_{u \in V_1} \sum\limits_{v \in V_2} dist(u, v) & otherwise \end{cases} \tag{15}$$

where V_1 and V_2 are sets representing compound concepts consisting of a conjunction ($\sqcap$) of its elements, $|\cdot|$ is set cardinality, and $dist(u, v)$ is the shortest path length between nodes u and v.

Ehrig et al. [8] analyze three layers on which similarity between concepts can be measured: data, ontology and context layer. We are interested on the ontology level. They use the function proposed by Li et al. in case of similarity between concepts. They also propose the following formula (cosine) to calculate the similarity between two sets of concepts:

$$sim(E, F) = \frac{\sum\limits_{e \in E} \boldsymbol{e} \cdot \sum\limits_{f \in F} \boldsymbol{f}}{\left|\sum\limits_{e \in E} \boldsymbol{e}\right| \cdot \left|\sum\limits_{f \in F} \boldsymbol{f}\right|} \tag{16}$$

with $E = \{e_1, e_2, \ldots\}$, $\boldsymbol{e} = (sim(e, e_1), sim(e, e_2), \ldots, sim(e, f_1), sim(e, f_2), \ldots)$, and the analogously for F and $\boldsymbol{f}$, respectively.

Sierra & Debenham [26] define the semantic similarity between two logical formulas as the maxmin similarity between the sets of concepts ($O(\cdot)$) that appear in the formulas:

$$sim(\varphi, \psi) = \max_{c_i \in O(\varphi)} \min_{c_j \in O(\psi)} \{sim(c_i, c_j)\} \tag{17}$$

In total, it seems that combinations of these approaches could be beneficial. Especially feature-set approaches seem to be worthwhile to be combined with approaches handling compound, complex DL expressions in order to get to more precise overall measures.

3 Matching Semantic Web Services

When having a closer look at current proposals to effectively annotating Web Services with Semantic descriptions, we can identify the following "hooks" for adding such annotations referring to ontologies as discussed so far. We focus here on components of semantic Web Service descriptions for which concepts in a taxonomy or complex ontology can be used for annotating them.

Service Taxonomies The entirety of the functionality offered by a Web Service can be described by a taxonomy, grouping service instances hierarchically, such as for instance the $CarRentalService$ or $CreditCardAccountService$ mentioned in Figure 1.

Operations When searching for a certain functionality, one often searches for a particular operation to execute, rather than the entirety of service functionality. Thus, most description frameworks support assigning offered operations to a taxonomy of operations in a service. Such an Operation could for instance be $RequestCreditCardBalance$, $BookRentalCar$, or all operations having a payment credential as input – which could be modeled by something like $WSDLOperation \sqcap \exists input.PaymentCredential$ – etc., all of which again may be grouped in a taxonomy/ontological hierarchy.

Inputs/Outputs Input values or output values of web services or certain service operations might be bound to a certain concept in an ontology. The problem of relating a concrete input or output message format (as for instance described in a WSDL file) is often referred to as lifting/lowering [16] problem and solved slightly different in the various Semantic Web service description approaches.

Preconditions/Postconditions Frameworks like OWL-S and WSMO offer functionality to annotate services and/or operations with pre- and postconditions, i.e. logical formulae expressing conditions over the state of the world. Since these conditions can usually not be expressed in a taxonomy or ontological hierarchy, rather more complex formalisms than Description Logics or OWL are proposed to describe these, like WSML logical expression in WSMO or SWRL rule bodies, expressing conjunctive queries on the "state space" in OWL-S. Our focus in the current paper is on applying concept similarity to service matching and we are not aware of service matching approaches which practically exploit pre-/postcondition matching at the moment. In summary, it seems to be not entirely clear, how pre-/postcondition matching can be done at all in open service environments, which might also be a reason why they have not been considered e.g. in SAWSDL.

Summarizing, we will try to take into account those parts of the service description which allow for a conceptualication in a formal ontology, namely inputs/outputs, overall service functionality, and operations. Moreover, we deem useful to assume the following attributes/roles:

– **hasInput:** domain: $Service \sqcup Operation$, range: $Input$
– **hasOutput:** domain: $Service \sqcup Operation$, range: $Output$
– **hasOperation:** domain: $Service$, range: $Operation$

Services or Operations might have additional attributes assigned, e.g. describing non-functional properties which likewise might be useful for precise matchmaking, but which we consider out of scope for the current paper being focused on matchmaking by concept similarity.

SWS frameworks In the following table, let us briefly analyse if and how the aforementioned components are supported by three of the most common Semantic Web Service Description Frameworks[5], namely, OWL-S [21], WSMO [4] and SAWSDL [10], which has just reached the status of a proposed recommendation within W3C. We note

	Service	**Operation**	**Input/Output**	**Pre/Postcondition**
OWL-S	yes	OWL-S service models	OWL-S service models	OWL-S service models
WSMO	yes	WSMO capabilities	WSMO choreography model	WSMO capabilities
SAWSDL	modelReference in wsdl:interface	modelReference in wsdl:operation	modelReference in xsd:element	no

Table 2. Where SWS approaches allow annotations by concepts from given ontologies

that while SAWSDL is in general to be viewed a simpler framework than the other two, it offers useful features in comparison to its predecessors, e.g. having sophisticated support to annotate inputs and output messages; SAWSDL allows to add annotations on the level of single XML Schema elements directly within XML Schema, describing parts of the allowed input/output messages, whereas OWL-S and WSDL concepts can be assigned only per input and tying to a particular XML Schema describing the concrete message format has to be defined in the so-called grounding, typically via an XSL transformation. We mention this, because at the time being, the main development effort and activity, as well as chance of making it through to becoming a standard is on SAWSDL.

[5] (in order of "appearance")

3.1 Notions of match

In the following, we will try to analyze how existing approaches for service matching cater for ranking and make suggestions where concept similarity measured could be plugged to refine the proposed notions of match.

Paolucci [23] Many of the current approaches to semantic web services matching, particularly those based on OWL-S, started from the work of Paolucci et al. [23]. This approach proposes a matching algorithm that takes into account inputs and outputs of advertised and requested services. An output matches iff for each output of the request there is a matching output in the service description. The authors differentiate four (ranked) degrees of match (OUT_S and OUT_R correspond to outputs of the advertised and requested services, respectively)[6]:

- *exact*: if $\text{OUT}_R \doteq \text{OUT}_S$; or OUT_R is a direct subclass of OUT_S under the assumption that by advertising OUT_S the provider commits to provide outputs consistent with every immediate subtype of OUT_S.
- *plug-in*: if $\text{OUT}_S \succeq \text{OUT}_R$, that is, OUT_S could be plugged in place of OUT_R.
- *subsumes*: if $\text{OUT}_R \succeq \text{OUT}_S$.
- *fail*: no subsumption relation between OUT_S and OUT_R exists.

If there are several outputs with different degree of match, the minimum degree is used. The same algorithm is used to compute the matching between inputs, but with the order of request and advertisement reversed. Finally, the set of service advertisements is sorted by comparing output matches first, if equally scored, considering the input matches.

Applying concept similarity In Paolucci's approach services are sorted according to their degree of match, being $exact > plug - in > subsumes > fail$. However, services falling into the same category (e.g. plugin) have the same priority. A concept similarity approach can be used to refine the ranking of services inside each degree of match category. In particular, only *plug-in* and *subsumes* should be refined. As they base their classification on the subsumption relation between concepts in a taxonomy tree, one of the structural (path length based) similarity approaches might be adequate. In case of several inputs (or outputs), they consider the minimum among their degrees of match. In the same line, the minimum value can also be used to compare their similarity measures.

OWLS-MX The OWLS-MX matchmaker [15] performs hybrid semantic matching that complements logic based reasoning with syntactic IR based similarity metrics. The first three degrees of match are logic based only and, although using the same naming as Paolucci, they are defined differently (e.g. in OWLS-MX inputs of the advertisement always must at least subsume the ones in the request, so the service can be invoked).[7]

[6] $\doteq$ and $\succeq$ terminological concept equivalence and subsumption, respectively.

[7] $LSC(C)$ (set of least specific concepts (direct children) of C), $LGC(C)$ (set of least generic concepts (direct parents) of C), $Sim_{IR}(A, B) \in [0, 1]$ the numeric degree of syntactic similarity between strings A and B according to chosen IR metric IR

- *exact*: iff $\forall$ IN_S $\exists$ IN_R: $\text{IN}_S \doteq \text{IN}_R \wedge \forall$ OUT_R $\exists$ OUT_S: $\text{OUT}_R \doteq \text{OUT}_S$.
- *plug-in*: $\forall$ IN_S $\exists$ IN_R: $\text{IN}_S \succeq \text{IN}_R \wedge \forall$ OUT_R $\exists$ OUT_S: $\text{OUT}_S \in$ LSC(OUT_R). S is expected to return more specific output data whose logically defined semantics is exactly the same or very close to the requested by the user.
- *subsumes* $\forall$ IN_S $\exists$ IN_R: $\text{IN}_S \succeq \text{IN}_R \wedge \forall$ OUT_R $\exists$ OUT_S: $\text{OUT}_R \succeq \text{OUT}_S$. This relaxes the constraint of immediate output concept subsumption.
- *subsumed-by* $\forall$ IN_S $\exists$ IN_R: $\text{IN}_S \succeq \text{IN}_R \wedge \forall$ OUT_R $\exists$ OUT_S: ($\text{OUT}_S \doteq \text{OUT}_R \vee \text{OUT}_S$ $\in$ LGC(OUT_R)) $\wedge$ SIM_{IR}(S, R) $\geq \alpha$. Output data is more general than requested. It is focused on direct parent output concepts to avoid selecting services returning data too general. It is combined with the syntactic similarity.
- *logic-based fail*: matching fails according to the above logic-based semantic criteria.
- *nearest-neighbor* $\forall$ IN_S $\exists$ IN_R: $\text{IN}_S \succeq \text{IN}_R \wedge \forall$ OUT_R $\exists$ OUT_S: $\text{OUT}_R \succeq \text{OUT}_S \vee$ SIM_{IR}(S, R) $\geq \alpha$.
- *fail*: service advertisement and request do not match according to the above criteria.

Applying concept similarity As occurred in the previous approach, concept similarity could be applied when the subsumption relation is checked ($\succeq$). Now the aggregation of similarity values is a little more complicated since, besides the set of inputs and outputs, it has also to be combined with the IR similarity value (in the case of *subsumed-by* and *nearest-neighbor*). This combination is not straightforward, maybe a parametrized function which allows scaling the contribution of each measure might be appropriate.

Li Horrocks[18] A DL concept is used to describe the inputs and another for the outputs of a service advertisement or request. They extend the degrees of match proposed by Paolucci et al. by adding an *intersection* match. Formally,

- *exact*: if $A \equiv R$.
- *plug-in*: if $R \sqsubseteq A$.
- *subsume*: if $A \sqsubseteq R$.
- *intersection*: $\neg(A \sqcap R \sqsubseteq \bot)$
- *disjoint*: $A \sqcap R \sqsubseteq \bot$.

Applying concept similarity Since this approach is focused on DL, in this case distance is difficult to be used, unless some canonical representation is found. Although some approaches to concept similarity for DL were reviewed in section 2.2, they resulted to be applied on very simple DL, thus more investigation in this line is needed.

3.2 Towards a combined notion of similarity-based Service matchmaking

In this section we provide preliminary ideas on how concept similarity might be combined with notions of match. Our aim is to provide a unified matching function which returns a numeric value that can be used for ranking services. We consider such a function with range [0..1] although, of course, any other range would be acceptable as well.

In section 3 we identified three components of semantic service descriptions: *inputs*, *outputs* and *operations*. For each component, a function should return its degree of match, I_M, O_M, and Op_M, respectively. Following current approaches, relations between components are usually classified according to several categories or *notions of match*(e.g. *exact*, *plug-in*, *intersection*,...). As analyzed in section 3.1 this classification is coarse grained and might be refined with concept similarity approaches. The ranking function must compare the *notion of match* first, and then the (numerical) similarity value. A way to facilitate the use of such a function is by dividing the range [0..1] in non overlapping intervals and scaling the similarity value to the interval corresponding to its category. Any division is acceptable under the condition that it keeps the order as is done for their categories. We define the following functions (being c_1 and c_2 concepts):

– $nom(c_1, c_2)$: returns the *notion of match* category $\in \{cat_1, cat_2, ..., cat_N\}$, where $cat_1 > cat_2 > cat_N$. Note that usually $cat_1 = exact$ and $cat_N = fail$.
– $inf(cat)$: returns the lower limit of the interval corresponding to category cat.
– $sup(cat)$: returns the upper limit of the interval corresponding to category cat.
– $sim(c_1, c_2)$: returns the concept similarity, which is a value $\in [0..1]$.
– $nosm(c_1, c_2)$: returns the *notion of similarity match* between c_1 and c_2, which is a value resulting of scaling the $sim(c_1, c_2)$ into the interval corresponding to the category $nom(c_1, c_2)$. This can be defined as

$$nosm(c_1, c_2) = inf(nom(c_1, c_2)) + sim(c_1, c_2) \cdot (sup(nom(c_1, c_2)) - inf(nom(c_1, c_2)))$$

I_M, O_M and Op_M should be defined based on $nosm$ applied to its individual elements (e.g. each of its inputs for I_M). Such a functions, for instance I_M might use aggregation functions, like the ones described for similarity of compound concepts in section 2.2 or others.

Finally, the three values need to be combined, for instance by taking a weighted sum:

$$match(S, R) = \alpha \cdot I_M + \beta \cdot O_M + \theta \cdot Op_M \tag{18}$$

where α, β and $\theta \in [0..1]$, and $\alpha + \beta + \theta = 1$.

4 Conclusions

In this paper we have provided a survey of current approaches to semantic service descriptions by means of ontologies and notions of matching between requests and service advertisements. These proposals rank service advertisements following a (coarse-grained) notion of match classification. We have also reviewed concept similarity frameworks in ontologies and discussed how these could be incorporated into the existing service description and matchmaking methodologies, so as to provide a fine-grained ranking of services. Finally, we have provided preliminary ideas aiming at a numerical notion of service match which combines notions of match with concept similarity.

This paper has reported on our work in progress. Some of the identified open issues include:

– What service description framework should we focus on? Should we select an existing one such as OWL-S, WSMO, etc, or a new one to which these approaches could be mapped?

– Which concept similarity measure better fits our framework? Is there a single "best" measure? What are the conditions that it must fulfill?
– How should values corresponding to different elements be combined?
– do different applications require the same framework or should it be configured for each of them?

Some of these questions are being tackled presently, while the in-depth coverage of others is subject to our future work.

Acknowledgements

This research has been partially supported by the European Commission under grants FP6-IST-511632 (CASCOM) and FP-6-IST-034718 (inContext), by the Spanish Ministry of Education and Science through projects URJC-CM-2006-CET-0300, TIN2006-14630-C03-02 and "Agreement Technologies" (CONSOLIDER CSD2007-0022, INGENIO 2010), as well as by the Science Foundation Ireland under the Lion project (SFI/02/CE1/I131).

References

1. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge, January 2003.
2. Alexander Borgida, Thomas Walsh, and Haym Hirsh. Towards measuring similarity in description logics. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Description Logics*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.
3. Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. DAML+OIL (march 2001) reference description. W3c note, 2001.
4. J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna, and M. Stollberg. *Web Service Modeling Ontology (WSMO)*. W3C Member Submission, 3 June 2005. Available from `http://www.w3.org/Submission/WSMO/`.
5. J. de Bruijn, D. Fensel, U. Keller, M. Kifer H. Lausen, R. Krummenacher, A. Polleres, and L. Predoiu. *Web Service Modeling Language (WSML)*. W3C Member Submission, 3 June 2005. Available from `http://www.w3.org/Submission/WSML/`.
6. M. Dean and G. Schreiber, editors. *OWL Web Ontology Language Reference*.
7. Harry Delugach, editor. *ISO Common Logic*. 2006. Available at `http://philebus.tamu.edu/cl/`.
8. Marc Ehrig, Peter Haase, Mark Hefke, and Nenad Stojanovic. Similarity for ontologies - a comprehensive framework. In *ECIS*, 2005.
9. N. Fanizzi and C. d'Amato. A similarity measure for the aln description logic. In *Proceedings of CILC 2006 - Italian Conference on Computational Logic, Bari, Italy, June 26-27, 2006*, 2006.
10. Joel Farrell and Holger Lausen, editors. *Semantic Annotations for WSDL and XML Schema*. W3C Proposed Recommendation, 05 July 2007. Available from `http://www.w3.org/TR/sawsdl/`.

11. Alberto Fernández, Matteo Vasirani, César Cáceres, and Sascha Ossowski. A role-based support mechanism for service description and discovery. In Jingshan Huang, Ryszard Kowalczyk, Zakaria Maamar, David Martin, Ingo Müller, Suzette Stoutenburg, and Katia P. Sycara, editors, *SOCASE*, volume 4504 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2007.
12. Michael R. Genesereth and Richard E. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, June 1992.
13. Tom R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 2(5):199–220, 1993. Available from `http://www-ksl.stanford.edu/kst/what-is-an-ontology.html`.
14. Jay J. Jiang and David W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *CoRR*, cmp-lg/9709008, 1997.
15. Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated semantic web service discovery with owls-mx. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA, 2006. ACM Press.
16. Jacek Kopecky, Dumitru Roman, Matthew Moran, and Dieter Fensel. Semantic web services grounding. In *Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, 2006.
17. Claudia Leacock and Martin Chodorow. Combining local context and WordNet similarity for word sense identification. In Christiane Fellbaum, editor, *WordNet: An Electronic Lexical Database*, pages 265–283. MIT Press, 1998.
18. Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. *Int. J. of Electronic Commerce*, 8(4):39–60, 2004.
19. Yuhua Li, Zuhair Bandar, and David McLean. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Trans. Knowl. Data Eng.*, 15(4):871–882, 2003.
20. Dekang Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA, 1998.
21. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDemott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. *OWL-S: Semantic Markup for Web Services*. W3C Member Submission, 22 November 2004. Available from `http://www.w3.org/Submission/OWL-S/`.
22. Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, and Marina Mongiello. Semantic matchmaking in a p-2-p electronic marketplace. In *SAC*, pages 582–586. ACM, 2003.
23. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In *ISWC*, pages 333–347. Springer Verlag, 2002.
24. R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man and Cybernetics,*, 19(1):17–30, 1989.
25. Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453, 1995.
26. Carles Sierra and John K. Debenham. Trust and honour in information-based agency. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *AAMAS*, pages 1225–1232. ACM, 2006.
27. A. Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.
28. Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138, Morristown, NJ, USA, 1994. Association for Computational Linguistics.

Hybrid OWL-S Service Retrieval with OWLS-MX: Benefits and Pitfalls

Matthias Klusch and Benedikt Fries

German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, 66121 Saarbruecken, Germany
klusch@dfki.de

Abstract. The OWLS-MX matchmaker selects OWL-S 1.1 services that are relevant to a given service request by means of combined logic based and approximative matching filters [5]. In this paper, we build upon this work and analyse its retrieval performance in terms of false positive and false negatives to reveal the benefits and pitfalls of both logic based and hybrid matching with OWLS-MX. The analysis results have been exploited in an improved version OWLS-MX+.

1 Introduction

Agent based service discovery aims at coordinating the ultimate service requester with the ultimate service provider agent. This coordination problem can be solved by means of either assisted mediation through middle agents such as matchmakers, brokers and mediators, or in a decentralized peer to peer fashion [4]. In particular, the majority of semantic Web service matchmakers today exploit semantics that are implicit, for example, in patterns or relative frequencies of terms in service descriptions as computed by techniques from data mining, linguistics, or content-based information retrieval.

In line with the recently started shift in semantic Web research towards scalable approximative rather than strict logic based reasoning [2], we proclaim that building semantic service matchmakers purely on DL inferencing, as realized by the majority of existing approaches, might be insufficient in practice. The quality of semantic service discovery can be significantly improved by appropriate exploitation of both crisp logic based and approximate matching where each of them alone would fail.

One example of such a hybrid semantic service matchmaker for OWL-S is OWLS-MX. It takes any desired OWL-S service as a query, and returns an ordered set of relevant services that match the query in terms of both crisp logic based and syntactic similarity. For this purpose, it applies five different hybrid matching filters with one selected token based IR similarity metric each. Logical subsumption failures produced by the integrated OWL-DL reasoner Pellet are tolerated, if the computed syntactic similarity value is sufficient. Experimental evaluation of the performance of hybrid over logic based only matching provided strong evidence in favor of the above claim in terms of both recall and precision of service retrieval [5]. This evidence is further supported by experimental results reported in [1].

In this paper, we build upon this work and provide an analysis of the retrieval performance of OWLS-MX in terms of false positive and false negatives to reveal the benefits and pitfalls of both its logic based and hybrid matching. The analysis results have been exploited in an improved version of OWLS-MX, called OWLS-MX+, but can be of inherent interest to any developer of hybrid service matchmakers for the semantic Web in general.

The remainder of this paper is structured as follows. Assuming the reader to be familiar with OWL-S, we first summarize the basic idea and hybrid matching filter definitions of OWLS-MX in section 2, followed by a detailed analysis of their false positives and false negatives in section 3. Section 4 briefly presents OWLS-MX+ that has been developed based on the results of this analysis. Finally, section 5 concludes with a summary and open problems.

2 OWLS-MX

The core idea of the OWLS-MX matchmaker is to complement crisp logic based with approximate IR based matching where appropriate to improve the retrieval performance. It takes any OWL-S service as a query, and returns an ordered set of relevant services that semantically match the query each of which annotated with its individual degree of logical matching, and the syntactic similarity value. The user can specify the desired degree, and individual syntactic similarity threshold.

2.1 Matching algorithm overview

OWLS-MX performs signature based service matching only, that is, it compares the input and output parameter values of given OWL-S 1.1 service (query) with those of a given service. More concrete, it first classifies the service query I/O concepts in OWL-DL into its local ontology that contains all I/O concept definitions of advertised services. We assume that the type of computed terminological subsumption relation determines the degree of semantic relation between any pair of I/O concepts.

Second, OWLS-MX then pairwisely determines the degree of logical (concept subsumption) match according to its filter definitions, and the syntactic similarity between the conjunctive service and query I/O concept expressions. These expressions are built by recursively unfolding each query and service input (output) concept in the local matchmaker ontology. The unfolded concept expressions are including primitive components of a basic shared vocabulary only.

Any failure of logical concept subsumption produced by the integrated description logic reasoner of OWLS-MX will be tolerated, if and only if the degree of syntactic similarity between the respective unfolded service and query concept expressions exceeds a given similarity threshold. The detailed matching algorithm with a brief example are given in [5].

2.2 Matching filters

Let T be the terminology of the OWLS-MX matchmaker ontology specified in OWL-DL; CT_T the concept subsumption hierarchy of T; $LSC(C)$ the set of least specific concepts (direct children) C' of C, i.e. C' is immediate sub-concept of C in CT_T; $LGC(C)$ the set of least generic concepts (direct parents) C' of C, i.e., C' is immediate super-concept of C in CT_T; $Sim_{IR}(A, B) \in [0, 1]$ the numeric degree of syntactic similarity between strings A and B according to chosen IR metric IR with used term weighting scheme and document collection, and $\alpha \in [0, 1]$ given syntactic similarity threshold; $\doteq$ and $\dot{\geq}$ denote terminological concept equivalence and subsumption, respectively. The matching filters of OWLS-MX are as follows.

Exact match. Service S EXACTLY matches request R $\Leftrightarrow \forall$ IN_S $\exists$ IN_R: $\text{IN}_S \doteq \text{IN}_R$ $\wedge$ $\forall$ OUT_R $\exists$ OUT_S: $\text{OUT}_R \doteq \text{OUT}_S$.

Plug-in match. Service S PLUGS INTO request R $\Leftrightarrow \forall$ IN_S $\exists$ IN_R: $\text{IN}_S \dot{\geq} \text{IN}_R$ $\wedge$ $\forall$ OUT_R $\exists$ OUT_S: $\text{OUT}_S \in \text{LSC}(\text{OUT}_R)$. Relaxing the exact matching constraint, service S may require less input than it has been specified in the request R. This guarantees at a minimum that S will be executable with the provided input iff the involved OWL input concepts can be equivalently mapped to WSDL input messages and corresponding service signature data types. We assume this a necessary constraint of each of the subsequent filters. In addition, S is expected to return more specific output data whose logically defined semantics is exactly the same or very close to what has been requested by the user.

Subsumes match. Request R SUBSUMES service S $\Leftrightarrow \forall$ IN_S $\exists$ IN_R: $\text{IN}_S \dot{\geq} \text{IN}_R$ $\wedge$ $\forall$ OUT_R $\exists$ OUT_S: $\text{OUT}_R \dot{\geq} \text{OUT}_S$. Compared to the plug-in filter the constraint of immediate output concept subsumption is relaxed. As a consequence, the returned set of relevant services is extended in principle.

Subsumed-by match. Request R is SUBSUMED BY service S $\Leftrightarrow \forall$ IN_S $\exists$ IN_R: $\text{IN}_S \dot{\geq} \text{IN}_R$ $\wedge$ $\forall$ OUT_R $\exists$ OUT_S: $(\text{OUT}_S \doteq \text{OUT}_R \vee \text{OUT}_S \in \text{LGC}(\text{OUT}_R))$ $\wedge$ $\text{SIM}_{IR}(\text{S, R}) \geq \alpha$. This filter selects services whose output data is more general than requested, hence, in this sense, subsumes the request. We focus on direct parent output concepts to avoid selecting services returning data which we think may be too general.

Logic-based fail. Service S fails to match with request R according to the above logic-based semantic filter criteria.

Nearest-neighbor match. Service S is NEAREST NEIGHBOR of request R $\Leftrightarrow$ $\forall$ IN_S $\exists$ IN_R: $\text{IN}_S \dot{\geq} \text{IN}_R$ $\wedge$ $\forall$ OUT_R $\exists$ OUT_S: $\text{OUT}_R \dot{\geq} \text{OUT}_S$ $\vee$ $\text{SIM}_{IR}(\text{S, R}) \geq \alpha$.

Fail. Service S does not match with request R according to any of the above filters.

The above filters are in the following total order according to the size of results they would return, i.e. according to how relaxed the semantic matching:

$$\text{EXACT} < \text{PLUG-IN} < \text{SUBSUMES} < \text{SUBSUMED-BY} <$$
$$\text{LOGIC-BASED FAIL} < \text{NEAREST-NEIGHBOR} < \text{FAIL}.$$

2.3 OWLS-MX variants

We implemented the following variants of OWLS-MX, called OWLS-M1 to OWLS-M4, each of which uses the same logic-based semantic filters but different (token based) IR similarity metric $\text{SIM}_{IR}(R, S)$ for content-based service I/O matching. The variant OWLS-MO performs logic based only semantic service I/O matching.

OWLS-M0. The logic-based semantic filters EXACT, PLUG-IN, and SUBSUMES are applied as defined in 2.2, whereas the hybrid filter SUBSUMED-BY is utilized without checking the syntactic similarity constraint.

OWLS-M1 to OWLS-M4. The hybrid semantic matchmaker variants OWLS-M1, OWLS-M2, OWLS-M3, and OWLS-M4 compute the syntactic similarity value SIM_{IR} (OUT_S, OUT_R) by use of the loss-of-information measure (M1), extended Jaccuard similarity coefficient (M2), the cosine similarity value (M3), and the Jensen-Shannon information divergence based similarity value (M4), respectively.

2.4 Implementation

The OWLS-MX matchmaker has been implemented in Java using the OWL-S API 1.1 beta with the tableaux OWL-DL reasoner Pellet developed at the university of Maryland (cf. `http://pellet.owldl.com/`). As the OWL-S API is tightly coupled with the Jena Semanic Web Framework, developed by the HP Labs Semantic Web research group (cf. `http://jena.sourceforge.net/`), the latter is also used to modify the OWLS-MX matchmaker ontology.

The OWLS-MX matchmaker in its current version 1.1c is accessible via a convenient graphical user interface. It also provides a module (OWLS-MXP) for service I/O compatibility checking on the WSDL grounding level based on respective XMLS data type checking. OWLS-MX is available as open source from the portal semwebcentral.org [1].

3 R/P Performance Analysis

In this section, we first provide an overview of the R/P performance of the OWLS-MX variants followed by an analysis of false positives and false negatives returned by OWLS-M0 and the hybrid variants. That is, we are interested in typical cases where logic based (OWLS-M0) matching benefits from complementary approximative reasoning (OWLS-M1 to OWLS-M4) on the description of service semantics, where it fails, and vice versa.

[1] http://projects.semwebcentral.org/projects/owls-mx/

3.1 Overall R/P performance of OWLS-MX

For measuring the service retrieval performance of each OWLS-MX variant we used the OWL-S service retrieval test collection OWLS-TC V2 [2], and adopted the evaluation strategy of micro-averaging the individual recall/precision (R/P) curves. The micro-averaged R/P curves of the top and worse performing IR similarity metric together with those for the OWLS-MX variants are shown in figure 3.1.

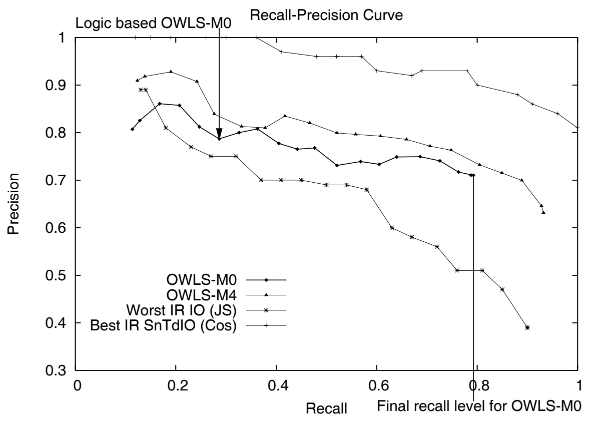


Fig. 1. R/P performance of OWLS-MX variants

This preliminary result reported in [5] provides strong evidence in favor of the proposition that building semantic Web service matchmakers purely on crisp logic based reasoning may be insufficient.

Quantitative analysis A preliminary quantitative analysis of these results in [5] showed that even the best IR similarity metric (Cosine/TFIDF) alone performed close to the pure logic based OWLS-M0 which can be significantly outperformed by hybrid semantic matching with OWLS-M1 to OWLS-M4 in terms of both recall and precision. Second, the hybrid matchmakers OWLS-MX, in turn, can be outperformed by each of the selected syntactic IR similarity metrics to the extent additional parameters with natural language text content are considered.

At this point, we add that the cosine IR similarity metric and OWLS-M4 using the Jensen-Shannon similarity metric performed best for syntactic and hybrid service profile I/O matching, respectively. Though, all hybrid OWLS-MX variants showed almost equal performance in average. These experimental results, of course, largely depend on the service retrieval test collection used.

[2] Available at `http://projects.semwebcentral.org/projects/owls-tc/`.

Building upon this preliminary quantitative analysis of the overall R/P performance, we continued our evaluation experiments to determine the main reasons of the false positives and false negatives of the logic based OWLS-M0 in more detail. In the following sections, we summarize the results of the respective qualitative analysis.

3.2 Logic based false positives

The qualitative analysis of irrelevant services returned by OWLS-M0 (false positives lower precision) in the experiments revealed the following characteristic reasons of their occurence in its answer sets.

1. The subsumption based distance between concepts in the matchmaker ontology insufficiently captures their real-world semantics (RWS) to be detected by logic based OWL-M0 filters. In contrast, a hybrid matching filter can mitigate this problem by determining the syntactic similarity between the respective concept expressions.
2. The surjective mapping of I/O concepts by M0 filters tolerates missing of concepts that are key for description of service semantics.
3. Same I/O concepts are used with inherently different semantics which cannot be detected by OWLS-M0.

These types of logic based false positives of OWLS-M0 are illustrated by example in the following.

Granularity of matchmaker ontology Of course, if the concept subsumption (or parent-child) relations in the ontology insufficiently capture the corresponding real-world semantics (RWS) any logic based filter risks to produce false positives.

For example, in figure 3.2, the service at best plug-in matches with the query, since their equally named output concepts "price" are equivalent, and the query input "HybridRotaryEnginePoweredCar" is far more specific than that of the considered service, that is "Automobile". In fact, the semantic distance between both input concepts in the ontology is probably too large for being of any interest to the user in practice. The reason why all logic based matching filters of OWLS-M0 fail in such a case is that they accept an unlimited input concept distance.

On the other hand, the subsumed-by filter of the hybrid variants like OWLS-M2 showed a better performance in these cases as the low syntactic similarity between the unfolded concept expressions treated as strings indicates a high degree of irrelevance. The relevant parts of the logical filter definitions are marked in red: first one is Plug-in(S,Q) "service S plugs into query Q", second is subsumes(Q,S) "query Q subsumes service S", third is subsumed-by(Q,S) "query Q is subsumed by the service S" with LSC and LGC denoting the set of least specific (direct child) and generic (direct parent) concepts.

The second example shown in figure 3.2 refers to the case of service and query output concepts with close distance in a coarse-grained matchmaker ontology but

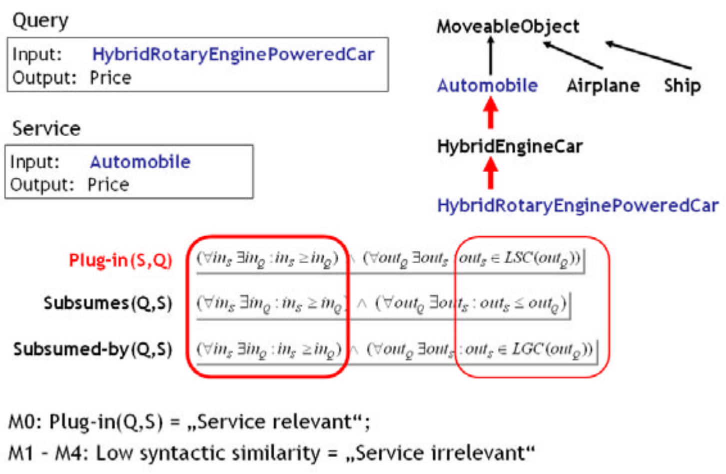

Query
Input: HybridRotaryEnginePoweredCar
Output: Price
Service
Input: Automobile
Output: Price
MoveableObject
Automobile
Airplane
Ship
HybridEngineCar
HybridRotaryEnginePoweredCar
Plug-in(S,Q)
$(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LSC(out_Q))$
Subsumes(Q,S)
$(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \leq out_Q)$
Subsumed-by(Q,S)
$(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LGC(out_Q))$
M0: Plug-in(Q,S) = „Service relevant";
M1 - M4: Low syntactic similarity = „Service irrelevant"

Fig. 2. Example: Unlimited input concept distance accepted by OWLS-M0

high RWS distance. In this example, even a least generic concept match of the logic based Subsumed-By(Q,S) filter of OWLS-M0 does not help.

Surjective mapping of concepts Another reason of false positives produced by the logic based OWLS-M0 is due to the surjective mapping of service and query concepts. In fact, the service and query input concepts might not be correlated by logical filters though they are carrying inherent semantics.

In figure 3.2, for example, the input "SFNovel" of the query "SFNovelPrice" does not match with any input of the service "EntranceFee" but "Author" with "Person". As a result, OWLS-M0 determines a plug-in(S,Q) or Subsumed-By(Q,S) match, hence the service relevant to the query though it definitely is not. The reason of this type of false positive is that the surjective mapping of service input concepts by OWLS-M0 filters tolerates the missing of concepts that are key for the description of query semantics.

Incomplete coverage of service semantics Another type of logic based false positives is caused by the insufficient coverage of service semantics by the definitions of I/O concepts used.

For example, the real world semantics of service "BookCopyCheck" and query "BookReview" in figure 3.2 are not related at all but both are determined by OWLS-M0 as semantically equivalent. The concept "Book" is used twice in the service input but with different semantics than in the query. In these cases, even our syntactic similarity measurements returned high relevance but at least not identity.

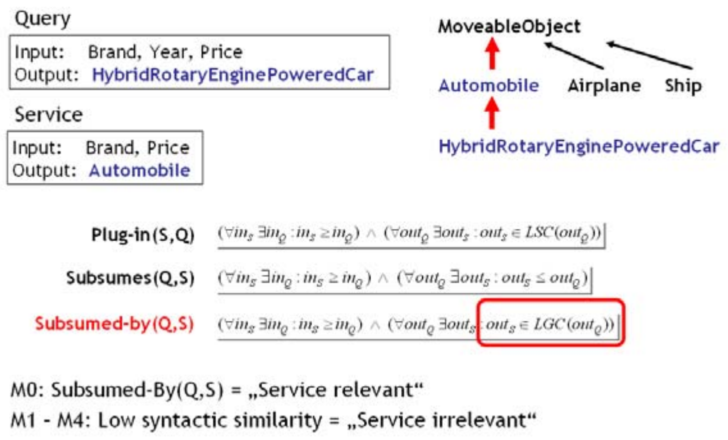

Query
Input: Brand, Year, Price
Output: HybridRotaryEnginePoweredCar
Service
Input: Brand, Price
Output: Automobile
MoveableObject
Automobile
Airplane
Ship
HybridRotaryEnginePoweredCar
Plug-in(S,Q) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LSC(out_Q))$
Subsumes(Q,S) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \leq out_Q)$
Subsumed-by(Q,S) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LGC(out_Q))$
M0: Subsumed-By(Q,S) = „Service relevant“
M1 - M4: Low syntactic similarity = „Service irrelevant“

Fig. 3. Example: Close output concepts with high RWS distance

3.3 Mitigating logic based FP problems

All FP problems described above call for the complementary use of text retrieval similarity metrics with fine-grained syntactic overlap measurement. In fact, our experimental results show that the number of false positives can be drastically reduced by using hybrid filters - which leads to a significant improvement in precision as shown in figure 3.3 for the case of applying OWLS-M0 vs OWLS-M2 to the respective test collection.

However, the same experiments also revealed that even hybrid OWLS-MX variants return false positives which we briefly illustrate next.

3.4 False positives of hybrid matching

Main reasons for hybrid OWLS-MX variants to return false positives are that all filters (a) do not require a total mapping of I/O concepts between service and query, and (b) used syntactic similarity measurements ignore the semantics of logical operators in concept expressions. In fact, our experiments showed that adding services of the above types of FP to the OWLS-TC2 causes a significant decrease in the precision of all OWLS-MX variants.

Surjective mapping of concepts According to their definition, no filter of OWLS-MX requires a total (bijective) mapping of service and query I/O concepts which can lead to false positives in situations where no concepts are provided .

For example, in figure 3.4, the query "BuyBook" and service "DatingService" are returned as equivalent since there is no query output concept to be matched.

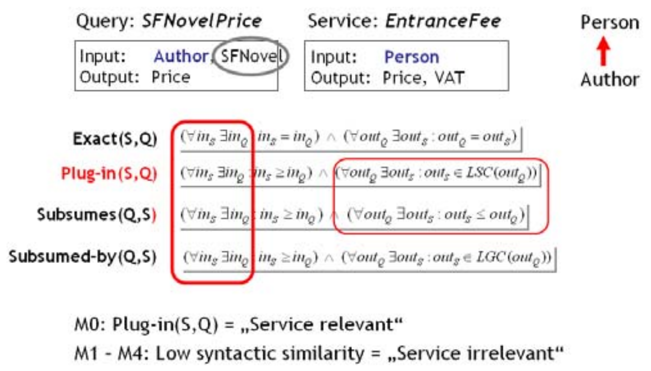

Query: SFNovelPrice
Input: Author, SFNovel
Output: Price
Service: EntranceFee
Input: Person
Output: Price, VAT
Person
Author
Exact(S,Q) $(\forall in_S \exists in_Q : in_S = in_Q) \wedge (\forall out_Q \exists out_S : out_Q = out_S)$
Plug-in(S,Q) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LSC(out_Q))$
Subsumes(Q,S) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \leq out_Q)$
Subsumed-by(Q,S) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LGC(out_Q))$
M0: Plug-in(S,Q) = „Service relevant“
M1 - M4: Low syntactic similarity = „Service irrelevant“

Fig. 4. Example: Tolerated missing of key concepts for service or query semantics

Logical operator semantics Another reason of false positives returned by hybrid variants is that the used syntactic similarity means ignore the logical language operators in concept definitions. In fact, connectives like "and", "or" are eliminated as stop words in the preprocessing step of IR similarity measurement. For example, in figure 3.4, the query "B/W-Print" is asking for print outs in black or white, while the service "BW-Print" is offering print outs in black and white. The service is logically irrelevant but identical for syntactic similarity metrics.

3.5 Logic based false negatives

The experimental analysis of relevant services missed by the logic based matchmaker OWLS-M0 (false negatives lower recall) revealed the following main characteristic reasons of failure.

1. In fine-grained matchmaker ontology, service concepts can logically differ from those of a given query but have similar real-world semantics (e.g. semantically close siblings in the ontology).
2. Opposed to the all-quantified matching filter constraints of OWLS-M0, the pairwise comparison of service and query I/O concepts does not always yield the same type of subsumption relationship.
3. In OWLS-M0, all service input concepts have to be equal or more generic than provided as it would, in case of a linear mapping of concepts to WSDL grounding data types, guarantee service invocation but may be too restrictive in certain cases.

Concept siblings Logic based false negatives can be caused by logically disjoint definitions of sibling concepts with similar real-world semantics. For example, in figure 3.5, the real world semantics of query output "Hopital-Physician" and the service output "Emergency-Physician" are quite similar.

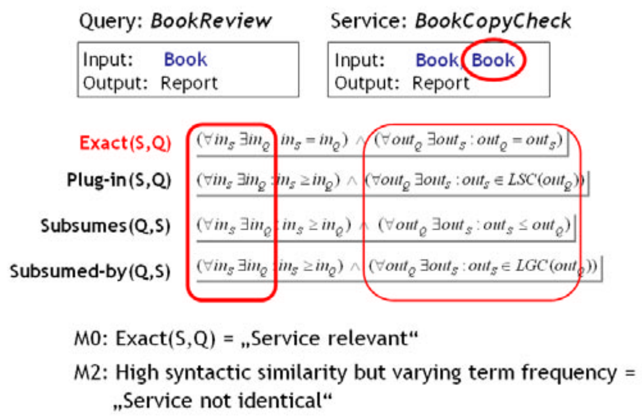

Query: BookReview
Input: Book
Output: Report
Service: BookCopyCheck
Input: Book, Book
Output: Report
Exact(S,Q)
Plug-in(S,Q)
Subsumes(Q,S)
Subsumed-by(Q,S)
$(\forall in_S \exists in_Q : in_S = in_Q) \wedge (\forall out_Q \exists out_S : out_Q = out_S)$
$(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LSC(out_Q))$
$(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \leq out_Q)$
$(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LGC(out_Q))$
M0: Exact(S,Q) = „Service relevant“
M2: High syntactic similarity but varying term frequency = „Service not identical“

Fig. 5. Example: False positive due to incomplete coverage of semantics

In the extreme, only one pair of conjunctive logical constraints or primitive components of siblings in the concept hierarchy remains unmatched during subsumption computation, hence causes a logical failure by OWLS-M0. Since both unfolded concept expressions are syntactically identical otherwise, any hybrid OWLS-MX would return both concepts as similar with respect to their implicit semantics exploited by selected IR similarity metric

Different subsumption relations Another major class of logic based false negatives is caused by the all-quantified matching filter constraints of OWLS-M0 that requires each pair of service and query I/O concepts to have the same type of subsumption relation.

For example, in figure 3.5, the types of subsumption relations between the output concepts of query "CarPlusBike" and service "4WheeledCarPackage" are different. As a result, the logic based OWLS-M0 fails while the hybrid variants determin the service as relevant, if the degree of the syntactic similarity between the considered pairs of concepts is sufficient.

Enforcement of more generic service input Finally, in OWLS-M0, all service input concepts have to be equal or more generic than provided which might not be the case though the service is relevant to the given query as shown in figure 3.5

3.6 Mitigating logic based FN problems

As mentioned above, both kinds of identified FN problems call for the complementary use of text retrieval similarity metrics with fine-grained syntactic overlap measurement.

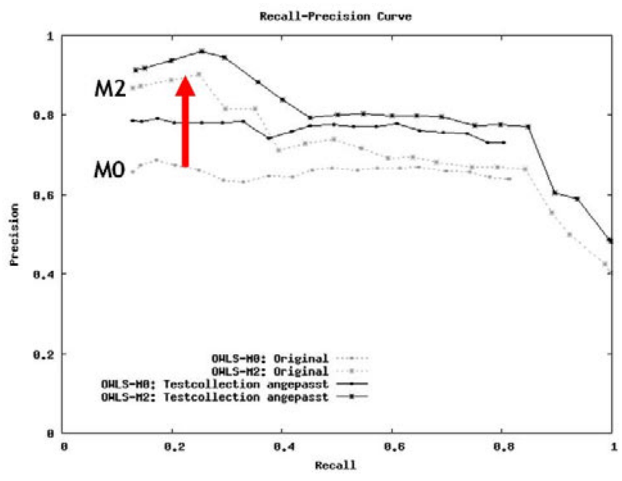


Fig. 6. Mitigating logic based false positives with hybrid matching filters that exploit syntactic similarity measurement.

The experimental results depicted in figure 3.6 show that the recall of all hybrid variants is significantly better than the one of OWLS-M0 applied to a test collection with sets of services that cause logic based false negatives. Main reason for this is, that the additional IR based similarity check of the nearest-neighbor filter allows OWLS-M1 to -M4 to find relevant services that the logic based OWLS-M0 would fail to retrieve.

4 Advancing OWLS-MX retrieval performance

Regarding the results of the FP/FN analysis of the retrieval performance of OWLS-MX in the previous section, we are interested in how most of the identified reasons of failure could be overcome. Basic idea is to improve the R/P performance by allowing for an integrated rather than just complementary hybrid matching on the deeper level of service I/O concepts.

That is, the respectively updated matchmaker version OWLS-MX+ determines whether

– a total (bijective) mapping of service/query I/O concepts exists, and
– further relaxes the match by tolerating also individual pairs of service/query concepts (instead of considering their input and output as a whole) with their logical relation deviating from that required by the all-quantification filtering constraints, if their syntactic similarity is sufficient.

The experimental results show that the R/P performance of OWLS-MX+ is significantly improved over that of OWLS-MX (cf. figure 4). Please note that OWLS-MX+ is not available in the latest version 1.1c of OWLS-MX; we are currently working on the next release (OWLS-MX 2.0) where it will be integrated.

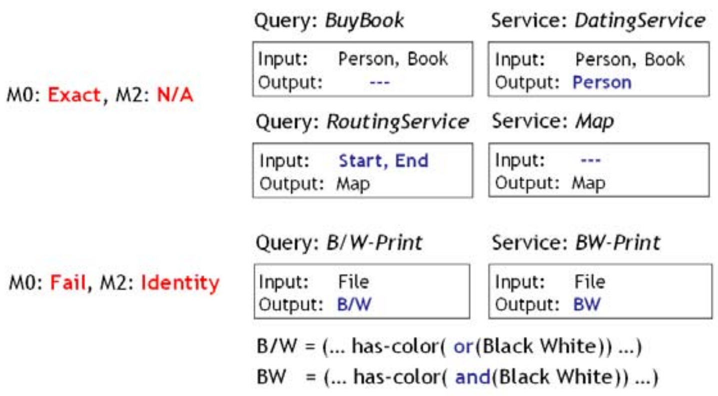


Fig. 7. Example: False positives of hybrid matching

5 Conclusions

Since its first release in May 2005, the OWLS-MX matchmaker is the only hybrid matchmaker for OWL-S services that applies both logic based and approximative matching filters. For an accessible account of other approaches to semantic Web service discovery, we refer to, for example, [3].

The results of our experimental analysis of hybrid semantic matching filters of OWLS-MX showed that the main characteristic problems of logic based only false positive/negatives can be largely mitigated by (a) syntactic overlap measurements of IR metrics, and (b) integrated hybrid matching on deeper concept level improves precision (OWLS-MX+).

However, one open problem of OWLS-MX+, as for any other logic based SWS matchmaker, is that the importance of individual concepts used to define the real world semantics of the considered service is highly subjective thus may vary for different users. This is hard to deal with in general. The problem of syntactic similarity metrics ignoring the semantics of logical operators "and", "or" in the expanded concept definitions as classical stop words will be solved in an updated version of OWLS-MX 2.0.

Finally, the query relevance sets of the used collection OWLS-TC2 are, of course, highly subjective. The construction of a large scale service retrieval collection has to be a joint effort of many people within and outside the SWS community. We encourage the reader to contribute to the evolution and enlargement of the OWLS-TC online at the sws-tc wiki http://www-ags.dfki.uni-sb.de/swstc-wiki

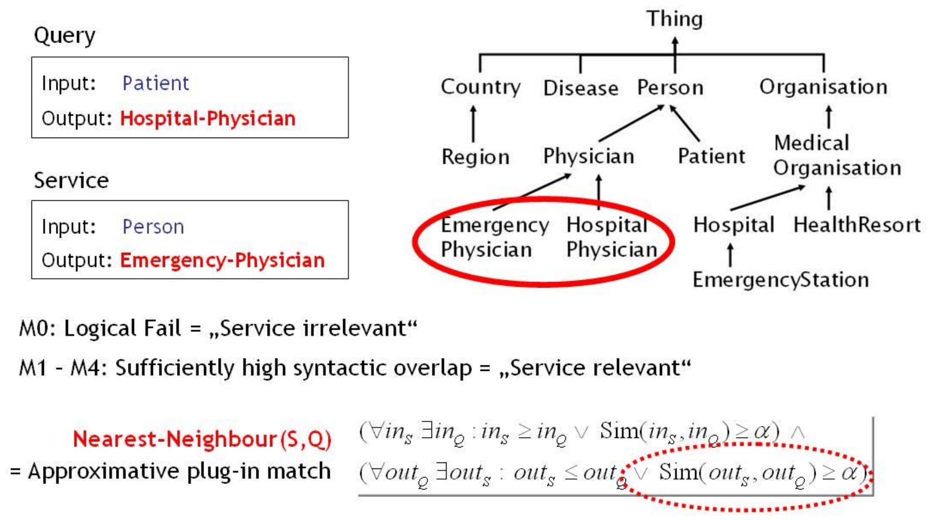

Query
Input: Patient
Output: Hospital-Physician
Service
Input: Person
Output: Emergency-Physician
Thing
Country
Disease
Person
Organisation
Region
Physician
Patient
Medical Organisation
Emergency Physician
Hospital Physician
Hospital
HealthResort
EmergencyStation
M0: Logical Fail = „Service irrelevant“
M1 - M4: Sufficiently high syntactic overlap = „Service relevant“
Nearest-Neighbour(S,Q)
= Approximative plug-in match
$(\forall in_S \exists in_Q : in_S \geq in_Q \vee \mathrm{Sim}(in_S, in_Q) \geq \alpha) \wedge$
$(\forall out_Q \exists out_S : out_S \leq out_Q \vee \mathrm{Sim}(out_S, out_Q) \geq \alpha)$

Fig. 8. Example: False negative for concept siblings in the ontology

References

1. A. Bernstein, C. Kiefer: Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins. Proc. ACM Symposium on Applied Computing, Dijon, France, ACM Press, 2006.
2. D. Fensel, F. van Harmelen: Unifying reasoning and search to Web scale. *IEEE Internet Computing*, March/April 2007.
3. S. Grimm. Discovery - Identifying Relevant Services. In: R. Studer, S. Grimm, A. Abecker (eds.). Semantic Web Services. Springer, 2007.
4. M. Klusch, K. Sycara: Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In: Coordination of Internet Agents: Models, Technologies and Applications. Springer, 2001.
5. M. Klusch, B. Fries, and K. Sycara. Automated Semantic Web Service Discovery with OWLS-MX. Proceedings of 5th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Hakodate, Japan, 2006
6. OWLS-MX: http://projects.semwebcentral.org/projects/owls-mx/.
7. OWLS-TC: http://projects.semwebcentral.org/projects/owls-tc/.

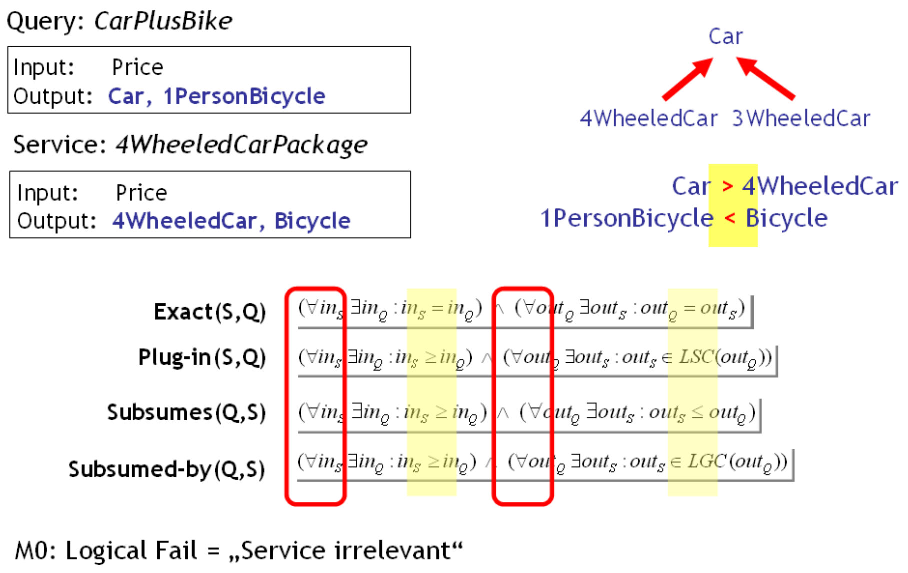

Query: CarPlusBike
Input: Price
Output: Car, 1PersonBicycle
Service: 4WheeledCarPackage
Input: Price
Output: 4WheeledCar, Bicycle
Car
4WheeledCar 3WheeledCar
Car > 4WheeledCar
1PersonBicycle < Bicycle
Exact(S,Q) $(\forall in_S \exists in_Q : in_S = in_Q) \wedge (\forall out_Q \exists out_S : out_Q = out_S)$
Plug-in(S,Q) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LSC(out_Q))$
Subsumes(Q,S) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \leq out_Q)$
Subsumed-by(Q,S) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LGC(out_Q))$
M0: Logical Fail = „Service irrelevant“

Fig. 9. Example: False negative due to different concept subsumption relations

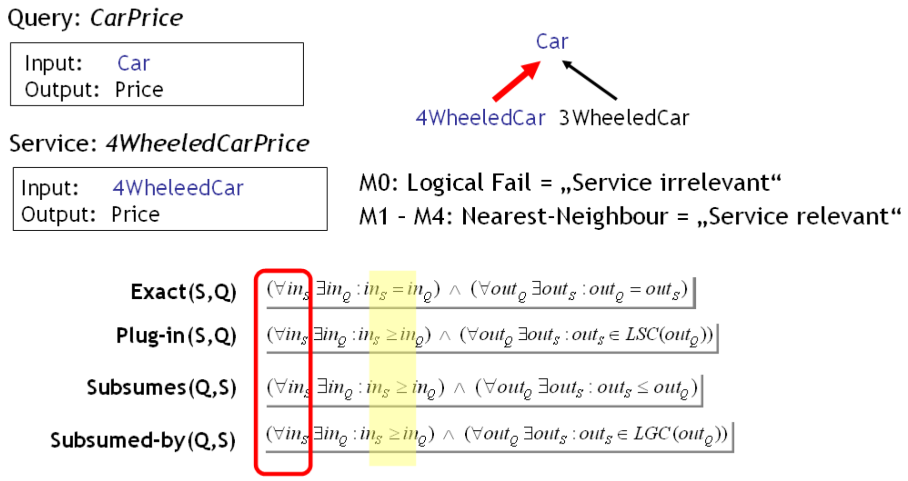

Query: CarPrice
Input: Car
Output: Price
Service: 4WheeledCarPrice
Input: 4WheeledCar
Output: Price
Car
4WheeledCar 3WheeledCar
M0: Logical Fail = „Service irrelevant“
M1 - M4: Nearest-Neighbour = „Service relevant“
Exact(S,Q) $(\forall in_S \exists in_Q : in_S = in_Q) \wedge (\forall out_Q \exists out_S : out_Q = out_S)$
Plug-in(S,Q) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LSC(out_Q))$
Subsumes(Q,S) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \leq out_Q)$
Subsumed-by(Q,S) $(\forall in_S \exists in_Q : in_S \geq in_Q) \wedge (\forall out_Q \exists out_S : out_S \in LGC(out_Q))$

Fig. 10. Example: False negative due to required more generic service input

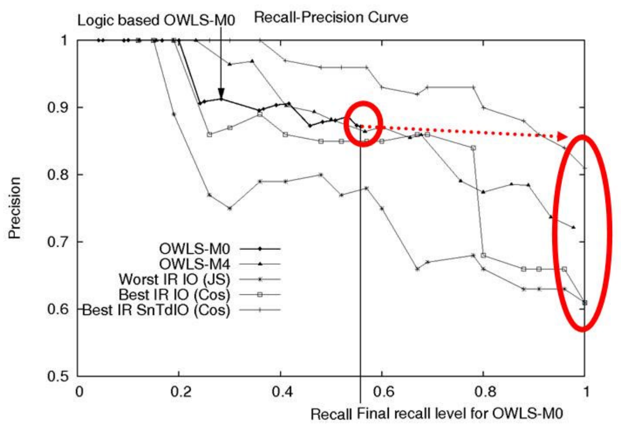


Fig. 11. Mitigating false negative problems of logic based OWLS-M0 with hybrid matching filters

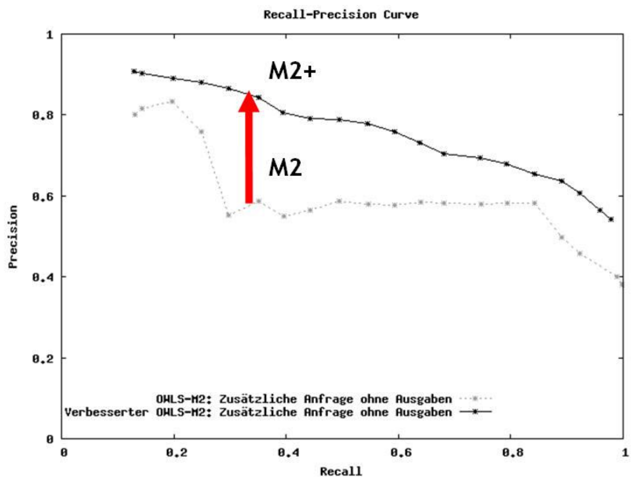


Fig. 12. Improved R/P performance with OWLS-MX+

Performance of Hybrid WSML Service Matching with WSMO-MX: Preliminary Results

Frank Kaufer[1] and Matthias Klusch[2]

[1] Hasso-Plattner-Institute at University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3, Potsdam,
frank.kaufer@hpi.uni-potsdam.de
[2] German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, Saarbrücken
klusch@dfki.de

Abstract. The WSMO-MX matchmaker applies different matching filters to retrieve WSMO-oriented service descriptions that are semantically relevant to a given query with respect to seven degrees of hybrid matching. These degrees are recursively computed by aggregated valuations of ontology based type matching, logical constraint and relation matching, and syntactic similarity as well. In this paper, we provide preliminary results of our experimental evaluation of the performance of WSMO-MX. In summary, it turned out that hybrid matching of WSML-MX services performs reasonably well.

1 Introduction

The problem of efficiently retrieving relevant services in the envisioned semantic web has been solved so far by only a few approaches for services described in OWL-S such as [1,2], and WSML such as [3,4,13]. Though, existing proposals for rule based service mediation in WSMO do not provide a general purpose matchmaking scheme for services in WSML. Recently, this gap has been filled by a hybrid semantic matchmaker, called WSMO-MX, that applies different matching filters to retrieve extended WSML services that are semantically relevant to a given query including the goal to be satisfied [5].

For this purpose, both services and goals are described in a Logic Programming (LP) variant of WSML, called WSML-MX, which is based on WSML-Rule. The hybrid matching scheme of WSMO-MX combines and extends the ideas of hybrid semantic matching realized by OWLS-MX [2], the object-oriented structure based matching proposed by Klein & König-Ries [6], and the concept of intentional matching introduced by Keller et. al [7]. WSMO-MX v0.4 is available at http://projects.semwebcentral.org/projects/wsmomx/.

In this paper, we build upon this work and show the results of our experimental evaluation of the performance of WSMO-MX based on a first, admittedly small service retrieval test collection for WSML services derived from the DIANE test collection.

The remainder of this paper is structured as follows. Section 2 provides an overview on how WSMO-MX works, while the testing environment and the preliminary results of the evaluation of its retrieval performance is given in section 3. Related work is strived in section 4, and section 5 concludes this paper.

2 WSMO-MX Overview

In this section, we briefly summarize the functionality of the WSMO-MX matchmaker and provide a brief example. For further details on the functionality and implementation of WSMO-MX, we refer to [5].

2.1 Service description in WSML-MX

WSMO-MX pairwisely matches services in a formally grounded variant of WSML called WSML-MX directly in F-Logic [8, 9]. It adopts the main and clearly motivated elements required for service matching from WSML, that are *goal*, *service*, *capabilities*, *preconditions*, and *postconditions* but not $effect$ and *assumption*. Central to describing services in WSML-MX is the notion of *derivative* which is an extended version of the object set introduced by Klein and König-Ries [6].

A derivative D_T in WSML-MX encapsulates an ordinary concept T (in this context called type) defined in a given ontology by attaching meta-information mainly about the way how T can be matched with any other type. Such information is defined in terms of different meta-relations of the derivative D_T. The type T is defined to be either atomic or a complex type with relations, the derivative D_T can also have a set of relations different from T, though this set is empty by default. A state is a set of state parts, which are derivatives each defined as atomic, or as complex by means of relations with derivatives as range. Hence, any service in WSML-MX can be represented as a directed object-oriented graph with derivatives considered as nodes and relations between them as edges, as shown in figure 1.

This variant of WSML allows for constraints on both relations and derivatives formulated in the full Horn fragment of F-logic. Hence, WSML-MX constraints are as expressive and, in general, only semi-decidable as are WSML-Rule axioms. However, the WSMO-MX matchmaker approximates query containment through means of relative query containment for constraint matching. Moreover, the matching of parts of WSML-MX expressions represented as acyclic object-oriented graphs without constraints is decidable in polynomial time.

The emphasis of WSML-MX on these parts of service modelling is motivated not only by clear separation of computationally tractable elements but the fact that it allows the matchmaker for a more detailed explanatory feedback to the user and more differentiated matching valuations. This is a lesson learned from matchmaking approaches relying on pure query containment which requires high ontological homogeneity and results in single match predicates based on overall and undifferentiated logical implication between goal and service descriptions.

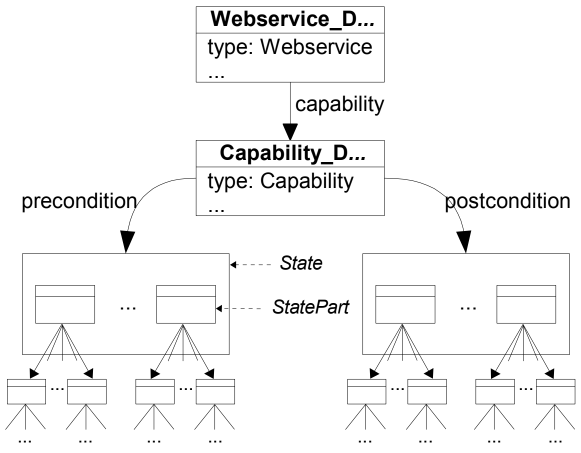


Fig. 1. Service derivative in WSML-MX

An example for a service in WSML-MX is shown in figure 2; the service offers tickets for any trip between any two German towns, but if the user departs from Berlin, her destination must be Hamburg.

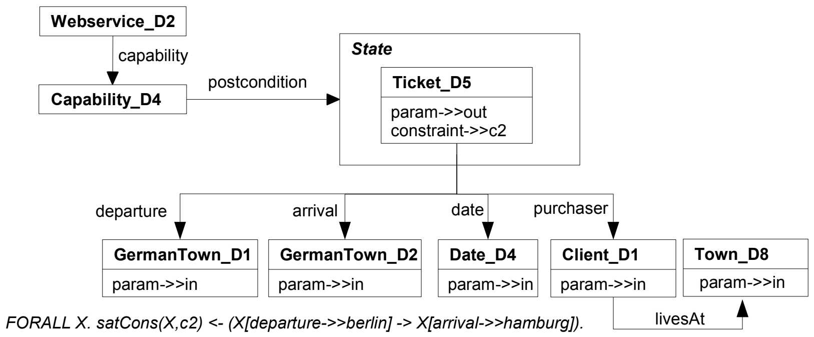


Fig. 2. Example service in WSML-MX

2.2 Hybrid matching degrees

The result of matching a derivative D_G from a goal description with a derivative D_W from a service description is a vector $v \in R^7$ of aggregated valuations of (a) ontology based type matching, (b) logical constraint matching, (c) recursive relation matching, and (d) syntactic matching. In this respect, the matching of WSMO-MX is hybrid.

Each real-valued entry in the so called (matching) valuation vector $v = (\pi_{\equiv}, \pi_{\sqsubseteq}, \pi_{\sqsupseteq}, \pi_{\sqcap}, \pi_{\sim}, \pi_{\circ}, \pi_{\perp})$ with $\pi_i \in [0,1]$ $(i \in \{\equiv, \sqsubseteq, \sqsupseteq, \sim, \sqcap, \circ, \perp\})$ and $\sum \pi_i = 1$, denotes the extent (similarity score) to which both derivatives D_G and D_W match with respect to the hybrid semantic matching degrees π_i of WSMO-MX. These degrees are the logical relations *equivalence*, $plug - in$ known from software component retrieval [10] or the similar *rule of consequences* from Hoare logic [11], $inverse - plugin$, *intersection* and *disjunction* $(fail)$ as degrees of *logic based* semantic match.

The degree of $fuzzy\, similarity$ refers to a non-logic based semantic match such as syntactic similarity or non-subconcept relations with respect to the type graph, while the degree *neutral* stands for neither match nor fail, hence declares the tolerance of matching failure. The set-theoretic semantics of the hybrid matching degrees are given in Table 1 based on the relations between the maximum possible instance sets of the derivatives D_G and D_W, denoted by $\mathcal{G}$ and $\mathcal{W}$. Since we use the heuristic relative query containment for the constraint matching, these sets are restricted to instances in the matchmaker knowledge base which satisfy the constraints. We acknowledge that it can not be taken for granted that the matchmaker is in possession of instances for every derivative with assigned constraints. However, these precedence instances could be retrieved by tracking service executions, sampling services/descriptions (services without real world effects) or - regarding goal derivative instances - by conducting systematic questionnaires. Furthermore, constraint matching could be ignored (configuration option in WSMO-MX) for coarse service discovery and applied for verification purposes in the service composition, when instances are available.

order	symbol	degree of match	pre	post
1	$\equiv$	equivalence	$\mathcal{G} = \mathcal{W}$	
2	$\sqsubseteq$	plugin	$\mathcal{G} \subseteq \mathcal{W}$	$\mathcal{W} \subseteq \mathcal{G}$
3	$\sqsupseteq$	inverse-plugin	$\mathcal{G} \supseteq \mathcal{W}$	$\mathcal{W} \supseteq \mathcal{G}$
4	$\sqcap$	intersection	$\mathcal{G} \cap \mathcal{W} \neq \emptyset$	
5	$\sim$	fuzzy similarity	$\mathcal{G} \sim \mathcal{W}$	
6	$\circ$	neutral	*by derivative specific definition*	
7	$\perp$	disjunction (fail)	$\mathcal{G} \cap \mathcal{W} = \emptyset$	

Table 1. Degrees of hybrid semantic matching of WSML service and goal derivatives

2.3 Hybrid matching process

In order to compute the degrees of hybrid semantic matching of given goal and service derivatives in WSML-MX, WSMO-MX recursively applies different IOPE matching filters to their preconditions and postconditions (inherently including service inputs and outputs as in WSML, but with an explicit parameter flag similar to the variables in [6]), and returns not only the aggregated matching valuation vector but also annotations of the matching process results as a kind of explanatory feedback to the user. That facilitates a more easy iterative goal refinement by the user in case of insufficient matching results. The annotations have a generic format and can be employ for several purposes. In the current version, WSMO-MX uses them to generate natural language explanations of the respective matching deviations. In the future they could also be used for graph-based visualizations of the matching result.

More concrete, the state of the goal is matched with that of the service by matching their state part derivatives and then recursively by the pairwise matching of relation range derivatives of equally named relations. Subsequently, WSMO-MX computes the maximum weighted bipartite graph match, where nodes of the graph correspond to the goal and service state parts. The respectively computed valuation vectors act as weights of edges existing between matched state parts.

At each step in the recursion, the parameter matching filter is applied first, since its result, an annotation record, is not valuated for any of the hybrid matching degrees. Then each of the logic based semantic matching filters (type, constraint, and relation matching) is applied. While type matching, in essence, bases on the subconcept relations and path distance between types (classes) in the matchmaker ontology, F-logic constraint matching is computed by means of relative query containment, and relation matching recursively matches the ranges of equally named relations with each other. Syntactic matching is performed in case one of these filters fails (compensative), or complementary in any case, if not specified differently. The user can also ask for just a first coarse-grained filtering by means of exclusively syntactic matching without any semantic matching.

Finally, all valuation vectors computed during recursive matching of goal and service derivatives are aggregated into one single valuation vector. For aggregation, each individual valuation vector is weighted for the respective matching filter as specified by the user for the given goal; the weighting is assumed to be equal by default. This aggregated valuation of hybrid matching degrees is then recomputed with respect to the intentions of the considered derivatives.

The overall result of the matching process is a ranked list of services with their hybrid matching valuation vector and annotations. Services are ranked with respect to the maximum value of hybrid semantic matching degrees in descending order (cf. table 1), starting with $\pi_{\equiv}$.

2.4 Example

Goal, service, ontology. Suppose the user defines a goal derivative *Ticket_D4* as shown in figure 3. That is, she is looking for any ticket for a trip between two arbitrary towns, but if it starts in Berlin, then it must not end in Bremen. Please note, that the user may specify matching relaxations for any object of the goal as exemplified, but also different weights for the matching filters to be applied. In this example, we assume the filters to be equally weighted.
The part of the type hierarchy in the matchmaker ontology and all instances used in this example are shown in figure 4.

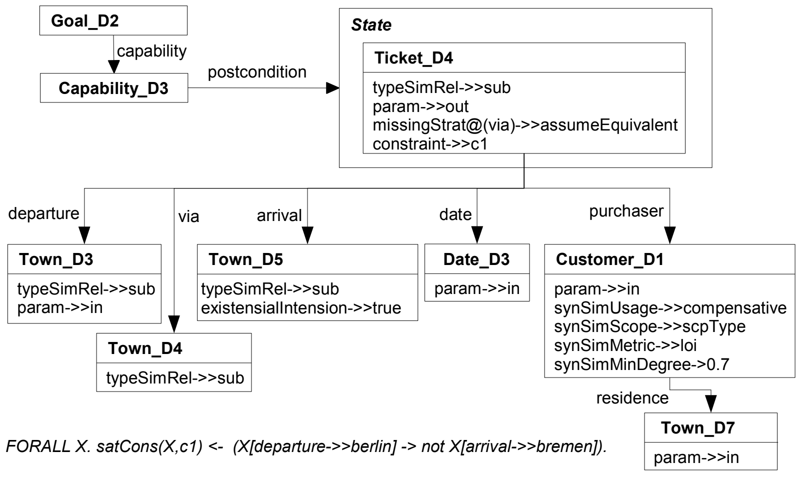
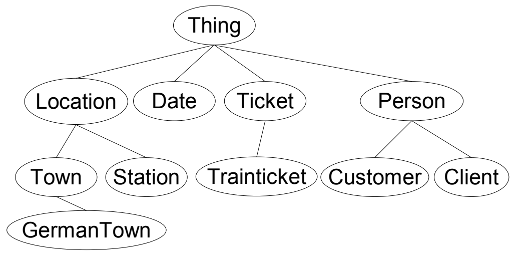


Fig. 3. Example goal in WSML-MX

In this example, the service derivative *Ticket_D5* given in section 2 will be matched against the goal derivative *Ticket_D4* as follows.

Matching. Since the capabilities of both goal and service derivatives do not include any precondition, the hybrid semantic matching of them is restricted to the matching of their postcondition states.

1. **match types**: the types of Ticket_D4 and Ticket_D5 are equal. Hence the valuation is $v_1 = (1, 0, 0, 0, 0, 0, 0)$.
2. **match parameters**: both are output parameters, no annotation necessary
3. **match relations**
   (a) *departure*: the types of Town_D3 and GermanTown_D1 are not equivalent, but Town_D3 allows subtypes. Since GermanTown is a subconcept of Town, the valuation is $v_2 = (0, 1, 0, 0, 0, 0, 0, 0)$.
   (b) *via*: this relation is not defined for Ticket_D3, but the missingStrategy for this relation is *assumeEquivalent* yielding a valuation $v_3 = (1, 0, 0, 0, 0, 0, 0, 0)$.
   (c) *arrival*: analogous to *departure* types of the ranges of *arrival* are subtypes and yield the valuation $v_4 = (0, 1, 0, 0, 0, 0, 0, 0)$.
   (d) *date*: is equal in goal and service, hence valuated as $v_5 = (1, 0, 0, 0, 0, 0, 0, 0)$
   (e) *purchaser*: type matching fails for Customer_D1 and Client_D1, but compensative syntactic matching is allowed using loss of information (LOI) metric. For the unfolding only the types of the derivatives should be used (*scpType*), yielding the term vectors $(Customer : 1, Town : 1, Person : 1, Location : 1, Town : 1)$ and $(Client : 1, Town : 1, Person : 1, Location : 1, Town : 1)$ for Customer_D1 and Client_D1,



```
t1:Ticket_D4[departure->>berlin;
             arrival->>leipzig; ...].
t2:Ticket_D4[departure->>berlin;
             arrival->>kiel; ...].
t3:Ticket_D5[departure->>hamburg;
             arrival->>bremen; ...].
t4:Ticket_D5[departure->>hamburg;
             arrival->>hannover; ...].
t5:Ticket_D5[departure->>berlin;
             arrival->>hamburg; ...].
t6:Ticket_D6[departure->>berlin;
             arrival->>bremen; ...].
```

Fig. 4. Example ontology (type hierarchy and instances)

respectively. The similarity degree is 0.75, and therefore greater than the declared minimum of 0.7. The resulting valuation vector is $v_6 = (0,0,0,0,0,0,1,0)$.

The aggregated relation valuation is $v_7 = \frac{v_2+\ldots+v_6}{5} = (0.4, 0.4, 0, 0, 0, 0.2, 0)$

4. **match constraints**: Ticket_D4 has the constraint c1. This is satisfied by the instances $t1, \ldots, t5$. The constraint c2, which is imposed on Ticket_D5 is satisfied by the instances $t3, \ldots, t5$. That means the instances for Ticket_D5 are a subset of those of Ticket_D4 and hence the valuation is $v_8 = (0,1,0,0,0,0,0,0)$

Finally, the aggregated valuation for the derivative matching of Ticket_D4 and Ticket_D5 is

$$v_9 = \frac{v_1+v_7+v_8}{3} = (\tfrac{7}{15}, \tfrac{7}{15}, 0, 0, 0, \tfrac{1}{15}, 0).$$

This means that the advertised service is hybrid semantically matching with the request. In particular, they are exactly and plug in matching to the same extent (0.46).

3 Evaluation of performance

The preliminary experimental evaluation of the retrieval performance of WSMO-MX focuses on measuring its recall, precision, and F1 based on an initial test collection.

3.1 Testing environment

At the time of writing, there is no service retrieval test collection for WSML available. As a consequence, for testing the performance of WSMO-MX, we had to develop an inital test collection.

Service Retrieval Test Collection WSML-TC1. For this purpose, we borrowed domain ontologies, service offers and requests from the DIANE project[3], and transformed parts of them from their F-DSD format into WSML-MX F-Logic. The resulting test collection WSML-TC1 contains approximately 300 concepts and over 800 instances in the domain ontology and 27 web services (offers) and 21 goals (requests) with over 1000 derivatives.

The goals belong to 5 different domains: book buy (10), cinema ticket booking (4), tv set buy (5), printing request (3), train ticket service (5). They are chosen such that they have large modeling overlaps making it more challenging for syntactical and hybrid matching approaches which rely on syntactic similarity. As usual, the relevance sets of WSML-TC1 were subjectively determined, that is whether a service is relevant for a request depends on a categorical point of view. That is motivated in part by the fact that semantic service matching shall be employed in a wide range of use case scenarios, from crisp composition planning to human oriented service discovery making it impossible for a general purpose matchmaker like WSMO-MX to take every possible prerequisite (e.g. available inputs/mediators, intended usage, etc) into account.

For the formal declaration of relevance in test environments, we define two further derivative meta relations: `declaredRelevanceTo` and `numberDeclaredRelevant`. The first one is used to assign a goal derivative with a service derivative, the latter counts all relevant web service derivatives by means of the following rule:

```
FORALL D,C D[numberDeclaredRelevant->>C] <-
      EXISTS R D[declaredRelevanceTo->>R] AND count(D,R,C).
```

Hardware. For the performance tests, WSMO-MX v0.5 and OntoBroker v4.3 were deployed on a machine with Win XP, Apache Tomcat 6 (also tested with 5.5), Java 6 (also tested with Java 5), 2.39 GHz, and 2 GB RAM.

3.2 Experiments

On the basis of the very first and admittedly small test collection WSML-TC1, we initially conducted six experiments to investigate the matchmakers behavior with respect to different configurations for semantic, syntactic, and hybrid matchings. We measured the retrieval performance of WSMO-MX in terms of its recall, precision, and F1-measure as known from information retrieval [12]. The unfolding of derivatives for the syntactic matching is done online, results are indexed only for one matching request, such that overall time needs are

[3] http://hnsp.inf-bb.uni-jena.de/wiki/index.php/DSD

comparable between syntactic and non-syntactic matchings. For non-testing environments at least all offers could be indexed in advance of the user request which reduces the effective matching time to some milliseconds.

Experiment 1: Pure logic based semantic matching At first we examined how well pure semantic matching performs, relaxing only derivative type deviations in three steps of increasing degree of fuzzyness as follows.

- *default*: Only type deviations explicitly granted in the goal descriptions are allowed
- *subSuper*: All service derivatives without explicitly defined type deviation are considered sufficient similar to each other if their types have a logical subconcept relationship
- *relation-3*: Types are only required to have a maximum distance of 3 in the taxonomy graph spanned by logical subconcept relation.

The strict configurations *default* and *subSuper* deliver solely relevant results. While the default configuration only achieves a recall of below 0.7 (Fig. 5), *subSuper* already delivers a full recall which is due to the fact that the test collection relies on one homogenous domain ontology. Only the fuzzier *relative-3* configuration attains 100% recall, but at the expense of a final 70% fallout and a time consumption of almost 1 sec per query which is similar to 0,89 sec per query for *subSuper*, but more than three times as much as needed for *default* matching (0,3 sec).

Experiment 2: Syntactic similarity metrics This experiment is about comparing the four provided metrics for syntactic matching: Jaccard, Extended-Jaccard, Multiset-Jaccard, and Cosine. Therefore the configuration was fixed to alternative syntactic matching with a minimum matching degree of 0.6. For the unfolding relations and types were considered up to a maximum depth of 2.

It could be observed that for the chosen threshold all metrics except Cosine deliver 100% or almost 100% correct results. But only Jaccard and Multiset-Jaccard also achieve a high recall of over 90%. Extended-Jaccard's recall is below 0.6, but additional variations showed that this metric is the least vulnerable with respect to precision. In total, figure 6 shows that Jaccard provides the best overall results. In this configuration, Multiset-Jaccard is only slightly worse.

It is remarkable that the syntactic matching achieves full precision and almost the same recall level as the pure semantic matching while consuming only a sixth of the overall computation time. Furthermore the most computation is needed for unfolding and index generation which could be done in advance of the matching. But please note that the computation time of the semantic matching is largely determined by the client-/server communication of the matchmaker and the reasoner. With the new reasoner API of OntoBroker 5 this can be reduced significantly and will be investigated in detail in future evaluations.

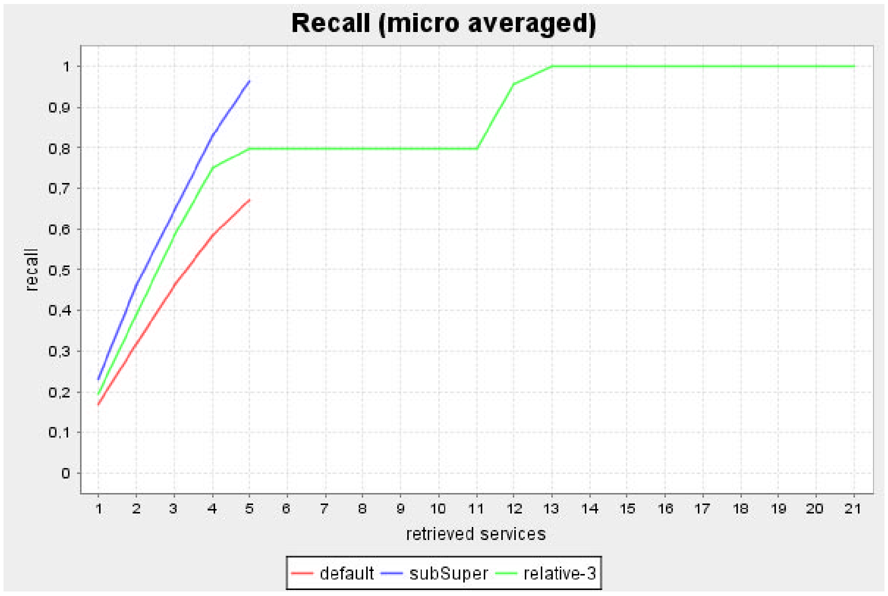


Fig. 5. Type matching

Experiment 3: Syntactic similarity threshold The minimal matching degree is crucial for WSMO-MX because syntactic matching is applied to single derivatives instead of whole service descriptions (like in OWLS-MX [2]). To find a suitable threshold, we used the best configuration from the last experiment (metric: Jaccard) and increased the threshold successively from 0.4 to 0.7 in steps of 0.05. Beginning with 0.6 the syntactic matching delivers only relevant documents, whereas 0.4 and 0.45 entail a final fallout of about 60% (50%) for their final but late full recall. A threshold of 0.6 achieved still a recall of 0.9 (0.65 - 80%, 0.70 - 70%). The results for the thresholds 0.5 and 0.55 are closer to those of 0.6-0.7 with a fallout of 10-20% and almost full recall. For this test collection 0.6 can be considered as most suitable threshold. For ontologies of more heterogeneous origin probably a lower threshold should be chosen.

Experiment 4: Unfolding service derivative scope In this experiment the scope of a derivative used for syntactic matching is varied. For the last experiments, both, types and relation were used. Now, this configuration is compared to only using types or relations. As can be seen from the F1-measure graph in figure 7, only combination of both leads to the best result. Separate consideration of recall and precision shows that types determine the recall (100% but final fallout of 40%) and the relations the precision (almost 100%, but recall only 50%). Syntactic matching with an unfolding of relations *and* types needs twice as much time as with only one of both scopes.

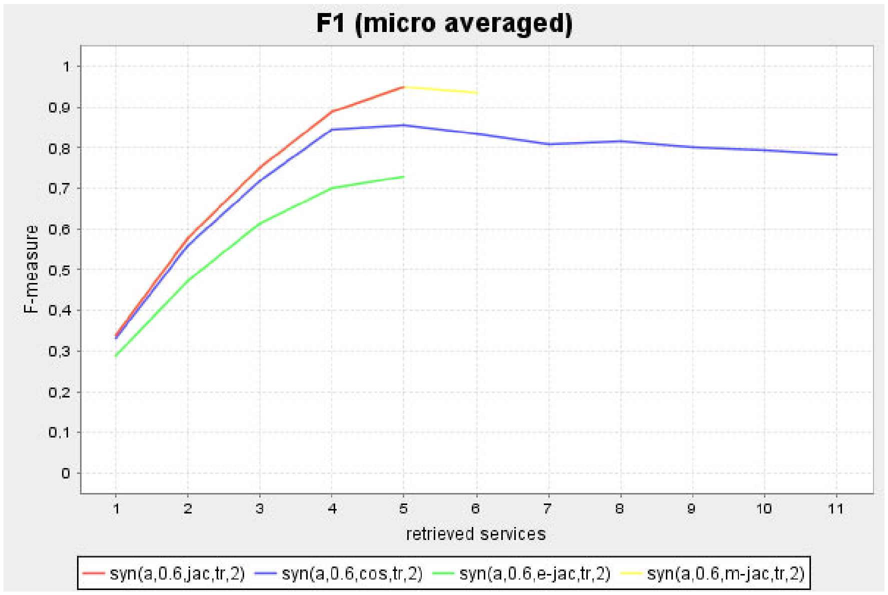


Fig. 6. Metrics experiment: Jaccard, Cosine, Extended Jaccard, Multiset Jaccard

Experiment 5: Unfolding depth Beside the scope the unfolding result of a derivative is dependent from the maximum unfolding depth. In this experiment we increased the depth starting with 0 which means that only the derivative itself is unfolded. From unfolding depth 3 on, no significant changes could be observed. Figure 8 clearly shows that only the unfolding depths 2 and 3 yield good results. Although the depths 0 and 1 achieve a final full recall, the high fallout of 80% (55%) is not acceptable. Unfolding (and matching) for depth 2 takes twice as much time as for 1 which in turn takes twice as much as for depth 0.

Experiment 6: Hybrid vs logic based matching Finally, we compared the performance of hybrid matching based on integrated and compensative syntactic matching with best syntactic only matching as well as *default* and *subSuper* matching from the first experiment. All four matchings have no fallout and only differ with respect to recall (figure 9) and computation time. Though not significant, the hybrid matching performed best, but only slightly better than *subSuper* and the pure syntactic matching. However, on average, the syntactic matching takes only 0,15 sec per query, whereas hybrid matching takes 0,61 and *subSuper* 0,89. Overall, this indicates the usefulness of syntactic matching in combination with, or instead of logical reasoning.

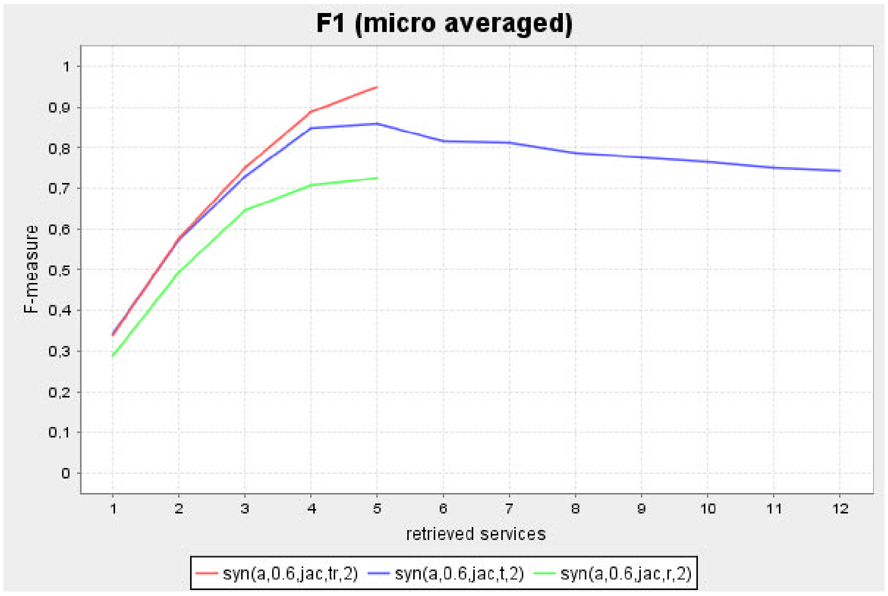


Fig. 7. Syntactic matching with different scopes: types (t), relations (r), both (tr)

4 Related work

WSMO-MX has been the first implemented full-fledged matchmaker for WSMO-oriented services. The mediator based discovery approaches and discovery models for WSML presented, for example, in [3, 4, 13] do not allow for general goal-service matching, but require problem specific mapping, or construction rules.

In general, semantic service matching determines whether the abstract semantic description of a requested service (or goal) conforms to that of an advertised service based on their functional and non-functional semantics. This is at the very core of any semantic service discovery framework. Current approaches to semantic service matching can be classified according to

- what kinds and parts of service semantics are considered for matching, and
- how matching is actually performed in terms of syntactic similarity measurements, logic based reasoning within or partly outside the service description framework, or a hybrid combination of both

Most matchmakers for the semantic Web today perform service profile matching while only very few perform process model matching or even a combined functional service matching. Since all semantic Web service description frameworks, except SAWSDL, provide a strict logic based profile semantics (IOPE), it comes at no surprise that most matchmakers rely on crisp logic based rather than hybrid semantic matching of semantic Web services of which only a few approaches exist. Process matching approaches or even combined approaches are more than rare leaving a lot of space for further research.

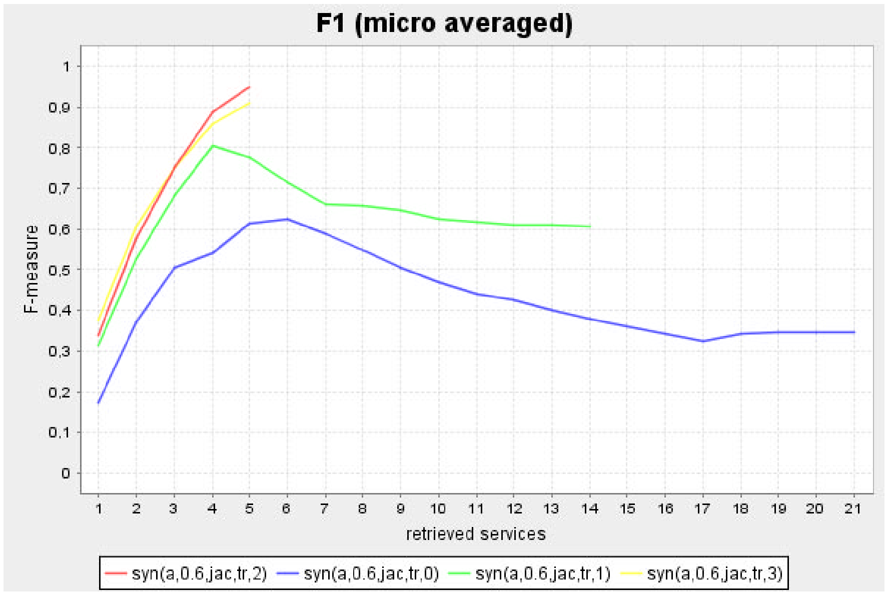


Fig. 8. Influence of unfolding depth

For WSMO-MX, we did improve on the idea of hybrid matching that has been applied to OWL-S service matching by OWLS-MX [2] and in general by LARKS [14] through allowing for a more fine-grained parametrisation, and integrated interleaving of syntactic and semantic matching. In any case, the lack of a sufficiently large and commonly agreed test collection for evaluating the performance of semantic Web service matchmakers for any of the current description frameworks is a general problem which can only be tackled by the community as a whole. In this respect, the presented results of the performance evaluation of WSMO-MX can only be considered preliminary.

5 Conclusions

WSMO-MX performs hybrid semantic web service matching based on both logic programming, and syntactic similarity measurement. It applies different matching filters to retrieve WSMO-oriented service descriptions that are semantically relevant to a given query with respect to seven degrees of hybrid matching. These degrees are recursively computed by aggregated valuations of ontology based type matching, logical constraint and relation matching, and syntactic similarity as well. WSMO-MX v0.4 is available at http://projects.semwebcentral.org/projects/wsmomx/. In this paper, we provided preliminary results of our experimental evaluation of the performance of WSMO-MX. In summary, it turned out that hybrid matching of WSML-MX services performs reasonably well, and can outperform crisp logic based match-

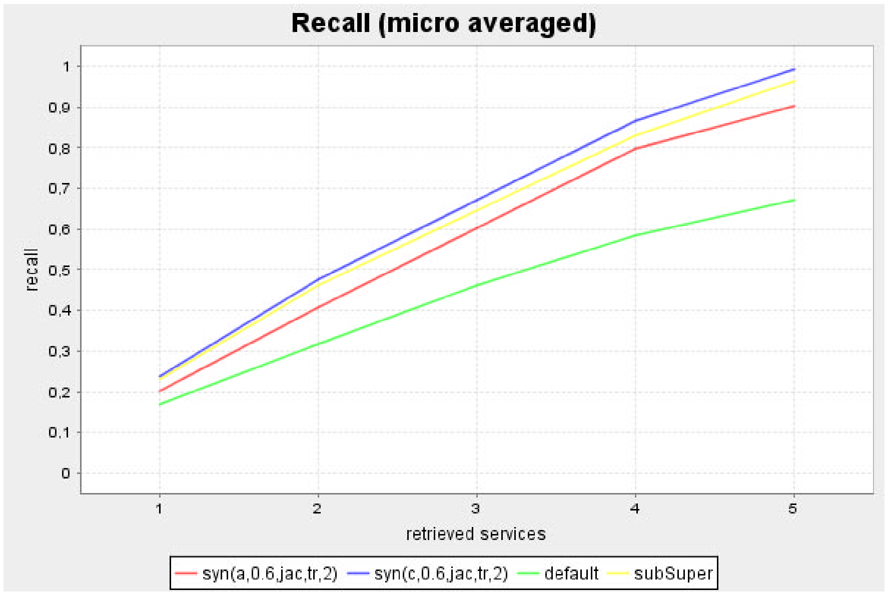


Fig. 9. Hybrid (with syntactic similarity) vs. pure logic based (default, subsumption) matching

ing. Furthermore, the experiments indicated that pure syntactic matching - if parametrized appropriately - can keep up with logic matching regarding recall/ precision and significantly outperforms it with respect to computation time. We are currently working on extending the test collection WSML-TC1, and the updating of WSMO-MX for an upcoming release.

References

1. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. Journal of Web Semantics **1**(1) (2003) 28
2. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with owls-mx. In: Proceedings of 5th International Conference on Autonomous Agents and multiagent Systems AAMAS, Hakodate, Japan (2006)
3. Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., Fensel, D.: A logical framework for web service discovery. In: Proceedings of the ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications. Volume 119., Hiroshima, Japan, CEUR Workshop Proceedings (November 2004)
4. Valle, E.D., Cerizza, D.: Cocoon glue: a prototype of wsmo discovery engine for the healthcare field. In: Proceedings of the WIW 2005 Workshop on WSMO Implementations. Volume 134., Innsbruck, Austria, CEUR Workshop Proceedings (June 2005)

5. Kaufer, F., Klusch, M.: Wsmo-mx: A logic programming based hybrid service matchmaker. In: Proceedings of the 4th IEEE European Conference on Web Services (ECOWS 2006), IEEE CS Press, Zurich, Switzerland. (2006)
6. Klein, M., König-Ries, B.: Coupled signature and specification matching for automatic service binding. In: Proceedings of European Conference on Web Services (ECOWS 2004). LNCS 3250, Erfurt, Germany, Springer (September 2004) 183
7. Keller, U., Lara, R., Lausen, H., Polleres, A., Fensel, D.: Automatic location of services. In: Proceedings of the 2nd European Semantic Web Symposium (ESWS2005), Heraklion, Crete (June 2005)
8. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM **42** (1995)
9. Angele, J., Lausen, G.: Ontologies in f-logic. In Staab, S., Studer, R., eds.: Handbook on Ontologies. Springer (2004) 29–50
10. Zaremski, A.M., Wing, J.M.: Specification matching of software components. In: 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering. (10 1995)
11. Hoare, C.: An axiomatic basis for computer programming. Communications of the ACM (CACM) **12**(10) (10 1969) 576–580
12. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press, Addison-Wesley. pages 75ff, ISBN 0-201-39829-X (1999)
13. Lara, R., Corella, M.A., Castells, P.: A flexible model for web service discovery. In: Proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives, Seoul, South Korea. (2006)
14. Sycara, K., Widoff, S., Klusch, M., Lu, J.: Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems **5** (June 2002) 173–2003

Semantic Web Services in the Web: A Preliminary Reality Check

Matthias Klusch and Zhiguo Xing

German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, 66121 Saarbruecken, Germany
klusch@dfki.de

Abstract. Where are all the semantic Web services today? In this paper, we briefly provide the preliminary results of searching the surface Web and the prominent citeseer archive as one element of the deep Web for publicly available semantic Web service descriptions written in OWL-S, WSML, WSDL-S or SAWSDL by means of the specialized meta-search engine Sousuo 1.4.

1 Introduction

The commun user of the Web might ask: Where are all the semantic Web services in the Web today? According to a focussed search conducted from January 1, 2007 to July 25, 2007 with a specialized meta-search engine Sousuo 1.4, the number of publicly acessible semantic Web service descriptions appears tiny compared to both the number of Web services and even the small fraction of the semantic Web indexed by Swoogle.

Of course, one can argue that this comes at no real surprise for two reasons. First, semantic Web service technology with a standard announced just recently, that is SAWSDL, is immature which provides insufficient common ground supporting its exploitation by end users. Independent from this, one could have expected the massive research and development of the field around the globe in the past half dozen years to have produced a considerable amount of even publicly visible semantic Web service descriptions beyond internal repositories.

Second, one might argue that it is not clear whether the surface Web and academic publications are the right place to look for semantic Web services, as many of them would be intended for internal or inter-enterprise use but not visible for the public. Though this is one possible reason of the low numbers reported above, there was no experimental evidence in favor of, or against this claim.

This motivated us to conduct our initial search experiment: How many semantic Web service descriptions are actually accessible to everyone searching the Web for them? How many of them are written in the standard SAWSDL, and the non-standard OWL-S or WSML? What is the distribution of their geographic locations and application domains? How many of these service descriptions are valid, and are grounded in standard Web service technology for their principled execution in practice? Finally, how many links to semantic Web service descriptions can only be found in academic publications such as those in the prominent scientific archive citeseer as one element of the deep Web?

In this paper, we provide preliminary answers to these questions based on our limited search experiments. In particular, we restricted our first search experiments on semantic Web service descriptions independent from whether they are grounded in actually deployed WSDL services, or not. This is part of future work but should not distract from the original questions above.

The meta-search engine Sousuo with which the search has been performed together with its testing environment are described in section 2. In section 3, we provide the performance of both the search engine and its topic crawler followed by the preliminary results of our experiment in section 4, while section 5 concludes this paper.

2 Meta-search engine Sousuo

The purpose of the specialized meta-search engine Sousuo for semantic Web services is to search the surface Web and the scientific archive citeseer for semantic Web service descriptions in OWL-S, WSML, WSDL-S and SAWSDL.

2.1 Architecture

The general architecture of Sousuo is shown in figure 1.

Overview. Users may select any combination of the following search methods of Sousuo to search for semantic Web services.

– Meta-search (MS) through most prominent search engines Google with A9,
– Sousuo's own focussed topic crawler (TC) based on the WebSphinx crawler,
– Inverse ontology based search (IOS) via Swoogle, and
– Full text scientific archive search (FTAS).

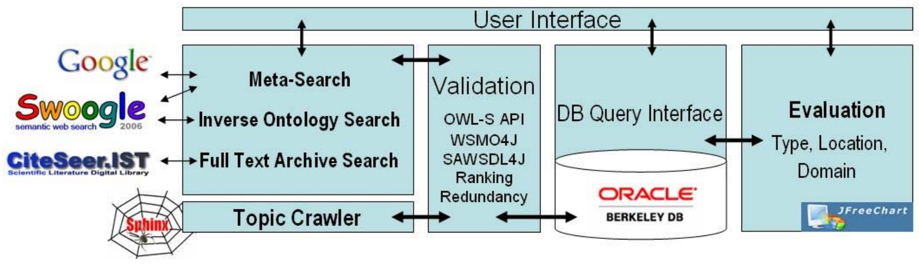


Fig. 1. General architecture of Sousuo.

Sousuo considers returned links to syntactically valid service descriptions relevant. For validation, Sousuo uses the validators of publicly available OWL-S

API, WSMO4J API, and SAWSDL4J API. Depending on the validation result, it determines the relevance ranking score of each link, that is a score of 1.0 if the service description is syntactically valid, 0.5 if validation failed due to minor syntax errors, and 0 else. Validation is complemented by checking whether the link has been already stored in the local database (Open Berkeley XML database[4]).

This database can be queried and evaluated by the user according to the distribution of geographic locations, description formats, domains and categories of semantic Web services, as well as the coverage of the total result returned for any combination of the different search methods. Sousuo 1.4 also informs about the actual performance of both its topic crawler and the whole meta-search engine. Sousuo has been implemented in Java and is publicly available through the software portal semwebcentral.org dedicated to semantic Web software.

Meta-search and topic crawler. The meta-search of Sousuo is restricted to querying Google, Swoogle, and A9 through their API with predefined and user given search keys. Predefined search keys focus on links to files of type, for example, filetype:{wsdl sawsdl, wsml wsml, owls service, owls profile, owl jp, owl kr, owl tw, owl cn, owl service Profile, owl profile, wsdl annotation }. This is complemented by a simple focussed topic crawler which performs a recursive depth-first search taking the continuously growing set of (initially given) links in the local database as root and base set, and terminates with a time-out or given maximum of search depth reached. Validation, ranking, and redundancy checking of found links are as described above.

Inverse ontology based search. This search method is looking for services that reuse ontologies imported by services that have been already located by Sousuo. For this purpose, Sousuo first checks for each link to a service stored in the database the assigned set of individual ontologies required to understand its semantics. It then searches for service descriptions that contain one or multiple of these selected ontology links through respective queries to Swoogle, A9 and Google. The intuition behind this inverse ontology based search is that semantic Web services share ontologies but may not be fully indexed.

Full text archive search. Sousuo queries the scientific archive citeseer via its API for references to papers on semantic Web services, retrieves the respective documents in the answer set formatted in pdf, and scans each of them for embedded relevant links to SWS descriptions. These links then get validated, ranked, checked for redundancy, and stored in the local database by Sousuo as mentioned above.

2.2 Testing environment

Search period and hardware. We ran our search experiment from January 1, 2007, to July 25, 2007, by executing Sousuo once every two weeks over 24 hours on a notebook with CPU Intel Core 2 Duo 2.0GHZ, 2GB RAM, and LAN 10 Mbit/s access to the Internet.

Search space and index. Sousuo does not search any surface Web directory such as yahoo. In fact, respective experiments showed that their answer sets were covered by the one of Sousuo. However, we are currently working to enlarge the search space of Sousuo by incorporating search engines claiming to provide access to parts of the deep Web such as clusty, intute, and infomine with an open, non-commercial API for inquiries.

Hence, the search space of Sousuo is the union of the indices of queried search engines, the discovered realm of the focussed topic crawler plus the index of the scientific archive citeseer. This size is, in general, impossible to determine accurately due to the privacy of information on size, redundancy, and overlapping (coverage). However, the size of the index of Google is estimated with 11.3 billion links [1], while the size of the Swoogle index and citeseer archive is estimated with 1.7 million [2], and 767,558 links, respectively. Besides, our focussed topic crawler explored 11.2 million links in total during the experiment. With an admittedly speculative 90% of an overlap of the latter indices with the Google index, the search space of Sousuo might be estimated with 13 billion pages. The current size of Sousuo's index equals that of its total answer set for the whole search experiment which amounts to 1439 non-redundant, validated links stored in the local database.

3 Performance

As the real relevance set of semantic Web services in the Web is unknown, and impossible to deduce from neither the set of crawled pages nor the answer sets of particular sources queried, we approximate the performance measures of precision and recall for the topic crawler by means of target recall and target precision as defined in figure 2 according to [5].

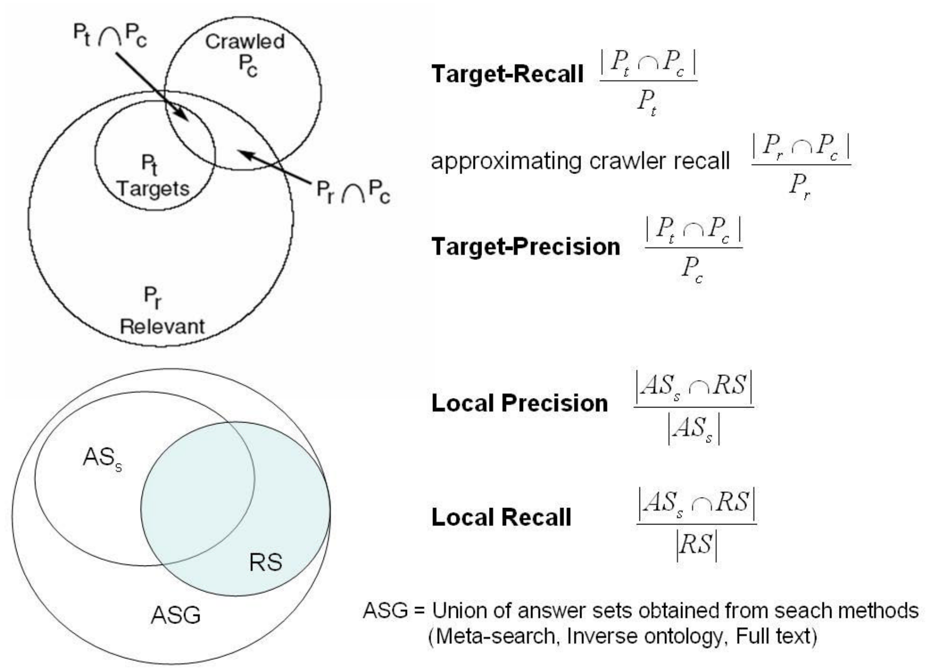
3.1 Performance of focussed topic crawler

Figure 3 shows the average target precision and recall of the focussed topic crawler of Sousuo. It explored around 11.2 million links in total with fairly reasonable throughput of 46 links per minute during the experiment. Regarding its focussed search the target precision is comparably fair enough as well [5].

3.2 Performance of Sousuo

For measuring the precision of the search enginge Sousuo, we determine the classical ratio between the size of the intersection of its answer set AS with the relevance set RS and the size of AS. The answer set of Sousuo equals the set of valid links taken from those returned by its topic crawler, the search using Google and A9 API which answer set is limited to 1k, respectively, 10k links per day,

[1] at http://www.linksandlaw.de/news234-indexgroesse.htm
[2] http://swoogle.umbc.edu/index.php?option=com-swoogle-stats



$P_t \cap P_c$
Crawled P_c
P_t Targets
$P_r \cap P_c$
P_r Relevant
Target-Recall $\frac{|P_t \cap P_c|}{P_t}$
approximating crawler recall $\frac{|P_r \cap P_c|}{P_r}$
Target-Precision $\frac{|P_t \cap P_c|}{P_c}$
AS_s
RS
ASG
Local Precision $\frac{|AS_s \cap RS|}{|AS_s|}$
Local Recall $\frac{|AS_s \cap RS|}{|RS|}$
ASG = Union of answer sets obtained from seach methods (Meta-search, Inverse ontology, Full text)

Fig. 2. Target precision and recall.

the inverse ontology based search via Swoogle, and the full text search through citeseer. The relevance set, however, is restricted to the subset of (manually determined) relevant links of the total union of answer sets produced during the search period.

Figure 4 displays the fairly high average local precision and recall of Sousuo while its target precision and recall is shown in figure 5.

4 Experimental results

The total number of semantic Web services in OWL-S, WSML, WSDL-S and SWASDL including the test collections amounts to 1439 of which only about four percent (65 services) are available outside these collections in the surface Web and through citeseer. Figure 6 shows the number of relevant links found by each of the individual search methods of Sousuo without redundancy checking and test collections OWLS-TC2 [2] and SWS-TC [3].

	Meta-Search	Citeseer	Topic Crawler	Inverse
Meta-Search	35	4	6	10
Citeseer	4	11	4	2
Topic Crawler	6	4	29	8
Inverse	10	2	8	20

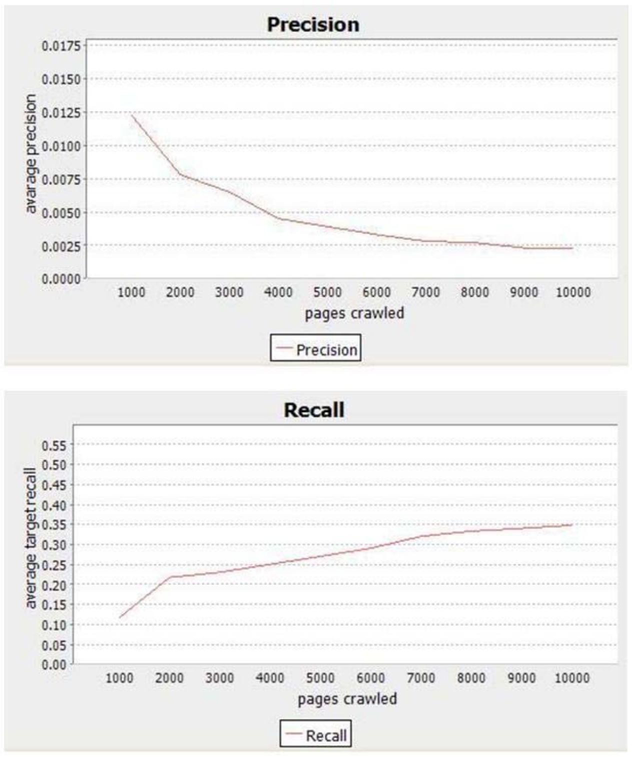


Fig. 3. Average target precision and recall of Sousuo's focussed topic crawler.

The overlapping of answer sets from individual search methods of Sousuo for those services not included in the above mentioned test collections is summarized in table 1. It shows, for example, that the full text archive search returned valid links to semantic Web services in the archive citeseer that were not returned by Google and A9, and a few that have not been returned by any other method. Despite its functional simplicity, the same result holds with our focussed topic crawler. This is in contrast to the inverse ontology based search which answer set is completely covered by those of the other search methods.

4.1 Distribution of semantic Web service formats

Figure 7 shows the distribution of semantic Web service descriptions in prominent formats, that are OWL-S, WSML, WSDL-S and SAWSDL.

Given the historic evolution of the field, and its reasonably fair software support, it comes at no surprise that the quantities of semantic Web service descriptions in OWL-S outnumber the considered alternatives. Remarkably, there are less public WSML services than for WSDL-S and SAWSDL together. Though SAWSDL became a proposed recommendation by the W3C just recently, its

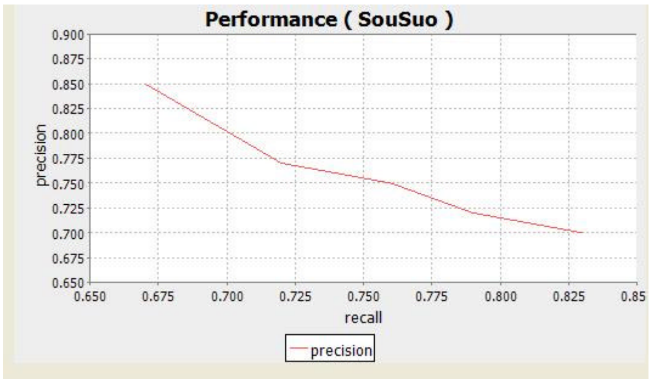


Fig. 4. Average local precision and recall of Sousuo.

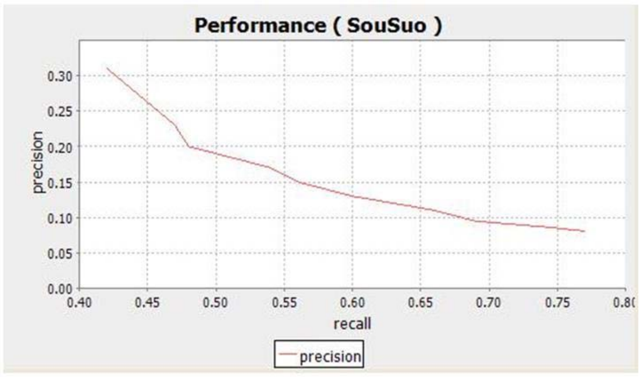


Fig. 5. Average target precision and recall of Sousuo.

software support world wide still appears comparatively negligible which might rapidly change in near term. Apart from two rather medium sized SWS retrieval test collections for OWL-S, that are the OWLS-TC2 [2] and the SWS-TC[3], we did not find any other collection in the Web.

4.2 Geographic distribution

Figures 8 and 9 provide an overview of the geographic distribution and locations of the publicly accessible semantic Web services. Regarding the history of the semantic Web vision, in particular the early joint start between researchers in the US and the EU on DAML+OIL, OWL and OWL-S, as well as the massive funding of WSMO related projects in this field by the European Commission, it might not come at a surprise that, according to the quantities reported, the domain appears clearly dominated by the US and Europe while being remarkably close to each other.

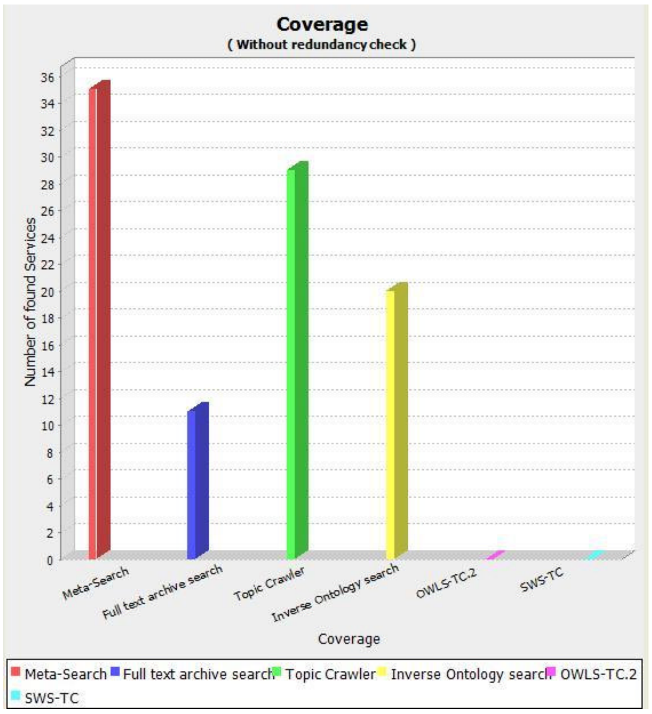


Fig. 6. Number of relevant links found by individual search methods of Sousuo.

However, what surprised us most is that, though major projects in the area exist, we could not find any valid semantic Web service description published in the rest of the world publicly, in particular the Asia and Pacific rim. Additional personal communication with few selected research groups at universities in these regions revealed that, if semantic Web service descriptions do exist at their site, the public retrieval from specific project related repositories is prohibited, hence invisible to any search engine. In general, we appreciate any reference by the interested reader to publicly accessible semantic Web services in one of the considered formats, in particular if published in the above mentioned geographic regions.

4.3 Internet domains and business categories

Figures 10 and 11 show the distribution of the domain and business category of found services outside the test collections. Business and travel are the most common categories, followed by finance and education.

In compliance with the prevalent geographic distribution and location of semantic Web service links in the US and the EU, the majority of links from Internet domains devoted to commerce (.com, 8 %), organisational (.org, 28%) and educational institutions (.edu, 15%) is hosted in these world regions. Remarkably,

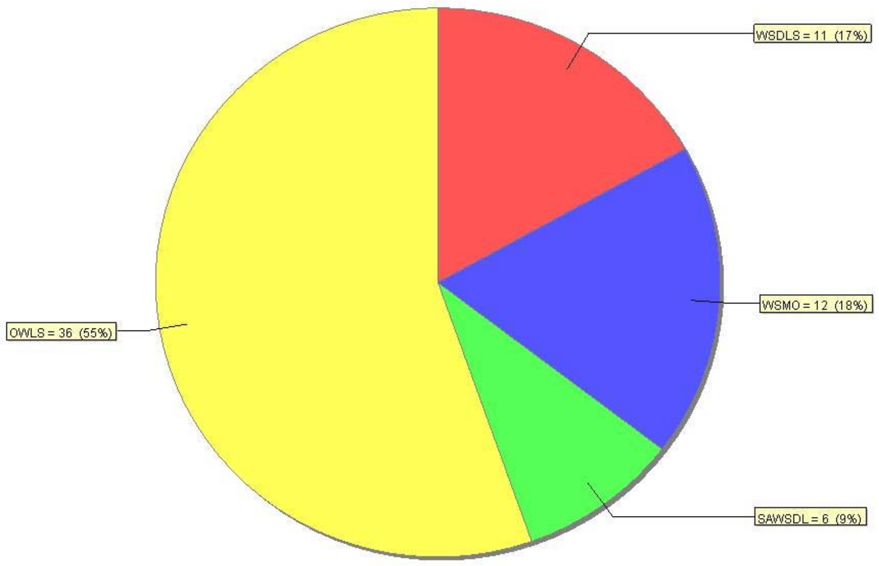


Fig. 7. Distribution of prominent semantic Web service formats.

most of these semantic Web services publicly accessible in the EU are located in the UK.

The most common business domain for semantic Web services according to their naming or statement in the profiles are business (16%) and travel (17%), followed by education (11%), finance (11%), and government (6%). One third of found semantic Web service links, however, belongs to a variety of other domains of smaller size such as sports and health.

5 Conclusions

The preliminary results of our experimental searching for semantic Web services in the surface Web by use of a specialized meta-search engine is rather desillusioning. We found not more than around 1500 indexed semantic service descriptions in OWL-S, WSML, WSDL-S or SAWSDL in the Web, of which only about four percent are located outside special test collections like the OWLS-TC2. This quantity appears tiny compared with the sheer volumes of estimated thirty billion and one million indexed resources in the Web, respectively, semantic Web encoded in RDF and OWL.

As mentioned above, the reported preliminary experimental result does not reflect the strong research efforts carried out in the SWS domain world wide in the past few years, independent from the status of maturity of SWS technology and implied low adoption by end users yet. Nevertheless, the result might encourage the community as a whole to increase its visibility and awareness to

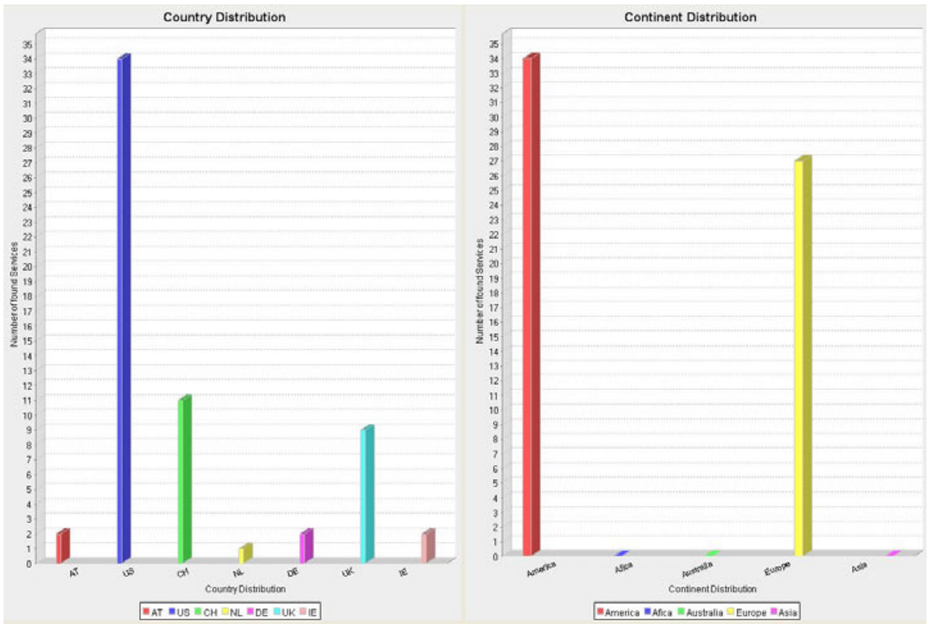


Fig. 8. Geographic distribution of semantic Web services.

the common Web user outside the community and savvy project teams also by publishing a significant number of SWS show cases in the surface Web.

Although one could have expected these results, in particular the majority of semantic Web services being published in protected internal project repositories and other sites of the deep Web[6], there was no experimental evidence available in favor of this claim or against. On the other hand, there still is plenty of space left to search both the surface and the deep Web: The search space of Sousuo in its current version is limited to only few selected indices of freely accessible and prominent search engines with open API, that are Google, A9, Swoogle, and the scientific archive citeseer.

However, raising awareness by a signifcant number of show cases is senseless if not complemented by efforts to equip users with easy to use software support for building, sharing and reusing semantic Web services. Unfortunately, this is missing either, despite the variety of SWS related software available at relevant open source software portals such as semwebcentral.org and sourceforge.net. Though, a first standard for SWS description has been announced just recently such that many of these tools, though pioneering, became more or less historic now.

Sousuo is available at http://projects.semwebcentral.org/projects/sousuo/ Ongoing work includes the improvement of searching for publicly available semantic Web services by Sousuo (v2) and the comparative analysis of the yet unknown set of publicly available Web services in WSDL.

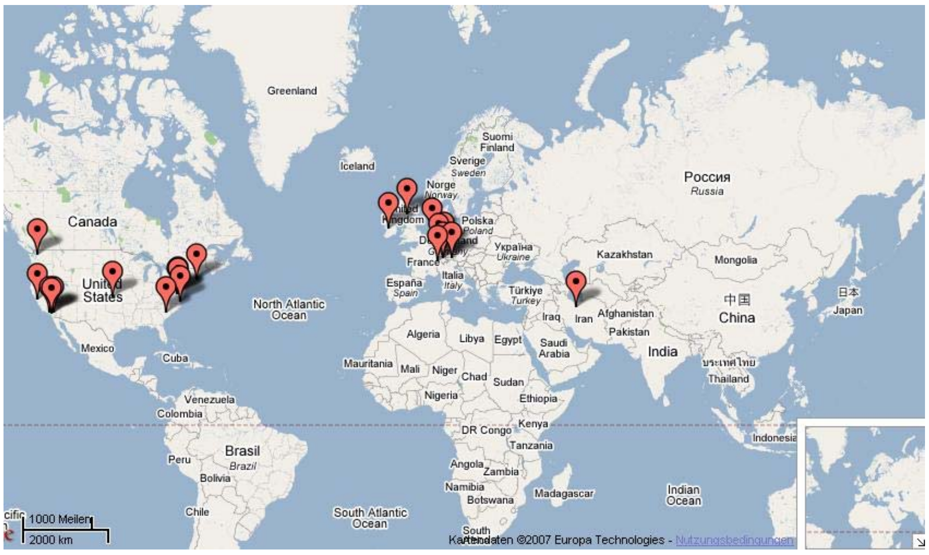


Fig. 9. Geographic locations of semantic Web service providers.

References

1. L. Ding, T. Finin, A. Joshi, R.S. Cost, J. Sachs: Swoogle: A Semantic Web search engine and metadata engine. Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management, 2004.
2. OWLS-TC2: http://projects.semwebcentral.org/projects/owls-tc/.
3. SWS-TC: http://projects.semwebcentral.org/projects/sws-tc
4. Oracle Open Berkeley XML Database: http://www.oracle.com/database/berkeley-db/xml/index.html
5. G. Pant, P. Srinivasan, F. Menczer: Crawling the Web. In M. Levene and A. Poulovassilis, eds.: Web Dynamics, Springer, 2004.
6. C. Sherman, G. Price: The Invisible Web: Uncovering Information Sources Search Engines Can't See. Cyberage Books, 2001. The Deep Web: http://en.wikipedia.org/wiki/Deep-web.
7. SouSuo 1.3: http://projects.semwebcentral.org/projects/sousuo/.
8. Swoogle: http://swoogle.umbc.edu/

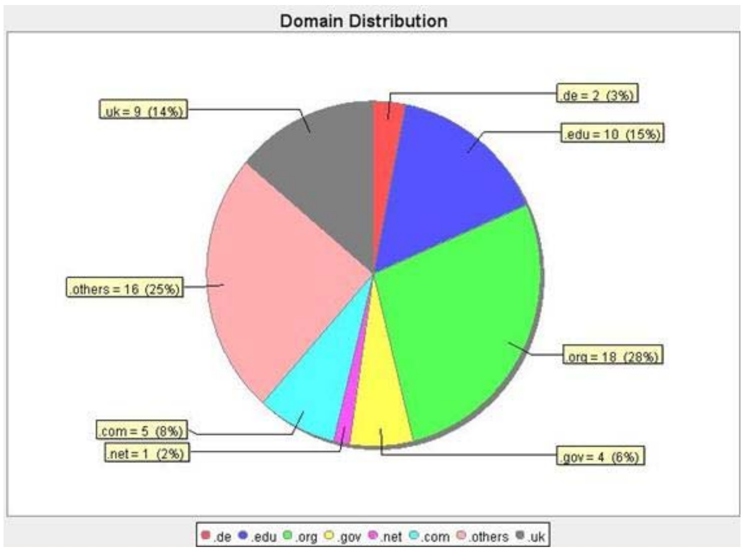


Fig. 10. Internet domains of semantic Web services.

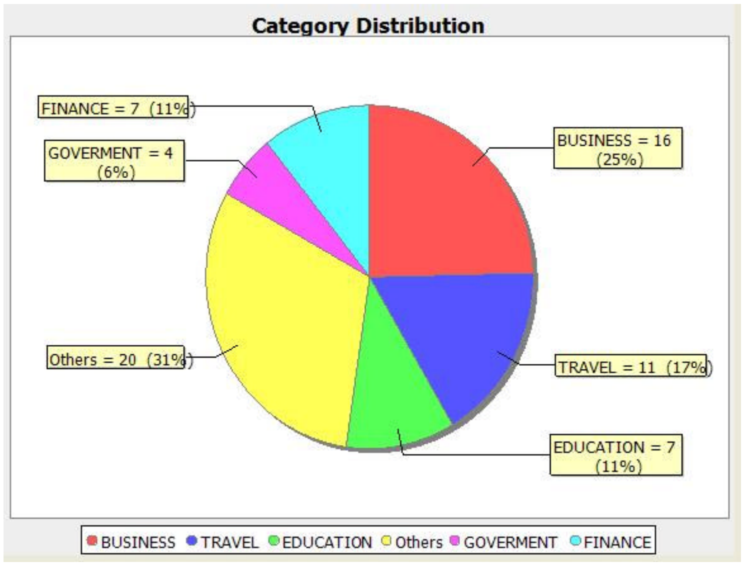


Fig. 11. Business categories of semantic Web services.

Towards Semantic Web Service Engineering

Gennady Agre[1], Zlatina Marinova[2], Tomas Pariente[3] and Andras Micsik[4]

[1] Institute of Information Technologies – Bulgarian Academy of Sciences
gennady.agre@abv.bg
[2] OntoText Lab. Sirma Group Corp, Sofia, Bulgaria
zlatina.marinova@sirma.bg
[3] ATOS Origin SAE, Spain
tomas.parientelobo@atosorigin.com
[4] MTA SZTAKI - Hungarian Academy of Sciences
micsik@sztaki.hu

Abstract. The paper presents the main results of the IST FP6 INFRAWEBS project. The project has developed an easy and effective way of constructing and using semantic descriptions for existing and new Web services. INFRAWEBS has adopted the WSMO (Web Service Modeling Ontology) and WSML (Web Service Modeling Language) specifications and has imposed no additional requirements to them. Therefore, the advanced software components developed during the project are of interest to the whole WSMO community. INFRAWEBS offers a SOA framework – INFRAWEBS Integrated Framework (IIF), based on the ESB integration paradigm, which can be easily used by different groups of users (application providers, designers of SWS, etc.). The IIF enables integration of components of different technologies. Furthermore, it can be considered as one of the first frameworks for semantic service engineering that covers the whole SWS life-cycle and allows for creation of complex, semantically-enabled applications.

Keywords: Semantic Web services, Semantic Web Service engineering, Service discovery, Service development, Service composition, WSMO, SOA.

1. Introduction

Semantic Web services (SWS) research is related to automating the development of Web service based applications through semantic Web technology. By providing formal descriptions with well defined semantics, SWS are another step in the direction of solving service engineering problems such as service inter-operation, discovery, choreography and orchestration. There are two major initiatives that work on developing a world-wide standard for the semantic description of Web services: OWL-S [17] and the Web Service Modeling Ontology (WSMO) [20]. WSMO is still under development and has been adopted over the past three years in several Integrated IST Projects such as DIP [9], SEKT [24], Knowledge Web [11] and ASG [5], by consortia including in total more than 50 academic and industrial partners.

At the moment the practical application of SWS technologies is still rather restricted due to several reasons, some of which are identified in [25]:

- The high complexity of both OWL-S and WSMO,
- The lack of standard domain ontologies and unavailability of mature tools supporting WSMO or OWL-S,
- The absence of pilot applications focusing on every-day needs of consumers, citizens, industry etc., which can demonstrate the benefits of using semantics.

The IST research project INFRAWEBS[1], successfully completed in the beginning of 2007, proposes a technology-oriented step for overcoming some of the above-mentioned problems. It focused on developing a Semantic Service Engineering Framework enabling creation, maintenance and execution of WSMO-based SWS, and on supporting semantic Web service applications within their life-cycle. Being strongly conformant to the current specification of various elements of WSMO (ontologies, goals and semantic Web services), the INFRAWEBS Framework *hides* the complexity of creation of such elements by:

- Identifying different types of actors (users) of SWS Engineering Technologies;
- Clarifying different phases of the Semantic Service Engineering process, and

[1] http://www.infrawebs.eu

- Developing a specialised software toolset, oriented to the identified user types and intended for usage in all phases of the SWS Engineering process.

In the rest of this paper the main features of the INFRAWEBS Framework are presented in more details. Section 2 gives a user-oriented overview of the framework, by identifying the types of INFRAWEBS users, describing the Framework architecture and discussing the support provided by the Framework for its users. Section 3 presents a SOA implementation of the INFRAWEBS Framework. Section 4 presents evaluation results of INFRAWEBS based on the two pilot applications. The last section is a conclusion.

2. INFRAWEBS Framework

Conceptually, the INFRAWEBS Framework is a Service-Oriented Architecture (SOA) comprised of coupled and linked INFRAWEBS semantic Web units (SWU). Each unit provides tools and components (realized as Web services) for analyzing, designing and maintaining WSMO based semantic Web services and SWS applications within the whole life-cycle. A very important aspect, concerning the development and operation of any SOA-based application, is to identify the actors and their roles in the scope of the application [8]. The following actors have been identified as potential users of the INFRAWEBS SWS Engineering Framework:

- *Semantic Web Service Provider* – any provider of already existing Web services, who would like to convert them to Semantic Web services and to publish them.
- *Semantic Web Service Broker (Aggregator)* – a provider, who would like to create and publish a service achieving its desired functionality via composition of several existing Semantic Web services.
- *Semantic Web Service Application Provider* – an organization, that would like to design its own application based on Semantic Web Service Technology.
- *Web Service Application Consumer* – an “ordinary” end-user of a Web Service Application, who transparently uses the INFRAWEBS Framework (while using the Application) for finding and executing a Web service or a composition of Web services able to satisfy his/her request (goal).

The developed categorization of INFRAWEBS Framework users allowed us to identify more clearly the set of different tasks that the Framework is able to accomplish, in order to satisfy the objectives of these users, as well as the set of necessary components. The INFRAWEBS Framework is developed to support all stages of the semantic Web service life-cycle (see Fig.1.):

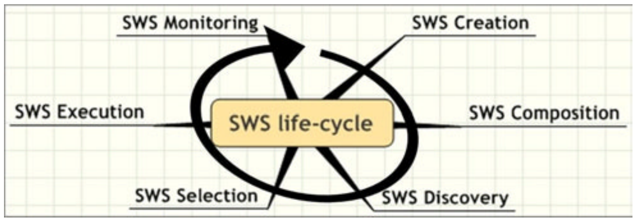


Fig. 1. INFRAWEBS SWS life-cycle

- *SWS Creation* – combines activities related to the creation of semantic descriptions of the Web services, as well as creation of necessary ontologies and goals. Created descriptions are then persistently stored and can be published for common usage.
- *SWS Composition* – already created semantic services can be combined (by their provider, or an external aggregator) to provide new, value-added services. Composed services are also represented and stored as semantic descriptions and can be further used just like other services.
- *SWS Discovery* – enables already described services to be discovered for usage. Discovery is a semantically-enabled process of matching user requests (specified as WSMO goals) to service functionality (represented as the WSMO service capability).
- *SWS Selection* – allows users (be they humans or other services) to choose in a pro-active manner which of the discovered services to be executed. Generally, selection can also be done automatically, for example by the discovery agent (tool), but INFRAWEBS supports also scenarios requiring user selection or domain-specific automated selection provided by semantically-enabled applications, and for this reason a dedicated step in the SWS life-cycle is necessary.
- *SWS Execution* – deals with the actual Web service delivery, when the user provides some input to the service and invokes it to obtain expected results.

- *SWS Monitoring* – an important phase when information about executed services is gathered in order to be further used for service selection.

The INFRAWEBS framework clusters these stages into two phases: design time and runtime (Fig. 2 and Fig. 3). The components involved in the design time phase are the following:

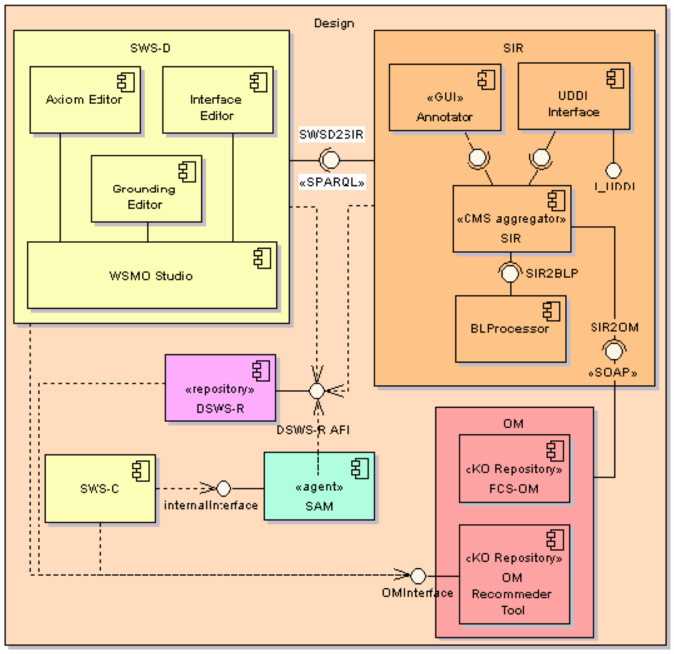


Fig. 2. Design Time Architecture of the INFRAWEBS Framework

- SWS-D (Semantic Web Service Designer) - responsible for the creation of WSMO-based semantic descriptions of Web services and goals
- SWS-C (Semantic Web Service Composer) - responsible for the composition of existing WSMO-based semantic Web services
- DSWS-R (Distributed Semantic Web Service Repository) - responsible for the persistent storage of WSMO-based descriptions and their publication within the Framework.
- SIR (Semantic Information Router) – responsible for registration and annotation of Web services (their WSDL definitions), which are then used in the process of SWS design.
- OM (Organizational Memory) – responsible for indexing and case-based retrieval of WSMO-based descriptions of Web services, goals and ontologies.

The components involved in the Runtime phase are the following:

- SAM (Service Access Middleware) - used by semantic service applications as a middleware supporting the steps of semantic Web service usage, including service discovery, selection and execution.
- SWS-E (Semantic Web Service Executor) - responsible for executing semantic Web services.
- QoS-Monitor (Quality of Service Monitor) - collecting monitored data and calculating quality of service statistics for the semantic Web services being executed.
- DSWS-R – providing run-time access to semantic descriptions of Web services, goals and ontologies.
- OM - used in the first step of the discovery process to retrieve an initial set of semantic Web services matching the current goal, based on ontological keywords similarity.

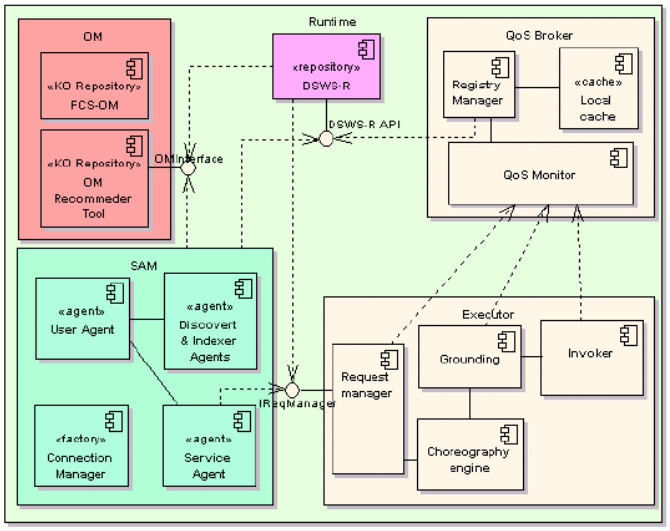


Fig. 3. Run-Rime Time Architecture of the INFRAWEBS Framework

The INFRAWEBS architecture reflects a novel approach to solving problems, occurring in the process of creation of SWS applications, through tight integration of similarity-based and logic-based reasoning. The similarity-based reasoning is used to find fast an approximate solution, which is further clarified by the logic-based reasoning. The following sections show how the architecture described supports different kinds of INFRAWEBS Framework users.

2.1 Use of the INFRAWEBS Framework by Web Service Providers

In INFRAWEBS a semantic Web service can be created in two ways – by converting an existing non-semantic Web service into a semantic one (this process is called "SWS design") or by composition of several existing SWS. The life-cycle of the SWS design process is defined by both the static WSMO-based structure of the semantic description, and by certain assumptions on what kind of additional semantic information is needed, where it is stored and how it can be found. In general, non-semantic Web services are implemented and formally described (as WSDL definitions) outside the INFRAWEBS Framework and are deployed in some Web service container (supported by the service provider).

In order to facilitate finding of such Web services, the service provider can annotate them with natural language based metadata (for example, according to the Dublin Core[2] schema). The addition of such metadata does not convert a Web service into a WSMO-based SWS - it simply enables the process of finding the annotated service by users or business partners. In the INFRAWEBS Framework this service annotation activity is considered as the first, preliminary step of the SWS design process and it is seen as a process of registration of Web services into the Framework. The activity is supported by a dedicated component – the Semantic Information Router (SIR), which is a metadata-based content management and aggregation platform for storing and annotating Web services [21]. SIR provides a Web interface for registration, annotation and categorization of WSDL and BPEL files to the INFRAWEBS system.

The SIR registration interface can be considered as the entry point to the overall INFRAWEBS system, since the phase of SWS creation typically starts by finding the appropriate WSDL file. This interface has been developed to support the roles of:

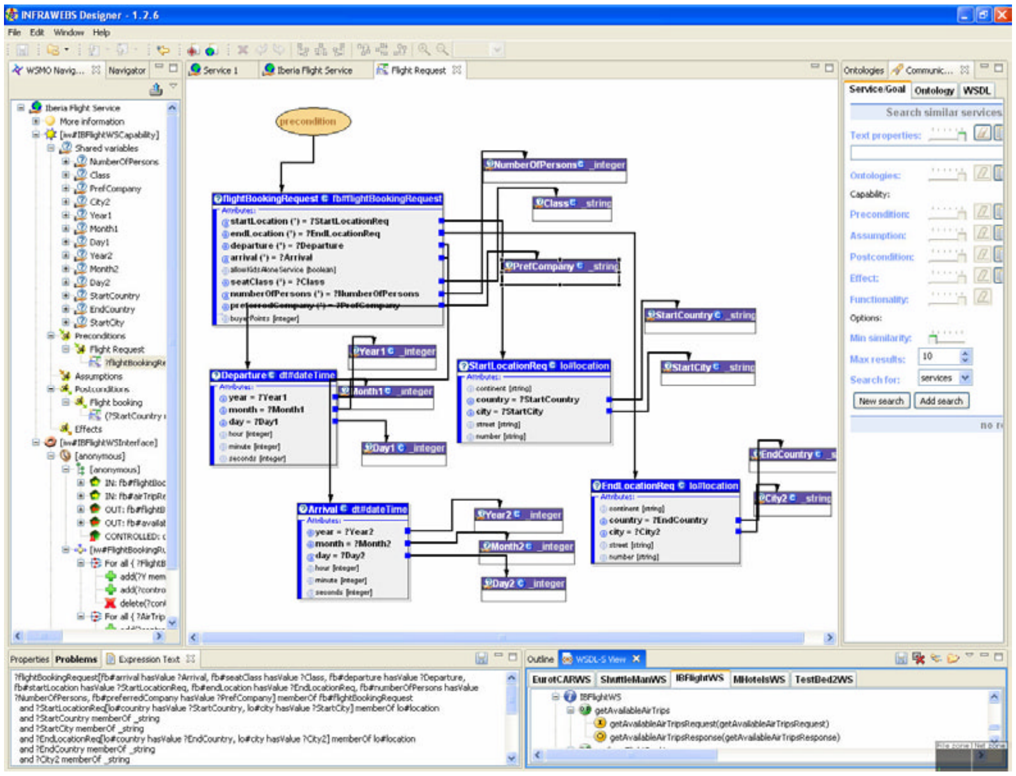
- Service providers (developers) who can register, annotate and register WSDL and BPEL files
- Administrators who can manage registered services and metadata schemas used to annotate the services by the developers.
- Semantic service engineers (at the SWS provider organization) who query the SIR in order to find appropriate WSDL definitions of the services they want to describe semantically.

SIR stores metadata about the registered WSDL and BPEL files natively in the Resource Description Framework (RDF) format. This metadata is exposed for querying via the Simple Protocol And RDF Query Language (SPARQL).

[2] http://dublincore.org/

The next step of the SWS design process is the creation of its WSML description conforming to the WSMO specification. In order to facilitate this very complex activity it is split into several sub-steps:

- Finding a desired non-semantic Web service (described by an annotated WSDL file).
- Finding a set of appropriate ontologies to be used for semantic re-formulation of the main Web service functionality in ontological terms. In the INFRAWEBS Framework ontologies are stored in the WSMO element repositories (DSWS-R).
- Creation of semantic description of the service behavior. According to WSMO, that description consists of service grounding, specifying the correspondence between data structures in the semantic and non-semantic descriptions of a WSMO service, and the service choreography describing how it is possible to communicate with the semantic service in order to execute properly the non-semantic Web service grounded to it.
- Semantically describing service functionality, advertising what the service can do. Created semantic service capability descriptions are used for service discovery.
- Publication of service descriptions. In the INFRAWEBS Framework the description of SWS (as well as other WSMO-based semantic objects such as ontologies, goals, etc.) are stored in the local (belonging to concrete local SWU) repositories (DSWS-R). The DSWS-R component is then responsible for propagating the advertisement of published objects within the INFRAWEBS Framework, in accordance with the propagation policy specified as part of the advertisement.



```
?flightBookingRequest[fb#arrival hasValue ?Arrival, fb#seatClass hasValue ?Class, fb#departure hasValue ?Departure,
fb#startLocation hasValue ?StartLocationReq, fb#endLocation hasValue ?EndLocationReq, fb#numberOfPersons hasValue
?NumberOfPersons, fb#preferredCompany hasValue ?PrefCompany] memberOf fb#flightBookingRequest
  and ?StartLocationReq[lo#country hasValue ?StartCountry, lo#city hasValue ?StartCity] memberOf lo#location
  and ?StartCountry memberOf _string
  and ?StartCity memberOf _string
  and ?EndLocationReq[lo#country hasValue ?EndCountry, lo#city hasValue ?City2] memberOf lo#location
  and ?EndCountry memberOf _string
  and ?City2 memberOf _string
```

Fig. 4. Screenshot of the INFRAWEBS Designer showing a graphical model of a service precondition.

All of the above steps (except the last one) are performed by means of a special tool – the INFRAWEBS Designer - a graphical, ontology-based, integrated development environment for designing WSMO-based SWS and goals (Fig. 4), [1]. The INFRAWEBS Designer is oriented to Web service providers, and Web service application providers, and does not require any preliminary knowledge of WSML – the logical language used for describing SWS and goals in WSMO [7]. The most important characteristics of the Designer are:

- *User-friendliness:* it proposes an intuitive graphical way for constricting and editing service and goal descriptions, abstracting away as much as possible from the concrete syntax of the logical language used for describing them. The WSML description of the semantic object under construction is automatically generated from the graphical models created by the user.

- *Intensive use of ontologies:* the process of constructing logical descriptions of semantic objects is ontology-driven - in each step of this process the user may select only these elements of the used ontologies that are consistent with the already constructed part of the description.
- *Reusability:* creation of semantic descriptions is a complex and time-consuming process, which can be facilitated by providing the designer with an opportunity to reuse existing, similar descriptions, created by the designer herself or by other users. The INFRAWEBS Designer provides the user with such an opportunity by applying the case-based reasoning approach.

The main novelty of the INFRAWEBS approach is in the combination between the Designer and Organizational Memory (OM), which plays the role of a case-based memory in the INFRAWEBS Framework, [4]. In order to facilitate the process of creating semantic descriptions, the OM can be consulted to find similar semantic descriptions that can serve as templates for the WSMO element under construction (Fig. 5). OM considers all semantic objects as special text documents – “knowledge objects” having specific structure, while each structured part of a document is written in a specific language (natural language and/or WSML). The structure of a document depends on the object type , and in the case of semantic descriptions consists of the main parts representing the element according to the WSMO specification. By means of a special “filtering” procedure (specific for each object type), the OM creates its internal “case” representation of each object - *{T, P, S}*, where *T* is the object type (service, goal, etc.), *P* is the compressed representation of the object content as a structural object feature vector and *S* is the object identifier (IRI), along with some natural language annotation briefly describing the object. In order to provide an efficient filtering process, the OM is equipped with different types of dictionaries – knowledge-domain keywords, generic keywords, abbreviations, stop words and synonyms. It also maintains a list of keyword recommendations that are automatically extracted from each newly filtered object. The OM applies a fuzzy mapping between ontology-based dictionaries and natural language dictionaries, which is used when retrieving similar semantic descriptions by natural language queries.

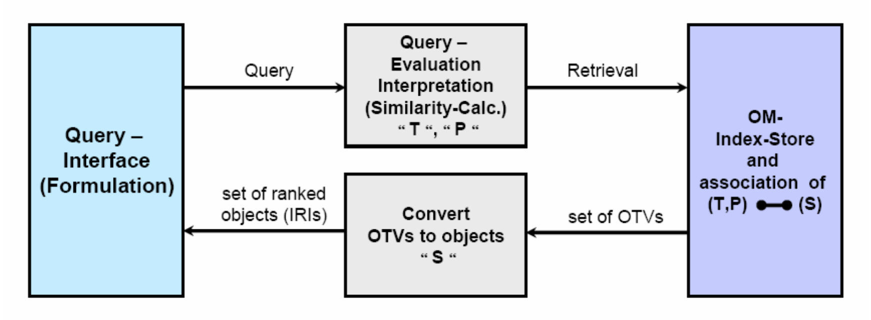

Query – Interface (Formulation)
Query
Query – Evaluation Interpretation (Similarity-Calc.) “ T “, “ P “
Retrieval
OM- Index-Store and association of (T,P) ●—● (S)
set of OTVs
Convert OTVs to objects “ S “
set of ranked objects (IRIs)

Fig. 5. Scheme for using OM for finding similar WSMO objects

In order to enable the efficient retrieval of the knowledge objects, all their case representations are organized in a hierarchical structure – a multi-layer classification tree, whose internal nodes are centroids and the leaves are concrete cases. Each centroid is a “typical” representative of a set of similar cases, while splitting cases to different sets is done by means of a fuzzy concept matching clustering procedure [16]. Each new case is placed into the classification tree, based on its similarity with the tree centroids. The resulting similarity quotient is calculated using a set of similarity functions, different for each structural part of the case, see [2].

In fact, the use of OM as a knowledge base, containing knowledge about available semantic (services, goals, ontologies) and non-semantic (WSDL files) descriptions, makes the otherwise tedious process of designing a WSMO element (requiring expert knowledge of the WSMO model and the WSML language) a task achievable by ordinary Web service developers.

The main objective for creating a semantic service or other semantic objects (goal, ontology etc.) is to allow this object to be discovered and used by other users. This is achieved by publication of an advertisement of the description in remote registries, provided by the DSWS-R component. Each DSWS-R component provides both functionalities of a registry and a repository, [14]. While the Repository component enables storage and management of WSMO descriptions within the scope of the organization (particular SWU), the Registry is responsible for their publication, propagation and access within the Framework (the network of SWUs). The Registry provides functionality that can be used both to specify which SWS descriptions are publicly available for discovery (and for subsequent composition or execution) and to exchange them with the other registries in the p2p network.

The p2p network is dynamically built by the registries of the SWUs joining the INFRAWEBS framework. Each organization that would like to offer its services (ontologies or goals) and use (or reuse) services provided by other organizations has to:

- Describe its own services as SWS and store them in its local Repository (within the DSWS-R).
- Set up its own registry (part of the DSWS-R) and deploy it on a Web server (as a Web service)
- Add all the endpoints of registries of its business partners in its local Registry configuration, this list of partner registries can be extended at any time.
- Publish advertisements of (some of) its own services in the local Registry and set a propagation policy for each advertisement. Then the Registry will automatically propagate each advertisement, according to its propagation policy, to partner Registries.

When a request to load a particular semantic description (e.g. a SWS) is received by the DSWS-R component it checks whether this description is locally stored or it is advertised in the Registry. Then the DSWS-R automatically retrieves the description either from the local Repository or through a direct request to the corresponding remote Registry.

The DSWS-R functionality can be used by Semantic Service Providers who would like to advertise their services within the INFRAWEBS Framework of partners, so that these descriptions are available for usage and also can be reused by other partners when creating similar descriptions.

2.2 Use of the INFRAWEBS Framework by Web Service Aggregators

INFRAWEBS enables Web service aggregators to compose, in design-time, already existing SWS descriptions in order to provide new, value-added services. Composed services are also described as WSMO services, and can then be discovered and used in the same way as atomic services. The INFRAWEBS Design-time Composer tool (SWS-C) enables combining different SWS to obtain a new, complex semantic service, [6]. It uses adapted workflow methodology for creating WSMO-based service compositions, as well as for the determination and visualization of data and control flows within the composed service.

Services to be included in a composition can be selected either directly, by browsing available services, or found through similarity search. In the latter case, a plain-text request is constructed and the Organizational Memory is queried to provide similar services. The result of composition - a WSML description of the service, can be saved locally or persistently stored in the DSWS-R. The graphical schema of the workflow diagram is saved separately as an associated XML file.

Since the WSMO specification of SWS orchestration language is still under development, design-time compositions of services, in INFRAWEBS, are described as choreographies combining the choreography descriptions of participating services. In the future, when the orchestration specification is fully developed, the SWS-C can be extended to create orchestration descriptions on the basis of the defined workflow model.

2.3 Use of the INFRAWEBS Framework by Semantic Service Application Providers

One of the main objectives of the INFRAWEBS Framework is to support organizations that want to create semantically-enriched applications based on semantic Web service technology. In the previous sections the support of the creation of SWS descriptions, via service design or composition, has been explained. The same service creation tools can be used by Semantic Service Application Providers to create the basic functionality of their applications, realised by SWS. However, in order to provide its functionality as services, the application should be able to use all possibilities provided by the runtime INFRAWEBS Environment. At runtime, discovery of a semantic Web service is done via matching a specific user goal description against all service capability descriptions, hence, the first task of the application provider is to prepare a set of goals presenting the application functionality from the user point of view.

According to the WSMO specification [20], each goal is represented as a set of WSML logical expressions and has a structure similar to that of a WSMO service. We consider absolutely unfeasible to assume that either the end-user of a semantic service application will be able to write directly such expressions, or that it is possible to devise an algorithm able to translate automatically each possible natural language query to a corresponding WSML logical expression (goal). That is why, the INFRAWEBS conceptual architecture assumes the presence of a (predefined) set of “general” WSMO-based goals that may be “re-used” or instantiated by the end-users. These general goals (called “goal templates”) should be prepared by the SWS Application provider in design time.

Even though both user goals and goal templates are syntactically represented as WSML goals, the semantic representation of a goal template is slightly different. In order to implement the mechanism for run-time instantiation of goal templates, it is necessary to mark in advance which variables (parameters) of the template are allowed for instantiation (i.e. could be replaced by some concrete values) when formulating a concrete user goal. Such “input” variables are marked in the goal template precondition by means of a special concept with a special attribute (“slots”) identifying the variables (as can be seen on Fig. 6).

The assumption that *the functionality of a service application* can be fully described by a *set of goal templates,* which, on the one hand, express all general requests the user can send to the application, and, on the other hand, provide abstractions of all services that can satisfy user goals in this application, is still rather demanding, since it requires the goal templates to be prepared in advance for *all possible* user goals. Being not able to further reduce this demand we, however, are able *to facilitate* the service provider in satisfying it. In order to do this, we assume that:

- Each goal can be represented either by an *atomic* goal template (describing basic functionality of the application) or by *combination (composition) of some atomic goal templates,* and
- The INFRAWEBS Framework provides to the application designer necessary and easy-to-use tools both for goal template creation and for goal composition.

In order to support SWS application providers in creating atomic and composite goal templates, we have developed a special tool - the *Goal Editor*, which is part of the INFRAWEBS Designer. An atomic goal can be created in a graphical way, similarly to the way semantic services are designed, as well as by re-using the descriptions of already existing goals and services. The process of constructing the description of a composite goal is implemented as creation of a new goal, whose capability definition is built by copying the corresponding axioms from several “sub-goals” of the composite goal under construction. Such “copy-paste” way of constructing the composite goal is extended by an additional operation marking “the source” from which the original axiom has been taken (see [3] for further details).

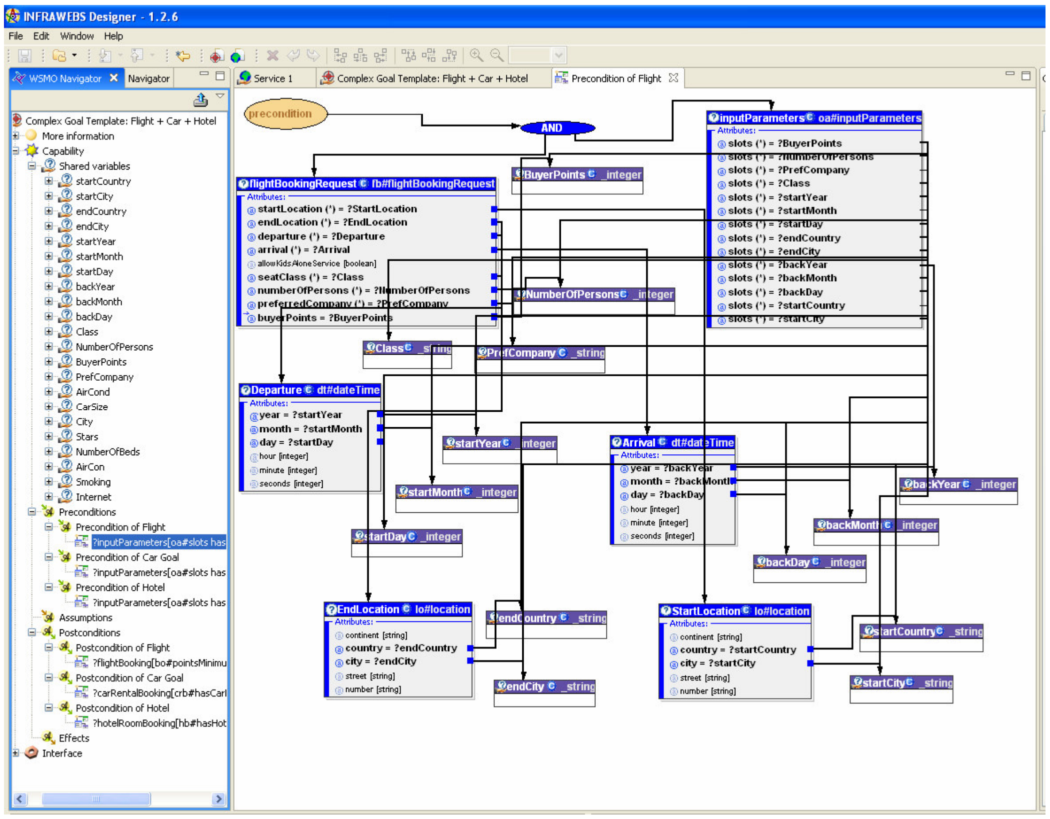


Fig. 6. Screenshot of the INFRAWEBS Designer showing a graphical model of a goal precondition.

The development of two pilot SWS applications (see Section 4) has shown that the integration of the INFRAWEBS Designer with facilities for creating WSMO goals has made it a powerful instrument for creating atomic and composite goal templates and significantly simplifies the work of the service application providers.

2.4 Running Semantic Service Applications

The functionality of a semantic service application is provided by the semantic Web services used by that application. In order to run such applications, the following functionality is provided by the INFRAWEBS Run-time Environment:

- ***Refinement of goal templates:*** as mentioned in the previous section, semantic service application functionality can be expressed as a set of generic goals (called "goal templates"), describing in terms of WSMO all general requests a user can pose to the application. From this point of view, each particular user request is seen as a concrete instantiation of the appropriate goal template. Thus, creation of a concrete user goal is done by refining a goal template through replacement of its free variables with user-supplied data.
- ***Service discovery:*** the concrete user goal is used for discovering a set of existing semantic services able to satisfy the goal. INFRAWEBS discovery is implemented as a three step process consisting of a pre-filtering step, a step for logical matching and result preparation. The aim of the pre-filtering step is to narrow the list of candidates by using traditional text-processing (keyword matching) algorithms. This step is implemented by sending a request to the OM component to find the most similar semantic services (at the level of ontology keywords representation) to the current goal. It is obvious that the result of such keyword-based discovery may contain non-matches from logical point of view. Nevertheless, the key constraint is to not filter out any good semantic matches in this phase, since logically-incorrect matches will be filtered out by the second step of discovery. During the second, logical step of discovery, the postconditions and effects of the goal and the services are matched to see whether each service can fulfill the request. In addition, the goal preconditions and assumptions are matched against service preconditions and assumptions. To compute semantic matching the INFRAWEBS discovery engine applies the unification facility of Prolog engines. Instead of matching large logical expressions, they are broken into small pieces using Disjunctive Normal Form, where the matching of each piece (clause) can be judged individually. The matching algorithm starts by unification of clauses and variables in both sets representing goal and service. The algorithm iterates over all clauses and labels them as matched, ignored or failed according to the specified unification rules (see [11] for details). The result of discovery is a list of matching services, ranked according to the available simple metrics: number of matching expressions, number of clauses matched in the goal, etc. In order to assist the user in service selection, additional information is provided for each service including service metadata (some of service non-functional properties) and available quality of service data (availability, etc.).
- ***Service selection:*** service selection is done from the list of matching services. In principle it can be done automatically by the application or by the user, on the basis of the annotation information attached to each service in the list. In the INFRAWEBS test bed applications selection is done by the service application user.
- ***Service execution:*** the main objective for running a SWS application is to execute the selected services and present their results to the application user. Generally, service execution is an interactive process carried out between the service requestor and the service provider, based on the defined SWS choreography. In our case the service requestor role is taken by the application which serves as a proxy to the actual user. Thus, the user has no notion of the service choreography, but simply provides (if willing to) data needed for the service to fulfill its task. In the INFRAWEBS Framework service execution is considered as an iterative process of: service invocation that initiates the execution and gets all input data; execution of the selected SWS choreography and providing additional input data if such is requested for the service delivery, see [15].
  - *Service invocation* is done by SAM (Service Access Middleware) based on the identifier of the selected SWS and the description of the user goal, which should be satisfied by the service execution. The description of the goal is automatically transformed into an "input ontology", which is used by the SWS as a source of input data. With this approach, the semantic application needs no knowledge and no special source code for ontology creation, which significantly simplifies the process of application development.
  - *Semantic service execution* is carried out by the SWS-Executor component of the INFRAWEBS Framework, [22], which retrieves the service description and feeds its choreography to the Choreography engine. WSMO choreographies are represented as Abstract State Machines consisting of states and state transitions, [22]. States are defined by the choreography state signature, while transitions are defined through rules. Each transition actually changes the current

state by means of adding, deleting or updating certain facts (instances) as specified by the corresponding rule. During state transitions, the related Web services are executed as defined in the specific SWS grounding. The Choreography engine first initiates the state with the data from the input ontology, and then enables transitions between states by firing the corresponding transition rules. When the current state contains insufficient data and no state transitions can take place, the additional user input may be necessary to continue the execution. Necessary input is specified in the output ontology returned to SAM by the Executor. Execution ends when the Executor sends the final output ontology and does not expect more user input. In case of errors, their types are described in the context object returned to SAM, which transfers it to the application,

- When *additional input data* is needed from the user, SAM sends to the application the current output ontology and the type of the variables that have no value. Instances in the output ontology are converted into the corresponding object structures that can be processed as plain Java objects. In that way, the application can easily find the empty attributes of the missing concept and fill them with relevant values. Such an approach demands considerably less code and expertise of semantic technologies from the SWS application provider.

2. 5 Run-time Service Composition

The INFRAWEBS user request modeling enables definition of "complex goals" which define independent user objectives that are interconnected by certain constraints. A typical example of a complex goal in the SFS scenario (see Section 4.1) is a request to book a flight for certain dates and reserve a room at a hotel in the flight destination city for the same dates. In this example, the independent goals are booking a flight and making hotel room reservation, but they are implicitly constrained by the same arrival and check-in dates as well as by same flight destination and hotel location. Complex goals are also predefined by application providers at design-time and capture any possible combination of "simple" goals that can be useful for the application functionality.

When the application user works with functionality represented as a complex goal (e.g. creates a user package for flight reservation from Munich to Madrid on March, 8th and books a hotel in Madrid for the same date), the same mechanisms for goal instantiation and discovery are used. In this case the discovery can find either atomic or composed services that match the goal instance, but in some cases no single service will match the goal. In such cases the INFRAWEBS run-time composition mechanism ([3]) is used to dynamically find an appropriate solution.

The Run-Time Composer performs its work in two phases. In the first phase (Fig. 7) the user goal is recursively decomposed into two sub-goals, one of them is atomic and the other one is either atomic or composite. A goal is always decomposed back to the original sub-goals that compose it. This is done using the "source" information kept as part of the composite goal description, see [3]. Both sub-goals are matched to services. If the composite sub-goal does not match any services, then further attempt at its decomposition is made. If no matches for an atomic sub-goal are found at some point, the process of run-time composition fails. When matching services are found for a sub-goal the application user is asked to select one particular service to be included in the ad-hoc composition set. The rest of the matching services are also cached as alternatives that can be used for substitution in case of execution failure.

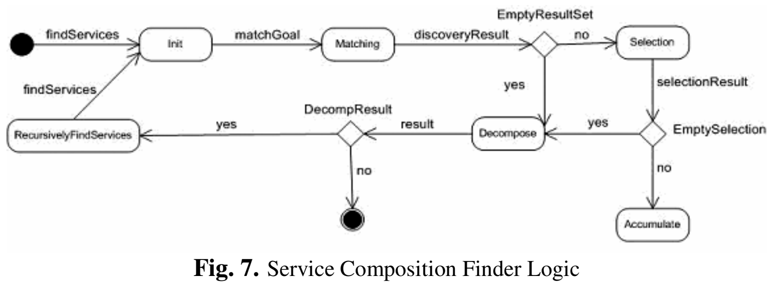


Fig. 7. Service Composition Finder Logic

The second phase of run-time composition is composition execution, when services of the dynamically composed set are iteratively executed, one by one (Fig. 8). Each service execution may result in providing the expected outcome (successful execution), in failure or in user input request, as described earlier. Composition context (input and output ontologies) accumulates all outcomes (instances) generated by successfully executed services.

If a service requests additional input, its execution is stopped and different options for getting this input are investigated. If any of the other services in the run-time composition can provide that input as a result of its execution, the service is put in a waiting list and execution continues with the next service in the composition. If no service can provide this input, the user is asked to enter it through the application. When all services of the composition have already been executed, the services in the waiting list are executed again based on the continuously updated context. The process is iteratively repeated until all services in the dynamically composed set are executed successfully or one of them fails. If any of the composed service executions fails, service replacements are presented to the user in order to continue the execution with an alternative service.

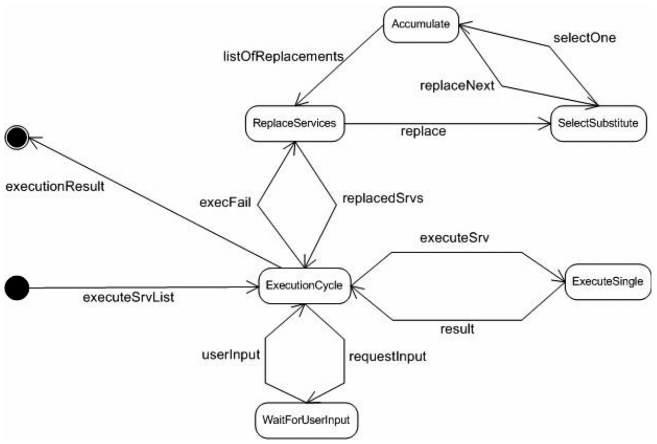


Fig. 8. Composition Execution Engine Logic

3. Integrated INFRAWEBS Framework

The INFRAWEBS Framework is implemented as an extensible Enterprise Service Bus (ESB) middleware that exposes the public methods of the INFRAWEBS components and can be easily extended by external components or services, [12]. The *Integrated INFRAWEBS Framework (IIF)* can be seen as an underlying infrastructure for communication and integration of all the INFRAWEBS components, and as a unique selling point for exposing the functionality of such components to the external world in the form of services. The IIF is deployed in a peer-to-peer network (Fig.9), with possible integration of components with different technologies within the peer. The IIF allows access to the methods of the components via Java APIs or Web service wrappers, so any application able to use these technologies can interoperate with INFRAWEBS components. IIF provides native support for Java-based components and partial support for non-Java based components, using WS technology.

Every peer can be deployed containing all or part of the set of INFRAWEBS stack of components. The IIF hides from the INFRAWEBS users the complexity of dealing with this p2p architecture.

The IIF provides an *IIF Connector* instantiated by Java-based components to allow them to exchange messages and an *IIF Server* routing those messages to the appropriate recipient. The *IIF Server* exposes a complete Web service implementation, which collects the full set of operations supported by the components plugged within the IIF. The service can be used by non-Java components to send messages to other components plugged within the IIF infrastructure or to external applications. IIF provides two ways of communication:

- *Specific* - by instantiating a concrete *IIF delegate* for a particular component's interface to which a message should be sent. The delegate “knows” the requested component and details on how to build and send the message.
- *Generic* - IIF provides methods to generically invoke a remote method. In this case the user just has to know the name of the component method and its parameters.

For both ways of communication IIF supports different kinds of message exchange patterns such as one way, in/out synchronous, in/out asynchronous, publication-subscription and broadcasting (see [12] for details)..

The INFRAWEBS *IIF Connector* is a Java client-side API that can be used for sending and receiving all types of messages in both specific and general ways of communication. For non-java components or applications, the Web service interface of the IIF shall be used instead.

The IIF is able to manage automatically the execution and shutdown of its components, as well as to check if the components are alive or not. Since the IIF consists of a set of different components and tools for creation, maintenance and execution of WSMO-based SWS, it can be considered as a Semantic SOA. The use of the underlying ESB middleware provides the IIF with the necessary extensibility to be a Semantic ESB.

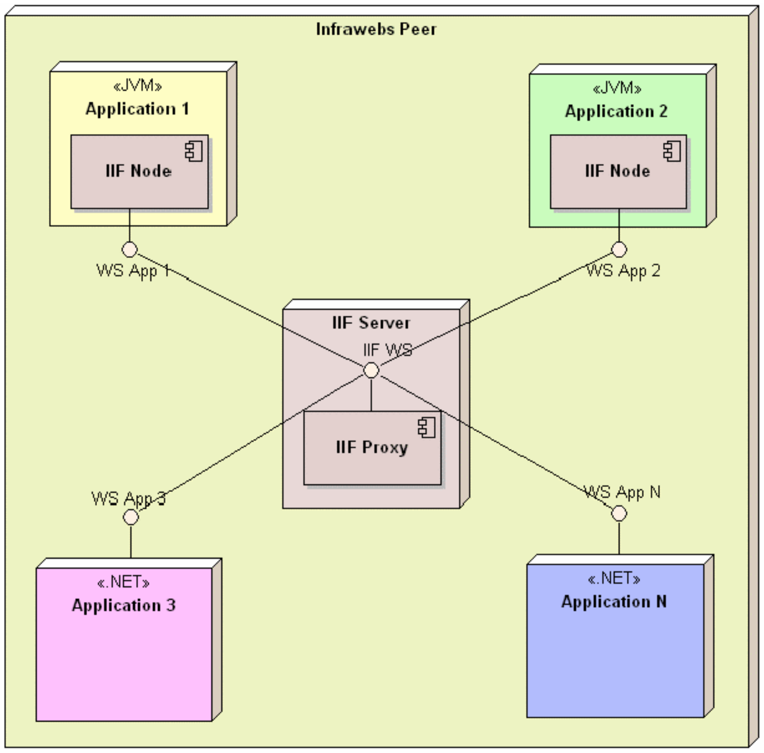


Fig. 9. IIF peer-to-peer architecture

4. INFRAWEBS Framework Evaluation

The INFRAWEBS Framework has been evaluated in two test beds – the first is used in a travel agency scenario [13] and the second is based on an eGovernment scenario [19].

4.1 Test bed 1 - Stream Flows! System

STREAM Flows! System (SFS) is a first prototype application of the IIF, which aims at overcoming such shortcomings of existing Frequent Flyers Programs as impossibility of their users to contract services, or combination of services, using asynchronous, real-time, anywhere and anytime system. The owner of the SFS is STREAM Airlines. Users can obtain points or miles from purchasing services of the STREAM group. These services can be airline tickets, hotel bookings, car rentals and many others. The customer purchases services (paying for them in any kind of money transfer) from many companies (engaged with the SFS program), which collect the information of the customer and send it to the SFS, adding the counterpart of the service in points to the SFS loyalty programme. The SFS semantic Web application collects all the information and stores it into its own databases.

The SFS uses the framework supplied by INFRAWEBS both for design and for runtime activities. First of all the Service Providers have access to the Designer (SWS-D) and the Composer (SWS-C), in order to define the SWS descriptions and compositions that are needed. In general, SWS descriptions can be provided by other parties, and be stored and advertised in different repositories of the INFRAWEBS p2p network.

The main functionality of the SFS allows the user to create or select travel packages. The user is able to create new packages using the framework provided by INFRAWEBS. The choice of a complex package triggers the selection of an appropriate composite goal template allowing dynamic composition of services (e.g. flight + hotel + car rental) during execution. SFS semantic application was described by 12 ontologies, 11 semantic Web services and 16 goal templates.

The test bed has shown several benefits of using applications based on SWS:

- Run-time composition of services: the use of composite goal templates enables easy creation of basic compositions of services in runtime.
- Selection of best offers based on SWS discovery: the use of semantics allows for finding the most appropriate services matching the user's needs expressed as a goal.
- Service providers have fewer difficulties adding new services: each new service that can be potentially used by the application should simply be described as a SWS and incorporated to the repositories, thus minimizing the integration process.
- Ideally no modifications to the code are required: the user input data forms can be created dynamically based on goal template slots (inputs), minimizing the impact on the maintenance of the application.
- As a J2EE application SFS system has been easily integrated with all INFRAWEBS components by using the Java API and the connectors provided by the IIF.

4.2 Test bed 2 – Opening New Business

The second test bed implements an e-Government scenario related to the process of opening a new commercial activity (for example, adding a new shop to an already existing shop chain). Usually, a new shop opening is bound to current local laws and regulations that may be changed in time. That is why, the user should first be familiar with the list of needed documents and certificates required to open the new activity. Then the user has to ask for each document the office or agency authorized to release such a certificate. The process is time consuming, as users need to travel to each appointed agency to post the request, sometimes return back to get the certificate if not shipped to them. The user must also know the proper relations or dependencies among released service certificates.

The proposed SWS application allows any user to take advantage of new technology for accomplishing all needed activities from her personal computer. It enables huge saving of time, error free processes, and much faster responses. The test bed was presented by 9 ontologies, 4 semantic Web services and 4 goal templates.

The test bed offers interesting aspects that complements the results of the SFS test bed:

- The application is well ahead with current services offered to customers.
- The application is a .NET (C#) application that uses the IIF Web service implementation. This has proved the possibility of integration of multiples technologies provided by the SOA approach followed by the IIF.
- Although it is a simple demonstrator aimed at taking advantage mostly from the semantic content, the application could be used by government agencies to show the main advantages they could offer to their citizens and how to dramatically decrease the internal costs due to automatic interaction and less personnel required.

4.3 Framework Evaluation Results

INFRAWEBS Framework has been evaluated in two dimensions ([18]):

- *Environment dimension*: measuring the degree of satisfaction and fulfillment of the Framework objectives with regards to both test bed results, based on the opinion of the different IIF users – SWS providers, SWS test-bed application providers and application end-users.
- *System dimension:* this evaluation takes into account the development of the overall service provision chain (annotation, design, composition, execution, monitoring), focusing on the primary (innovation–related) questions that have been identified. The system dimension does not depend on a particular demonstrator, because it is more related to the Framework capabilities and features.

The evaluation has shown that INFRAWEBS offers a framework covering the whole SWS life-cycle - from design and static composition - to discovery, run-time composition, execution and monitoring. It gives the possibility of creating a new set of semantic-enabled applications that was not possible to develop before in such a consistent way with pre-existing platforms or frameworks.

Moreover, INFRAWEBS is fully based on the current state of WSMO and provides advanced tools for the whole WSMO community. The INFRAWEBS design-time components offer graphical user interface for designing and publishing the semantic descriptions of WSMO-based Web services and goals in an easier way than before. They allow the user to create such semantic objects without any knowledge of WSML language used for their descriptions.

All INFRAWEBS components make use of the INFRAWEBS Integration Framework (IIF) to communicate with each other. Such Framework allows the integration of components of different technologies, and by using Web service interface can be used as an open framework. The use of both a Java-based API and Web service interfaces for the IIF methods allows the application providers to access INFRAWEBS features independently of the technology they use. Moreover, a unified API, provided by the SAM component, allows application providers to interact with the run-time INFRAWEBS Environment without writing or reading any WSML expressions, significantly facilitating the process of creating semantically-enabled applications.

Along with the clear benefits of INFRAWEBS, the evaluation process has allowed to identify some underlying assumptions that should be known for better understanding the applicability of the INFRAWEBS Framework and trends for its future development. The most important of them are:

- All INFRAWEBS components have been developed in strict conformance with the current WSMO specification. Since the WSMO itself is in the process of active development, it can lead to necessity for adjusting the future version of the INFRAWEBS Framework.
- The current version of the INFRAWEBS Framework does not support ontology mediation. All ontologies used are assumed to be known by all service and service application providers. The ontologies are shared using the DSWS-R functionality.
- User requests to a SWS application are expressed as WSMO goals, which are automatically constructed based on the predefined goal templates created in advance by the application developers. This restricts a set of possible goals that potentially can be formulated by the user, but allows the users with no knowledge of WSMO to use INFRAWEBS applications.
- The INFRAWEBS run-time composition is only possible for goals created from goal templates preliminary designed by the INFRAWEBS Goal Editor. That is why, in order to be decomposable, a WSMO goal created by any other tool should be rewritten by means of the Goal Editor.
- Since INFRAWEBS deals with WSMO-based semantic services, the execution of such services requires the presence of so called “adapters”, which ground ontological concepts used in the semantic descriptions to the XML Schema data types used in the WSDL descriptions of the Web services. Automatic creation of such adapters is an ongoing research in WSMO, and that is why, the current version of the INFRAWEBS Framework does not include tools for adapter design.
- Because of lack of clear specification of WSMO-based service orchestration, the current version of INFRAWEBS Composer offers a limited support for orchestration - only on the functional level.
- The INFRAWEBS choreography engine does not clearly guide the user during the execution process. A better engine would be a desirable future enhancement.

5. Conclusion and Future Trends

In this paper we have presented the main results of the IST FP6 INFRAWEBS project. The project has developed an easy and effective way of constructing and using semantic descriptions for existing and new Web services. The advantage of semantic Web services is that they can be understood better by other programs and human beings, since not only syntactic information is available, but also the meaning (semantics). The presence of such semantic information allows for better possibilities for searching and finding useful services.

The project has adopted the WSMO and WSML standards and imposed no additional requirements to them. Therefore, the advanced software components developed during the project are of interest to the whole WSMO community. Most of the developed components are available as open source software under LGPL license.

One of the novel aspects in the approach of INFRAWEBS is the combination between the Organizational Memory (OM) and the Semantic Web Service Designer. This tool allows a user to compose a WSMO-based semantic description of a given Web service based on existing WSDL description of this service and a set of WSML ontologies. To ease this process the OM can be consulted to find look-alike semantic Web services that can serve as templates for the WSMO object under construction. In fact, the use of the OM as a knowledge base holding knowledge about available semantic Web services makes the otherwise tedious process of composing a WSMO object (requiring expert knowledge of the WSMO model and WSML language) a task doable for the ordinary web service developers. Easing the hurdle of WSMO object construction is an achievement of the project that has a potential impact on the adoption of semantic Web services on a larger scale.

INFRAWEBS offers a SOA framework (IIF) based on an ESB middleware which is easy to use by different users (application providers, designers of SWS, etc.) and allows the integration of components of different technologies.

All of the above mentioned analysis allows us to conclude that INFRAWEBS is one of the first frameworks for semantic Web service engineering that covers the whole SWS life-cycle and allows creation of complex semantically-enabled applications.

References

1. G. Agre. INFRAWEBS Designer – A Graphical Tool for Designing Semantic Web Services. In: *AIMSA 2006, LNAI 4183*, Springer-Verlag Berlin Heidelberg, 2006., 275-289.
2. G. Agre. Using Case-based Reasoning for Creating Semantic Web Services: an INFRAWEBS Approach. In: *Proc. of EUROMEDIA'2006*, April 17-19, 2006, Athens, Greece, 130-137.
3. G. Agre and Z. Marinova. An INFRAWEBS Approach to Dynamic Composition of Semantic Web Services. *Cybernetics and Information Technologies (CIT), Volume 7, №1*, 2007, Bulgarian Academy of Sciences, Sofia, 45-61.
4. G. Andonova, G. Agre, H.-Joachim Nern, A. Boyanov. Fuzzy Concept Set Based Organizational Memory as a Quasi Non-Semantic Component within the INFRAWEBS Framework. In: Bernadette Bouchon-Meunier (Ed.) *Proceedings of IPMU2006*, July 2-7, 2006, Paris, France, Editions EDK, ISBN 2-84254-112-X, 2268-2275.
5. http://asg-platform.org/, Last accessed May 2007.
6. T. Atanasova, H. Daskalova, V. Grigorova and D. Gulev. Design& Realisation of Case-based Composition of SW services in Design Time (Design-time Composer). *INFRAWEBS Deliverable D5.4.2*, September 2006.
7. Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A.; Predoiu, L., Kifer, M. and Fensel, D. D16.1 – The Web Services Modeling Language (WSML). *WSML Draft*, October 2005.
8. M. Colombo, E. Di Nitto, M. Di Penta, D. Distante, and M. Zuccal. Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems. In: *Proc. of the 3rd International Conference on Service Oriented Computing (ICSOC),* Amsterdam, the Netherlands, December 2005.
9. http://dip.semanticweb.org/, Last accessed May 2007.
10. L.Kovács, A. Micsik and P. Pallinger. Handling User Preferences and Added Value in Discovery of Semantic Web Services. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2007), 09-13 July 2007, Salt Lake City, Utah, USA, pp. 225-232,
11. http://knowledgeweb.semanticweb.org/, Last accessed May 2007.
12. A. López, T. Pariente and Y. Gorroñogoitia . Final INFRAWEBS Integrated Framework (IIF). *INFRAWEBS Deliverable D11.3.6*. February 2007.
13. O. López, A.López Pérez, C. Pezuela, Y.Gorroñogoitia, E. Riceputi. Requirement-Profiles & Technical Risk Management & Demonstration & Testbeds & Specification Checker. *INFRAWEBS Deliverable D10.1-2-3-4.1*, November 2005.
14. Z. Marinova, Agre, G., Ognyanov, D. Final Dynamic DSWS-R and integration in the IIF. *INFRAWEBS Deliverable D4.4.3*, February 2007.
15. A. Micsik, L. Kovács, P. Pallinger, Z. Tóth. Specification and Realisation of the Execution Control Component. *INFRAWEBS Deliverable D6.3.3*, February 2007.
16. J. Nern, A. Boyanov and G. Jesdinsky. Specification & Realised Reduced Organisational Memory and Recommender Tool. *INFRAWEBS Deliverable D2.1.1*. November 2005.
17. http://www.daml.org/services/owl-s/, Last accessed May 2007.
18. T. Pariente Lobo, A. Lopez Perez and J. Arnaiz Paradinaz. Demonstrator. *INFRAWEBS Deliverable D10.8.3*, February 2007.
19. E. Riceputi, Requirement Profile 2 & Knowledge Objects. *INFRAWEBS* Deliverable D10.5-6-7.2, September 2006.
20. D. Roman, U. Keller, H. Lausen (eds.) Web Service Modeling Ontology, October 2006, *WSMO Final Draft*.
21. J. Saarela. Specification & General SIR, Full Model & Coupling to SWS Design activity. INFRAWEBS *Deliverable D3.2-3-4.2*, July 2006.
22. J. Scicluna, A. Polleres, Dumitru Roman and D. Fensel. D14v0.2. Ontology-based Choreography and Orchestration of WSMO Services. *WSMO Final Draft* 3rd February 2006.
23. J. Scicluna, Marinova, Z.; Agre, G. D7.4.3 Final SWS-E and Running P2P-Agent. INFRAWEBS Deliverable *D7.4.3*, February 2007.
24. http://sekt.semanticweb.org/, Last accessed May 2007.
25. Software, Services and Complexity Research in the IST Programme: An overview. *Software Technologies Unit, Information Society and Media DG, European Commission, Ver. 1.0, draft*, September 2006.

Formulation of Hierarchical Task Network Service (De)composition

Seiji Koide[1] and Hideaki Takeda[2]

[1] The Graduate University for Advanced Studies (SOKENDAI), 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430 Japan
koide@nii.ac.jp,
WWW home page: http://www-kasm.nii.ac.jp/~koide/
[2] National Institute of Informatics and SOKENDAI, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430 Japan
takeda@nii.ac.jp

Abstract. The Hierarchical Task Network (HTN) planning method is conceived of as a useful method for web service composition as well as being for task planning. However, no complete success of service composition by HTN is achieved as yet. The reason is the Web service composition process involves interactive dataflow between variables in precondition and input/output parameters of services. While such dataflow requires to evaluate variables in order to compose services, the performance of services is undesirable, because world-altering Web services cause to alter the world in composition processes. In this paper, instead of the HTN task planning method, we address more radical approach of HTN method for web service composition and decomposition on the premise of the openness and uncertainty of WWW. We capture composite services, which contain abstract concepts with respect to the variables of Web services, as abstract programs to be tailored to individual users and to be instantiated to executable programs in which every variable can ground in execution. In this view, the web service composition process can be conceived of as a sort of automated programming process, and HTN is deemed as a structural workflow or a prototype of actual programs. We formalize web service composition/decomposition by HTN method using the idea of satisfiability of situation calculus, and address the algorithm for Web service (de)composition that does not require to perform services.

1 Introduction

The Hierarchical Task Network (HTN) planning method is conceived of as a useful method for web service composition, and several works on the web service composition have been attempted with HTN [6]. However, there are a few but serious discrepancies between task planning and service composition. First, a web service involves inputs and outputs in addition to precondition and effects, and the interaction between I/O parameters and precondition/effects may happen. Second, the constraint in web service composition is not partial orders of tasks but control constructs in which there are dataflows and control flows between subtasks. Therefore, the semantics of web service composition are much more complex and analogous to programming rather than planning. The realization of the service composition ought to differ from that of task planning.

Sirin, et al. [10] achieved the web service composition using the HTN task planning method. They invented the translator from the OWL-S service description to the SHOP2 [6] task planning domain. To enable the translation from the web service composition domain to the HTN task planning domain, they assumed that an atomic Web service is either a strict information providing Web service, which does not have the effects, or a world altering Web service, which does not have outputs, but they did not maintain world-altering and information-providing services.

This assumption is ascribed to the fact that they used SHOP2, which was originally developed for classical task planning problems, and they did not reformalize HTN method for Web services. A precondition of method/operator in HTN is evaluated to test whether the precondition hold on the state. In case that a variable in a precondition is unified to some output parameter of a Web service in planning, we need the value of the output. They argued that we do not want to actually alter the world during planning, and do want to gather information from information-providing Web Services.

Besides the complexity in the service composition, Semantic Webs stand on the assumption that the world is open and dynamic. Therefore, we must consider the uncertainty of the world in service composition and execution processes. More precisely, we cannot expect service precondition that hold in a composition process also hold in a execution process. As a result, we cannot help but abandon the soundness and completeness of planning when we consider both composition process and execution process in the dynamic world. In fact, the authors embrace the problem how to deal with the progression of situation in our decision-making support application [5], in which light anomalies of the rocket launch operation process in planning time may change to heavy anomalies in execution time and one control mode may progress to the successive control mode.

The authors claim that the uncertainty of WWW must be coped with by Web service agents situated in circumstances, that is known as situated planning agent [9], which monitors the plan execution, detects failures of the performance, replans the plan, and adapts the behavior to changeable situations. The behavior under the uncertainty is the common observation among animals and intelligent human beings in the real world, and we argue it is the same on even Web service agents in use.

Under the premise of the situated planning agent in the future, in this paper, we formalize the web service composition and decomposition with expanding the HTN formalization using the terminologies in automated planning in state space [2] and situation calculus [8]. In Section 2, we review the formalization of task planning by HTN [2]. Then, we expand it for Web service composition. We discuss the concept of applicability, satisfiability, executability for Web services from the viewpoint of situation calculus, and address an algorithm for service composition/decomposition. In Section 3, we describe the implementation of the service (de)composition algorithm, which is straightforwardly embodied of the algorithm using nondeterministic search technique with computational continuation. The algorithm is incomplete because of the open world assumption but do not compel us to perform services in Web composition process. In Section 4, we discuss the related work, and we make some concluding remarks.

2 Formalisation of HTN for Web Services

2.1 Hierarchical Task Network Planning

In this subsection, we review the classical task planning by HTN according to the description in [2]. The expansion of HTN to web service composition and decomposition is described after the next subsection.

Let $\mathcal{L}$ be a first-order language for planning, in which there are predicate symbols, constant symbols, and variable symbols. If an atomic formula, which does not contain logic connectives, does not contain any variable symbol, it is called *ground atom*, otherwise *unground atom*.

A *state* s is a set of ground literals, i.e., ground atoms or negations of ground atoms. S denotes a set of states. An atom p holds in s iff $p \in s$.

Definition 1. *A planning operator in task planning is a triple such that*

$$o = \langle name(o), precond(o), effects(o) \rangle.$$

- $name(o)$ is a name of operator, which has a syntactic expression of the form $n(x_1, ..., x_k)$ where n is a unique symbol called *operator symbol*, and $x_1, ..., x_k$ are all of variable symbols that appear anywhere in o.
- $precond(o)$ is the precondition of o, and $effects(o)$ is the effects of o. Both are a set of literals.

Definition 2. *An HTN method in task planning is a 4-tuple, that is,*

$$m = \langle name(m), task(m), subtasks(m), constr(m) \rangle.$$

Where $name(m)$ is an expression of the form $n(x_1, ..., x_n)$, n is a unique symbol for the method, and $x_1, ..., x_n$ are all of variables that occur anywhere in m. A set of pair $\langle subtasks(m), constr(m) \rangle$ makes a task network for m.

If an instance of operator contains ground atoms and does not contain unground atoms, it is called *ground*, otherwise *unground*. A ground operator that includes ground atoms in s is called *action* for s.

A task is an expression of the form $t(r_1, ..., r_k)$ like the name of operator and method, but a name symbol t of task differs from a method name symbol n, and a task has fewer parameters than the name of the corresponding method. t is called *task symbol*, and $r_1, ..., r_k$ are terms. Every operator symbol is a task symbol, and every operator can be a task. When a task symbol t is an operator symbol and its terms can be unified to variables of the operator, the task is called *primitive*, otherwise it is *nonprimitive*. A task such as $task(m)$ on a method m is nonprimitive, but a subtask may be an operator.

Definition 3. *A task network in task planning is a pair*

$$w = \langle U, C \rangle.$$

Where U is a set of all task nodes in the network and C is a set of constraint such as partial task ordering and preconditions for tasks. Note that w contains only problematic front part of whole network in partial order HTN, and it evolves along with the progression of the state and the plan.

Each task node $u \in U$ contains a task t_u. If all of tasks in U $\{t_u \mid u \in U\}$ are ground, then w is called ground, otherwise w is unground. If all of tasks $\{t_u \mid u \in U\}$ are primitive, then w is called primitive, otherwise nonprimitive.

2.2 Web Service Network by HTN

In this subsection and hereafter, we formalize Web service network by extending HTN task planning described above.

We expand the definition of state so that it includes not only atoms from precondition and effects but also inputs and outputs of Web service. Outputs of Web service are added as atomic formula into the state as well as positive effects. Note that inputs of Web services are taken from atoms in the state and/or outputs of predecessor services. The data stream from an input to an output via services is called *dataflow*, and we call a data stream that streams in and out via services *IO fluent*.

Definition 4. *An atomic service is an expansion of the operator, and de ned as a 5-tuple such that*

$$as = \langle name(as), inputs(as), outputs(as), precond(as), effects(as) \rangle.$$

- $name(as)$ is a name of service. It has a same syntactic expression of the form $n(x_1, ..., x_k)$ as operator name $name(o)$, but variable symbol x_i may appear not only $precond(as)$ and $effects(as)$ but also $inputs(as)$ and $outputs(as)$.
- $inputs(as)$ denotes inputs to the Web service as, and $outputs(as)$ denotes outputs returned by the Web service. $precond(as)$ represents preserving or causal condition of as for the web service execution. $effects(as)$ is the side effects or causal effects onto the state s by the web service execution. $inputs(as)$ and $outputs(as)$ is a sequence of variables respectively, while $precond(as)$ and $effects(as)$ is a set of literals.

Definition 5. *A composite service is an expansion of the method, and de ned as a 7-tuple, that is,*

$$cs = \langle name(cs), task(cs), inputs(cs), outputs(cs), subtasks(cs), \\ precond(cs), controlConstruct(cs) \rangle.$$

Where the definition of $name(cs)$, $inputs(cs)$, $outputs(cs)$, and $precond(cs)$ are same as the definition of that in atomic service, $task(cs)$ is similar to that of HTN, but the notation of $subtasks(cs)$ and $controlConstruct(cs)$ firstly appear here in a composite service.

A task in Web services is defined as same way in HTN task planning. If the task symbol t is a name symbol of $name(as)$ of atomic service as and its terms $r_1, ..., r_n$ is unifiable to variables of the atomic service, the task is called primitive.

A task network in Web service is a pair of a set of $subtasks(cs)$, and a set of $controlConstruct(cs)$.

Definition 6. *A task network in web service composition and decomposition is expressed as*

$$w = \langle U, CC \rangle.$$

Where U is a set of all task nodes in the network, and CC is a set of all control construct included in the network. Note that the constraint of CC includes control flows, dataflows, and preconditions of task-corresponding composite services and atomic services. The task flows (abstract control flows) and dataflows are contained in $controlConstruct(cs)$. A composite service can contain only one control construct in definition, but a control construct can contain subtasks and sub control constructs in the specific form of various kind of control constructs, specifically, $sequence$, $ifThenElse$, $loopWhile$, etc.

We can consider various controlConstructs, but in this paper we define only three as follows.

Definition 7. *A sequence is a tuple of any number of subtasks.*

$$seq = \langle elt_1(seq), elt_2(seq), ..., elt_k(seq) \rangle$$

Where $elt_i(seq), i \leq k$ is a subtask in the cs. In the execution of $sequence$, $elt_i(seq)$ is performed in the order of the sequence.

In HTN task planning, constraint $constr(m)$ represents partial orders of subtasks. Therefore, a predecessor and the successor of tasks can be interleaved with another task. However, no service can part the task sequence in sequential performance of Web services.

Definition 8. *An $ifThenElse$ is a 3-tuple as follows.*

$$ite = \langle if(ite), then(ite), else(ite) \rangle$$

Where $if(ite)$ is a condition that does not cause any side effect in evaluation, and $then(ite)$ is a subtask or a control construct that is performed when $if(ite)$ holds in the state. Optional $else(ite)$ is a subtask or a control construct that is performed when $if(ite)$ does not hold.

Note that $then(ite)$ is performed if and only if the condition of $if(ite)$ holds in the state, but $else(ite)$ is performed in case that not only the negation of condition $if(ite)$ holds but also it is unknown with the premise of the open world assumption of Web Semantics.

Definition 9. *$loopWhile$ is a 2-tuple such as*

$$lw = \langle while(lw), seq(lw) \rangle$$

Where $while(lw)$ is a condition during which holds $seq(lw)$ is performed repeatedly. $seq(lw)$ is a $sequence$. Note that performance of $sequence$ is terminated even if $while(ls)$ becomes unknown in the execution process.

2.3 Web Service Composition and Decomposition by HTN

On one hand, the objective of task planning is to obtain totally ordered sequences of actions that achieve a goal, where a goal g in task planning is a set of ground literals produced from effects. On the other hand, the objective of web service composition and decomposition is to obtain a set of executable task flows or a program that is an instantiated network of web service performance in the environment, where input and output parameters are variables to be unified to constants in state, and a goal g may include output variables.

We have two categories of goals in essence in web service composition, i.e., the alteration of world by Web service performance and the information retrieval by performance of Web services. The former is the same as task planning but the latter differs from task planning in given goals. We give ground literals in atomic formula (say using individuals in OWL, like $on(A\ B)$ for Block A and Block B) as goals in task planning, but we do not designate values of service outputs as goals in web service composition. The values of output variables are the very thing we want to know in the information providing services. We are able to only designate types of variables (note that a variable is also an instance of an OWL class but unifiable to the range of instances of a class) as goal (say $?roomtype$, a room type of hotel available). Web service decomposers must generate instantiated task flow that includes atomic services that achieve world-altering goals and information-retrieval goals. Moreover, the coupling of world-altering service performance and information-retrieval service performance may happen through variables.

In this subsection, we discuss the (de)composability of services from the standpoint of satisfiability in situation calculus [8].

Satisfiability of Web Service: In HTN task planning, an operator is an abstraction that stands for all instance operators named by an operator symbol. In an instance of operator o, a variable symbol in $name(o)$, $precond(o)$, and $effects(o)$ is substituted with a corresponding constant symbol. In Web service (de)composition, an atomic service can be instantiated through the substitution of all variables except IO parameters in the precondition, effects, inputs, and outputs with ground literals in the state space. In addition, IO parameters in a service must satisfy the state in the execution of the instantiated service.

To discuss the interpretation of assertions in Web service (de)composition, we consider an state transition machine. Let Σ be the state transition machine for task planning [2]. We consider a mapping from a state s in assertions of the planning problem P to a state in Σ. For a state s described in planning language $\mathcal{L}$, the corresponding state in Σ is denoted by $s^{\mathcal{I}}$, and $\mathcal{I}$ is called *interpretation.* In case that there is a mapping from s_{i-1} to $s^{\mathcal{I}}_{i-1}$ and s_i to $s^{\mathcal{I}}_i$, if an instance operator o in P that links from s_{i-1} to s_i has a mapping to $o^{\mathcal{I}}$ that links from $s^{\mathcal{I}}_{i-1}$ to $s^{\mathcal{I}}_i$, it is called *satis able* with respect to the operator o.

We expand this interpretation for task planning to Web service (de)composition. In case that there is a mapping from s_{i-1} to $s^{\mathcal{I}}_{i-1}$ and s_i to $s^{\mathcal{I}}_i$, and from an atomic web service as in P that links from s_{i-1} to s_i to $as^{\mathcal{I}}$ that links from $s^{\mathcal{I}}_{i-1}$ to $s^{\mathcal{I}}_{i}$, we call it *satis able* with respect to the atomic web service as.

For an atomic service as, iff the precondition $precond(as)$ is satisfiable in s, namely a interpretation of condition of $precond(as)$ holds in an interpretation $s^{\mathcal{I}}_{i}$, and inputs $inputs(as)$ is satisfiable in s, namely we can find the unification for each variable of inputs onto s and s has a mapping $s^{\mathcal{I}}_{i}$, where the unification is identical to that for the instantiation of $precond(as)$, then the atomic service as is satisfiable with the respect to s.

$$s \models as \Leftrightarrow (s \models precond(as)\ \wedge s \models inputs(as))$$

On the other hand, a composite service cs is satisfiable, iff $precond(cs)$, $inputs(cs)$, and $controlConstruct(cs)$ is satisfiable.

$$s \models cs \Leftrightarrow (s \models precond(cs)\ \wedge s \models inputs(cs)\ \wedge s \models controlConstruct(cs))$$

When the precondition and inputs of a service are satisfiable, let us call the service *applicable*. An applicable atomic service is always satisfiable but an applicable composite services are not necessarily satisfiable. In other words, an applicable atomic service has a model on s but an applicable composite service may have no model on s.

In order to know the satisfiability of a composite service, we need to know the satisfiability of *controlConstruct* and task.

Satisfiability of Task: In case that a task $t(r_1, ..., r_n)$ is primitive, the task $t(r_1, ..., r_n)$ is applicable and satisfiable, iff the corresponding atomic service as which may be partially instantiated is satisfiable.

$$s \models t \Leftrightarrow (name(as) \equiv n(x_1, ..., x_n) = \sigma(t(r_1, ..., r_n), \theta)) \wedge (s \models \sigma(as, \theta))$$

Where $\sigma(t(r_1, ..., r_n), \theta)$ expresses the substitution of $t(r_1, ..., r_n)$ by unifier θ for s. Note that θ contains the accumulation of the past unification in (de)composition process.

In the case that a task is nonprimitive and the corresponding service is composite, then the task $t(r_1, ..., r_n)$ is satisfiable, iff cs is satisfiable.

$$s \models t \Leftrightarrow (name(cs) \equiv n(x_1, ..., x_k) \simeq \sigma(t(r_1, ..., r_n), \theta)) \wedge (s \models \sigma(cs, \theta))$$

Where $k \geq n$ and $\simeq$ expresses the equality of name $n = \sigma(t)$ and $x_i = \sigma(r_j)$ when $k = n$ but some of parameters x_i may not be unified to r_j when $k > n$.

Progression in Web Service Composition Let consider a progress by task t for state s_{i-1}. It seems to be the same as the expression on action $s_i = do(a, s_{i-1})$ by Reiter [8] whereas a is an action that contain only preconditions and effects. The application of task t under the state s_{i-1} yields the state s_i. However, we cannot really execute the task in service (de)composition processes, if the task is a world-altering task. Then, we cannot know values of IO parameters. Instead, we make a progression by the unification that change abstract types of variables to more special types. The progression in web service (de)composition by unification is expressed as below.

$$s_i = \gamma(s_{i-1}, t)$$

Satisfiability of *sequence* Control Construct: Let be *seq* a *sequence*, and let s_0 the initial state for *sequence seq*. Let us express the performance of $elt_i(seq)$ by inputs $in^1_{i-1}, ..., in^k_{i-1}$ as $elt_i(seq)(in^1_{i-1}, ..., in^k_{i-1})$. If $elt_1(seq)$ is satisfiable for s_0 and inputs $in^1_0, ..., in^k_0$, then we can make a progress for s_0 and obtain s_1 for $elt_1(seq)$ by making a progress of unification.

$$s_1 = \gamma(s_0, elt_1(seq)(in^1_0, ..., in^k_0))$$

Then, if $elt_2(seq)$ is satisfiable for s_1 and inputs $in^1_1, ..., in^l_1$, we can obtain the next state s_2, and so forth. Please note that here we omit the substitution of each element by the unifier that accumulates unifications according to the progress. Namely, $elt_2(seq)$ is $\sigma(elt_2(seq), \theta)$ exactly.

$$s_2 = \gamma(s_1, elt_2(seq)(in^1_1, ..., in^l_1))$$

We cannot say that if all tasks $elt_i(seq)(in^1_{i-1}, ..., in^k_{i-1})$ in *sequence* are independently satisfiable for each states then the *sequence* is satisfiable, because the satisfiability of the control construct also depends on the dataflow. We represent this constraint of dataflow in control construct $dataflow(seq)$. If the dataflow constraint is held correctly in the progress of control constructs, we call the control constructs has a model. Thus,

$$s_0 \models seq \Leftrightarrow s_0 \models dataflow(seq) \bigwedge_{i=1 \quad k} s_{i-1} \models elt_i(seq)$$

Note that each task $elt_i(seq)$ may be composite and its satisfiability is decided with the satisfiability of the corresponding composite service. Obviously, this definition for the control construct and the composite service is recursive but the computation of satisfiability terminates, because the composite service is decomposed down to atomic services in an acyclic task network and the satisfiability computation for every atomic service terminates.

Satisfiability of *ifThenElse* Control Construct: Let be *ite* an instance of *ifThenElse*, and s_{i-1} is the state for *ite*. Then, we have three possibilities on the satisfiability of *ite* with respect to the condition.

– For positive condition, we have

$$s \models ite \Leftarrow (s \models if(ite) \ \wedge s \models then(ite)).$$

– For negative condition, we have

$$s \models ite \Leftarrow (s \models \neg\ if(ite)\ \wedge s \models else(ite)).$$

– For unknown condition, we have

$$s \not\models ite \Leftarrow (s \not\models if(ite)\ \wedge s \not\models \neg\ if(ite)).$$

If $if(ite)$ holds and $then(ite)$ is satisfiable for s, then the ite is satisfiable. If the negation of $if(ite)$ holds and $else(ite)$ is satisfiable, then ite is satisfiable. However, when the condition of $if(ite)$ is unknown, we cannot deduce whether ite is satisfiable. In the execution process, the value of $if(ite)$ is usually known as a result of the service execution and the progression of state, but it may be unknown in composition processes without the service execution. Therefore, the decomposer may not proceed the reasoning at this branch possibility of control. In such a case, an agent may select one of two strategies, i.e., *speculative strategy* or *assurance strategy.* In the speculative strategy, the agent aggressively takes one of the possibilities of branches, and the soundness of the instantiated program is not guaranteed. If the executer fails the execution of the generated program, the agent repairs the failure and replans at the point. In the assurance strategy, the agent defensively carries the incomplete programs to the execution phase and executes it up to the undecomposed point. The agent restarts to plan when the value of the $if(ite)$ is known. Thus, the functionality of incremental planning and replanning is requisite for the agent in the open world.

Satisfiability of *loopWhile* Control Construct: Let be s_0 an initial state for an instance of $loopWhile$, lw. When a while-condition $while(lw)$ of lw does not hold by the negation of $while(lw)$, lw is satisfiable and the state s_0 does not evolve. If while condition $while(lw)$ holds for s_0, then lw is satisfiable iff $seq(lw)$ is satisfiable. However, when it is unknown whether $while(lw)$ hold, we treat it in the same way as $ifThenElse$.

As a result of one round of iteration for sequence $seq(lw)$, the state evolves and this process is repeated again while $while(lw)$ is satisfiable for the evolving state in the loop.

$$\begin{aligned} s \models lw \Leftarrow&\ \ s \models \neg\ while(lw) \\ s \models lw \Leftarrow&\ \ s \models while(lw)\ \wedge seq(lw) \\ s \not\models lw \Leftarrow&\ \ s \not\models while(lw) \wedge \not\models \neg while(lw) \end{aligned}$$

Note that the states may evolve on the satisfiability check for even the same procedure, because types of variables evolve more precisely step by step using the typed unification, which is described later. In the worst case, the evolution stops even if $while(lw)$ holds, thus the decomposer must detect no progression in the iteration and exit from the loop.

2.4 Algorithm of Web Service Composition and Decomposition

We cannot decompose the control construct of composite services into subtasks as HTN task planning does. Instead, we collect all of variable bindings that make a control construct in hand satisfiable, and search all possibilities of progression by substituting all variables in service parameters. Here note that input and output parameters are typed in DL or OWL, and variables in precondition and effects also typed. Therefore we need typed unification to compute satisfiability. The typed unification algorithm is described in the next section.

Let us call an instance of atomic service *ground*, if it contains ground atoms except unground variables that are input and output variables in some atomic services. In HTN method for task planning, a ground instance of operator that is applicable to s is an active candidate for the plan solution. There is no unground variable in ground operators. In HTN method for service (de)composition, we have input and output variables in ground atomic services. Therefore, the active candidate for the plan solution must be a pair of ground atomic service and its unifier that instantiated the atomic service.

Suppose that a task node u in U, $u \in U$, is in the network w. Here $task(as) = t_u$ or $task(cs) = t_u$. When $w = \langle U, CC \rangle$ is primitive, if U is grounded and CC is satisfiable for s, then w is a solution for s such that the executer can execute CC for s. Then as instantiates u so as to produce the instantiated task network w' from w. If $w = \langle U, CC \rangle$ is nonprimitive, then w is a solution for s if we can find a satisfiable unification that satisfies CC and a primitive task network w' is obtained as a result of decomposition of cs. In other words, the problem solving of service composition is to find the path of evolution of partial network w by decomposition for the subtasks of cs in the unification.

The algorithm of HTN web service composition and decomposition is described as follows. Where s is a state in a situation, w is a part of whole task network that is to be instantiated. w' is a part of task network that is instantiated. The initial input of w is a network that includes only a top task, and w' is null set. AS is a set of atomic services, and CS is a set of composite services. D is a domain knowledge of the target field.

```
procedure SWHTN($s, w', w, AS, CS, D$)
  if $w = \emptyset$ then return $w'$
  nondeterministically choose any $u \in U$ that has no predecessors in $w$
  if $t_u$ is primitive then
    $active \leftarrow \{\langle \sigma(as), \theta \rangle \mid as =$ discover$(t_u, w, AS, D)$ and $as$ is satisfiable in $s$
                      $\theta$ is a unifier with satisfiable bindings of $as$
                      $\sigma(as)$ is a substitution of $as$ with $\theta\}$
    if $active = \emptyset$ then return fail()
    nondeterministically choose any $\langle \sigma(as), \theta \rangle \in active$
    SWHTN($\gamma(s, as), w' + \{\sigma(u)\}, \sigma(w - \{u\}), AS, CS, D$)  ; { } means a network.
  else ; ; $t_u$ is nonprimitive.
    $active \leftarrow \{\langle \sigma(cs), \theta \rangle \mid cs =$ discover$(t_u, w, CS, D)$ and $cs$ is satisfiable in $s$
                      $\theta$ is a unifier with satisfiable bindings of $cs$
                      $\sigma(cs)$ is a substitution of $cs$ with $\theta\}$
    if $active = \emptyset$ then return fail()
    nondeterministically choose any $\langle \sigma(cs), \theta \rangle \in active$
    return SWHTN($\sigma(s), \sigma(w' + \{u\}), \sigma(w - \{u\} + sub\{u\}), AS, CS, D$)
```

$\gamma(s, as)$ is a progression by an atomic service as. For a nonprimitive service, this algorithm decompose it into subtask nodes and evolves the state by the satisfiable unifier. For a primitive service, this algorithm makes a progression of state and accumulates the instantiated network. This algorithm contains the loop of SWHTN() via the tail recursive call. $fail()$ causes automatic backtracking to the point of the last choice of nondeterministic selection. The algorithm is very similar to HTN of task planning [2] and SHOP2 [6], because we already have a convenient terminology *satis able*. We used it instead of the terminology of *ground action* in HTN task planning for collecting *active* atomic service. The satisfiability checking, which deeply searches down to atomic services from a composite service, returns several unifier that satisfy the node to the state s.

In the worse case, we cannot obtain complete solutions from this algorithm, because it is possible that we encounter unknown conditions without the execution of Web services. In such a case, the agent resolves the problem in the manner of the *speculative strategy* or *assurance strategy*.

3 Implementation

3.1 State Space and Variables

Generally, an atom can be expressed as a form $p(r_1, ..., r_{n-1}, r_n)$. If an atom is a state variable such that the combination of predicate p and variables $r_1, ..., r_{n-1}$ has a mapping to r_n on each state s, it can be expressed as $p(r_1, ..., r_{n-1}) = r_n$. A state variable can be also expressed as $p(r_1, ..., r_{n-1}, r_n) = true$. Then, a negation of atom $\neg p(r_1, ..., r_{n-1}, r_n)$ can be expressed as $p(r_1, ..., r_{n-1}, r_n) = false$. On the close world assumption, the absence of positive atom means the negation of the assertion. On the open world assumption, the absence of positive and negative assertion means unknown on the assertion.

The state is expressed as a list of state variables as atom. In Lisp, the following shows an example of the state in which an individual `Lucy` has an appointment, and `HAL` has also an appointment. Note that `Lucy` is already defined as individual of `Person`, and `HAL` is already defined as individual of `Robot`.

```
(setq *state*
      (make-state '((hasAppointment(Lucy) = LucysAppt)
                    (hasAppointment(HAL) = HALsAppt))))
```

On the other hand, we make a typed variable in the following form in our system.

```
(make-condition '((hasAppointment((?p - Person)) = ?appt)))
```

If `Person` is defined as class in OWL and `hasAppointment` is defined as an object property with the domain constraint `Appointment`, the system can create `?p` as an instance of `Person` and `?appt` as an instance of `Appointment`.

Note that this appointment with variables typed `Person` is unified with `Lucy`'s appointment but not unified with `HAL` appointment.

3.2 Typed Unification

The unification algorithm by Russel and Norvig [9] is expanded to accept OWL classes and individuals. A variable is also an individual of a class in the domain. Consider the following unification algorithm, where variablep(x) tests for not lisp symbols but OWL entities whether x is an OWL entities for variable, and variable?(x) tests for lisp symbols whether x is a variable. Note that a variable symbol and a constant symbol are bound to an OWL entity in our system, then a lisp symbol is unified to a lisp symbol in semantics of OWL as shown later.

```
**function** Typed-Unify($x, y, \theta$)
  **if** $\theta$ = failure **then return** failure
  **else if** $x = y$ in semantics of OWL **then return** $\theta$
  **else if** $x \neq y$ in semantics of OWL **then return** failure
  **else if** variablep($x$) **then return** Typed-Unify-Var+($x, y, \theta$)
  **else if** variablep($y$) **then return** Typed-Unify-Var+($y, x, \theta$)
  **else if** variable?($x$) **then return** Typed-Unify-Var($x, y, \theta$)
  **else if** variable?($y$) **then return** Typed-Unify-Var($y, x, \theta$)
  **else if** compound?($x$) and compound?($y$) **then**
    **return** Typed-Unify(Args($x$), Args($y$), Typed-Unify(Op($x$), Op($y$), $\theta$))
  **else if** list?($x$) and list?($y$) **then**
    **return** Typed-Unify(Rest($x$), Rest($y$), Typed-Unify(First($x$), First($y$), $\theta$))
  **else return** failure
```

As shown below, the algorithm of Typed-Unify-Var looks like the same as original Unify-Var in [9] at the surface level, but making a new binding $\{var/x\}$ differs from the original. In addition to the symbol level binding between var and x, the class level bindings are taken into account. If two classes of var and x are disjoint each other, then no unification is made and failure is returned. If two classes relates each other in subsumption relation, then a mapping to the specific class is made. Otherwise, a mapping to the intersection of both classes is made.

```
**function** Typed-Unify-Var($var, x, \theta$)
  **if** $\{var/val\} \in \theta$
    **then return** Typed-Unify($val, x, \theta$)
  **else if** $\{x/val\} \in \theta$
    **then return** Typed-Unify($var, val, \theta$)
  **else if** $var$ occurs anywhere in $x$
    **then return** failure
  **else return** make $\{var/x\} \in \theta$
```

Typed-Unify-Var+ is prepared for the binding of OWL individual objects. In practice, our system is built on top of SWCLOS, which is an OWL Full reasoner and language [4]. In SWCLOS, every OWL entity is an object in CLOS. In the integration of our (de)composition system to SWCLOS, a variable *var* in Typed-Unify-Var+ is an object typed to OWL classes in the domain, while x may be an individual object of domain classes or may be an variable object typed to a domain class.

```
function Typed-Unify-Var+($var, x, \theta$)
  if $x$ is individual then
    if disjoint?(class($var$),class($x$)) then return failure
    else if class($var$) = class($x$) in semantics of OWL
      then return make $\{symbol(var)/symbol(x)\} \in \theta$
    else if subsumed?(class($x$),class($var$))
      then return make $\{symbol(var)/symbol(x)\} \in \theta$
    else if subsumed?(class($var$),class($x$))
      then return make $\{symbol(x)/\text{individual}(\text{class}(var))\ \} \in \theta$
                  make $\{symbol(var)/symbol(x)\} \in \theta$
    else return
      then return make $\{symbol(x)/\text{individual}(\text{intersection}(\text{class}(var),\text{class}(x)))\} \in \theta$
                  make $\{symbol(var)/\text{individual}(\text{intersection}(\text{class}(var),\text{class}(x)))\} \in \theta$
                  make $\{symbol(var)/symbol(x)\} \in \theta$
  else return failure
```

Where individual() creates an individual object of the parameter. Through this unification, the type of variable is specified step by step. The value of variables are bound to abstract concepts to special concepts along with the progression via unification. However, if we have poor ontologies with respect to the class hierarchy, this unification easily leads silly results. For example, if there is no knowledge that xsd:integer is disjoint with xsd:float, the intersection of xsd:integer and xsd:float is resulted. However, if there is an assertion that ship is disjoint with automobile, the system fails to find the route by amphibious-vehicle. Generally speaking, it is valuable to give rich information of negation, disjunction, and complement in ontologies for the open world.

3.3 Nondeterministic Choice by Continuation

The computational continuation is well known as program control technique in Scheme language. In short, it is a program frozen in action [3]. When the computational object that contains the state of a frozen computation is evaluated, it is restarted where it left off. This machinery can be a great help to implement the exception handler, multiprocessing, and nondeterministic search and choice. In order to implement our (de)composer, we have adopted the technique of continuation in Lisp [3]. The (de)composition algorithm SWHTN in Subsection 3.4 can be straightforwardly implemented with the continuation.

4 Related Work and Concluding Remarks

4.1 Toward Reasoning in Services from Reasoning in Action

The study on task planning has a long history. Recently, Ghallab, et al. [2] published a comprehensive text on automated planning of action. Reiter [8] enlightened on task planning problem from situation calculus. From the advent of Semantic Webs, Web Services plus Semantic Webs has emerged as a new field in planning, and many efforts has been made in various approaches. Berardi et al. [1] discussed the synthesis of Web services from situation calculus, but the work still stays at the closed world assumption. Sohrabi et al. [12] demonstrated the web service composition using agent programming language GoLog, which is based on situation calculus. However, the problem of the interaction between precondition and inputs/outputs, which is posed by Sirin et al. in service composition by SHOP2 [10], seems to be left still open.

All of works mentioned above strongly stick the soundness and completeness of service composition. However, the authors argue that the openness and uncertainty of WWW lead us to the incompleteness when we consider the execution process. The problem must be solved by the intelligent behavior of agent in the changeable world. In this paper, we formalized Web service composition/decomposition by HTN using the idea of satisfiability in situation calculus, and addressed the algorithm for service (de)composition. We also suggested that we need situated planning agent that adaptively behaves in use under the incomplete service composition and the uncertainty of WWW with the premise of the open world assumption.

Sirin and Parsia [11] deeply discussed the integration of OWL and the task planner. In a sense, it could be said that this paper is a legitimate argument on the HTN formalization touched by them. We addressed the typed unification to make a progress on variable bindings. The authors' system is based on SWCLOS [4] for OWL reasoner. Sirin and Parsia pointed out the existentially bound variables in preconditions may cause the disparity of binding between planning time and execution time. We have no solution on this problem in this paper. We know that SWCLOS cannot reason out correctly on the existential quantifier. On the other hand, the problem on the creation of anonymous individuals mentioned by them is easily solved with SWCLOS, because SWCLOS is built on top of Common Lisp Object System (CLOS) and every concept and individual is an object, even if it is anonymous.

4.2 Web Service Composition and Decomposition

In this paper, the terminologies of both composition and decomposition, and occasionally composition/decomposition and (de)composition are used. Usually, HTN is conceived as a method for web service composition. However, the composition process in HTN is strictly coupled with the decomposition process. Ghallab, et al. often used the terminology of *decomposition tree* of HTN in their textbook [2]. We consider an agent in which the composer composes Web services in coarse grain size from scratch. In this service composition, the partial order planner technique like UCPOP [7] may be useful rather than HTN. Then, the decomposer in the agent decomposes the composed service into fine grain services by HTN. In this paper, we concentrated the discussion to HTN (de)composition process, in which we have a top task node of HTN and separated other subtasks from work flow library at first, and the top task and related abstract tasks are combined and instantiated along with the reduction the ambiguity of task parameters step by step. We call this HTN planning process *service (de)composition.*

4.3 Framework of Web Service Agent

The agent system includes an executer, memory, and user interface in addition to the composer and the decomposer [5]. The executer interprets and executes the instantiated programs with invoking Web Services. The machinery of memory works as memory for various internal data of agent. Some part in memory reflects the variations of outer world with sensing data and poling queries, etc. The user interface works for the communication between the agent and a user. Some ambiguity and nondeterministic choice in task planning may be solved with the help from the user through this interface.

References

1. Berardi, D., Calvanese,D., Giacomo, D., Mecella, M.: Reasoning about Actions for e-Service Composition. International Conference on Automated Planning & Scheduling, ICAPS 2003 (2003)
2. Ghallab, M., Nau, D., Traverso P.: Automated Planning Theory and Practice. Morgan Kaufmann (2004)
3. Graham, P.: On Lisp. Prentice Hall, (1993)
4. Koide, S. Takeda, H.: OWL-Full Reasoning from an Object Oriented Perspective. Asian Semantic Web Conf., ASWC2006 (2006) 263–277
5. Misono, S., S. Koide, N. Shimada, M. Kawamura, and S. Nagano: Distributed Collaborative Decision Support System for Rocket Launch Operation. IEEE/ASME Int. Conf. Advanced Intelligent Mechatronics, AIM2005, (2005)
6. Nau, D., T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman: SHOP2: An HTN Planning System. J. Artificial Intelligence Research, **20**-12, (2003) 379–404
7. Penberthy, J. S. and D. Weld: UCPOP: A Sound, Complete, Partial-Order Planner for ADL. Third International Conference on Knowledge Representation and Reasoning (KR-92), Cambridge, MA, (1992)
8. Reiter, R.: Knowledge in Action. MIT Press (2001)
9. Russell S. Norvig .P: Artificial Intelligence: A Modern Approach. Prentice Hall, (1995)
10. Sirin, E., B. Parsia, D. Wu, J. Hendler, and D. Nau: HTN Planning for Web Service Composition Using SHOP2. J. Web Semantics, **1**, Elsevier (2004) 377–396
11. Sirin, E. and B. Parsia: Planning for Semantic Web Services. In Semantic Web Services Workshop at 3rd International Semantic Web Conference, (2004)
12. Sohrabi, S., Prokoshyna, N., McIlraith, S.A.: Web Service Composition via Generic Procedures and Customizing User Preferences. Int. Semantic Web Conf., ISWC2006 (2006) 597–611

OWL-Q for Semantic QoS-based Web Service Description and Discovery

Kyriakos Kritikos and Dimitris Plexousakis

Foundation of Research and Technology, Heraklion, Greece,
kritikos,dp@ics.forth.gr

Abstract. Semantic Web Services are emerging for their promise to produce a more accurate and precise Web Service discovery process. However, most of research approaches focus only on the functional part of semantic Web Service description. The above fact along with the proliferation of Web Services is highly probable to lead to a situation where Web Service registries will return many functionally-equivalent Web Service advertisements for each user request. This problem can be solved with the semantic description of QoS for Web Services. QoS is a set of non-functional properties encompassing performance and network-related characteristics of resources. So it can be used for distinguishing between functionally-equivalent Web Services. Current research approaches for QoS-based Web Service description are either syntactic or poor or non-extensible. To solve this problem, we have developed a rich and extensible ontological specification called OWL-Q for semantic QoS-based Web Service description. We analyze all OWL-Q parts and reason that rules should be added in order to support property inferencing and constraint enforcement. Finally, we line out our under-development semantic framework for QoS-based Web Service description and discovery.

1 Introduction

The success of the Web Service (WS) paradigm has led to a proliferation of available WSs. Current WS standard technologies involve the advertisement of static functional descriptions of WSs in UDDI registries, leading to a WS discovery process that returns many irrelevant or incomplete results. While semantic functional discovery approaches, like the one in [1], have been invented to overcome the above problem, the amount of functionally equivalent WS advertisements returned is still large. The solution to this problem is: a) the description of the Quality of Service (QoS) aspect of WSs, which is directly related to their performance; b) filtering of WS functional discovery results based on user constraints on their QoS descriptions; c) sorting the results based on user weights on QoS metrics.

QoS of a WS is a set of non-functional attributes that may impact the quality of the service offered by the WS. Each QoS attribute is measured by one or more QoS metrics, which specify the measurement method, schedule, unit, value range and other measurement details. A QoS specification of a WS is materialized as a set of constraints on a certain set of QoS metrics. These constraints restrict the metrics to have values in a certain range or in a certain enumeration of values. Actually, the current modeling efforts of QoS specifications only differ in the expressiveness of these constraints. However, these efforts fail in QoS metric modeling. The main reason is that their QoS metric model is syntactic, poor and not extensible. In this way, the most prominent QoS-based WS discovery algorithms produce irrelevant or incomplete results.

There are two main approaches for QoS-based Web Service Discovery. The first one, analyzed in [2], relies on the subsumption of the compared QoS-based WS descriptions for matchmaking. However, as indicated by the authors of this approach, subsumption is quite slow and additional techniques must be devised for speeding it up. The other approach, analyzed in [3], transforms the compared QoS-based WS descriptions to a Constraint Satisfaction Problem (CSP) [4] and then solves this problem. This approach has been shown [3] to be quick and efficient in realistic scenarios. In addition, tools for CSP solving are more mature than reasoning tools. Thus, the second approach is more appropriate for QoS-based WS discovery. Unfortunately, this approach also suffers from some shortcomings that will be analyzed in detail in the sequel of this paper.

Based on the above deficiencies, we have developed OWL-Q [5], a rich, extensible and modular ontology language that complements the WS functional description language OWL-S. In addition, we have extended the most prominent CSP-based QoS-based WS discovery approach [3]. In this paper, after reviewing the state-of-the-art in QoS-based WS description and discovery, we analyze in detail all parts of OWL-Q, as OWL-Q's design has been finalized. Next, we explain that OWL cannot be used for reasoning about relations between properties and for enforcing constraints so as to justify the extension of OWL-Q with SWRL rules. In addition, we provide examples of types of rules that have been added to OWL-Q. Then, we analyze our QoS metric matching and alignment and CSP-based WS Discovery algorithms [5, 6] and we shortly describe the building tools of our QoS-based WS Discovery Engine, which is currently under development. Finally, we conclude by drawing directions for further research.

2 Related Work

The *WSDL* and *UDDI* WS standards are *syntactical* approaches that do not express the QoS aspect/part of WS Description. While *OWL-S* is a standard *semantic* approach for WS Description, it does not describe any QoS concept.

Ran [7] proposes a syntactic extension to UDDI for QoS-based WS description. Maximilien and Singh [8] present an architecture and a conceptual model of WS reputation that does not include concepts like QoS constraints, offers and demands. Furthermore, the QoS metrics model is not rich enough. Tosic, Pagurek et. al. [9] present the XML-based *Web Service Offerings Language* (WSOL). Their work comes with the following shortcomings: (a) no specification of a QoS demand; (b) metrics ontologies are not developed. *Web Service Level Agreement* (WSLA) [10] is a XML language used for the specification of Service Level

Agreements (SLAs). It represents a purely syntactic approach that is not accompanied by a complete framework. Tian et. al. [11] propose an ontology-based approach for QoS-based WS description. However, not only there is no complete and accurate description of QoS constraints, but also metrics ontologies are only referenced. Oldham et. al. [12] offer a semantic framework for the definition and matching of *WS-Agreements*. However, only unary QoS metric constraints can be expressed while QoS metric matching could only be enforced by manual incorporation of rules.

Zhou et. al. [2] extend OWL-S by including a QoS specification ontology. In addition, they propose a novel matchmaking algorithm, which is based on the concept of *QoS profile compatibility*. The deficiencies of this research effort are the following: (a) The metrics model is not rich enough; (b) QoS metrics have $\mathbb{N}^+$ as their range; (c) QoS Profile subsumption reasoning is quite slow.

Martín-Díaz et. al. [3] use a symmetric but syntactic QoS model and propose a CSP-based approach for discovery. Before matchmaking, a QoS specification is transformed to a CSP which is checked for *consistency/satisfiability*. Matchmaking is performed according to the concept of *conformance*. Concerning WS Selection, the (QoS) score of an offer is computed by a *Constraint Satisfaction Optimization Problem* (CSOP) [4].

3 QoS-based Web Service Description

3.1 Requirements for QoS-based Web Service Description

After reviewing related work in QoS-based WS Description, we have come up with the following requirements that must be satisfied by a QoS-based WS description language [13]:

- *Devise an extensible and formal semantic QoS model*
- *Comply with standards*
- *Support the syntactical separation of QoS-based and functional parts of service specification*
- *Support refinement of QoS specifications and their constructs.*
- *Allow both provider and requester QoS specification*
- *Allow fine-grained QoS specification*
- *Devise an extensible and formal QoS metrics model*
- *Devise a corresponding extensible and formal QoS attributes, units, functions and measurement directives model.*
- *Allow classes of service specification*
- *Enabling of tractable matchmaking algorithms*

3.2 OWL-Q

Based on the requirements of QoS-based WS Description we have set in the previous subsection, we have developed an OWL-S extension (*syntactical separation*), called OWL-Q [5], for QoS-based WS description of both requests and offers.

We have extended OWL-S ontological description for two reasons: to comply with Semantic WS description standards (*standards compliance*) and to use the OWL ontology formalism (*extensible and formal semantic QoS model*). OWL-Q is actually an upper ontology comprised of many sub-ontologies/facets, each of which can be extended independently of the others (*syntactical separation and refinement of QoS specifications*). Each facet concentrates on a particular part of our QoS WS description. In its new form, OWL-Q has eleven facets: OWL-Q (main), Measurement Directive, Time, Goal, Function, Measurement, Metric, Scale, QoSSpec, Unit and ValueType. In the sequel, a small analysis of each facet of OWL-Q will be provided while the most important changes with respect to its previous form will be indicated. The whole ontology will be available soon at: http://www.csd.uoc.gr/∼kritikos/OWL-Q.owl.

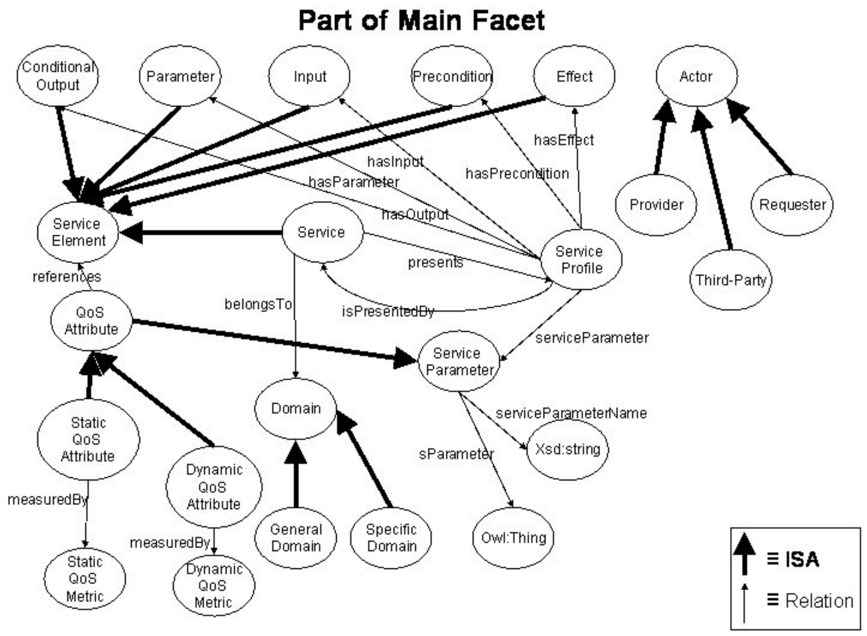


Fig. 1. Part of Main Facet.

OWL-Q (Main) Facet As can be seen in Fig. 1, the Main Facet connects OWL-S with OWL-Q and provides the high-level QoS concepts. For the connection of the two ontological descriptions, the *ServiceAttribute* class is a subclass of OWL-S *ServiceParameter* and references a *ServiceElement*. Subclasses of the latter class are *ConditionalOutput*, *Parameter*, *Input*, *Precondition*, *Effect*, and *Service*. That is a *ServiceAttribute* can reference any *ServiceElement* of a service's functional description (*fine-grained QoS specification*). Another point of connection is that a *ServiceProfile* contains one or more *QoSOffers*'s or one QoS Request (*classes of service requirement*). A final point of Connection is that the *Actor* class is separated into three subclasses: *Provider*, *Requester*, *ThirdParty* so as to define the main actors involved in QoS-based WS description and measurement. A Service Attribute contains two subclasses: QoSAttribute and ContextAttribute and is a subclass of the general class *Attribute*. An attribute can be separated into **a)** physical or service attributes, **b)** measurable or unmeasurable attributes and **c)** unique or derived attributes. Physical attributes like *Time*, *Temperature* and *Location* characterize environmental (contextual) factors of a WS or its requester while service attributes like *Availability* or *NumOfInterfaces* are functional or non-functional characteristics of a WS. Measurable attributes like *Time* are measured by specific metrics while unmeasurable attributes like *Manageability* cannot be measured. Unique attributes like *Time* are not derived by other attributes and are measured by resource metrics while derived attributes like *Throughput* are produced by complex metrics computed by functions using metrics of other attributes. The *Domain* class represents the domain of knowledge that a service applies to and is separated into two subclasses: **a)** *GeneralDomain* and **b)** *SpecificDomain*. The *GeneralDomain* stands for every possible WS. Specific Domain can be further specialized/subsumed, for example a possible subclass could be the Travel domain. The *Value* class represents any possible integer, double, string or list-based value, it is subsumed by special symbol classes like Infinity or Limit, and it is mainly used in Goal, Value Type and Measurement definitions. Other classes that are defined but are unfolded in separate sub-ontologies are: *Function*, *Measurement*, *MeasurementDirective*, *Metric*, *QoSSpec*, *Scale*, *Schedule*, *Trigger*, *Unit*, *ValueType* and *Goal*.

QoSSpec Facet In this facet, the classes representing QoS offers and requests are defined. The class *QoSSpec* is separated into two subclasses: *QoSOffer* and *QoSDemand* in order to enable WS providers and requesters to define in the same way their QoS constraints *(both provider and requester QoS specification)*. Of course, the WS requester is enabled not only to specify constraints (by the *QoSDemand* class) but also to provide weights to metrics of his interest (by the *QoSSelection* class). The *QoSSelection* class is actually a list of <metric, weight> entries. The *QoSSpec* class represents the actual QoS description of a WS. It describes the security and transaction protocols used (URIs), the cost of using the service (double) and the associated currency for the cost (unit), the validity period of the offer or demand (*CalendarClockInterval* class of the time ontology [14]) and a list of conjunctive QoS goals/constraints with their weights (value of 2.0 if it is a hard constraint or a value in $(0.0, 1.0)$ if it is soft).

Goal Facet Mathematical formulas and QoS goals/constraints were previously expressed in OpenMath (http://www.openmath.org). Now this has changed due to change of philosophy regarding the QoS Metric Matching Algorithm described in the next section. QoS constraints are expressed in the following form: $(f(arguments)|metric)$ *op value*, where f is a function, arguments are a list of functions, metrics and values, op can be one of $\leq$, $\geq$, $<$, $>$, $=$, $!=$. For example, the fact that metric M is less than 0.1 could be expressed by the user as: $M \leq 0.1$, where op $= \leq$. An appropriate interface will be provided to the user in order to enable him to specify constraints in our user-friendly customized expression form.

Measurement Facet Measurements are now modeled in OWL-Q so as to enable their storage and statistical processing by registries or other parties. Statistical processing leads to new metric derivation and to validation of QoS-based WS provider guarantees. The Measurement class in OWL-Q contains a single value (*Value* class), it is produced by an *Actor* at a specific time point (*Calendar-ClockDescription* class of [14]), it concerns a specific *Metric* and belongs to a specific party (*Actor*).

Function Facet Functions in OWL-Q are separated into functions applied to metrics (for producing complex metrics or checking satisfiability of their constraints) and functions applied to scales. Metric functions have specific arity and contain arguments that are either Metrics, values of other Metric Functions. Scale transformation functions are further categorized into five disjoint subclasses and are used for converting one scale expression to an expression of another scale.

Measurement Directive Facet The *MeasurementDirective* class specifies the way simple metrics are measured. It specifies by a URI how the value of a managed resource is going to be achieved and by a *ValueType* the type of the return value. In addition, it specifies if the party responsible for the measurement will ask for the value or get it when it is ready (i.e it specifies the access model, where $AccessModel = Pull \cup Push$). This class can have many subclasses, some of which may require a possible extra attribute (*timeOut*) specification concerning the time duration (*DurationDescription* of [14]) that the measurement party will wait to get the measurement value (consider for example the *Status* measurement directive [10]).

Time Facet This facet specifies the *Schedule* and *Trigger* classes. A schedule is used to describe the frequency (*frequency* has range *DurationDescription*) and time interval (*interval* has range *CalendarClockInterval* [14]) of a complex metric computation. Alternatively, a complex metric computation can be executed at a specific time point (*CalendarClockDescription* [14]), information that is encapsulated in a trigger definition.

Metric Facet The Metric Facet describes all the appropriate classes and properties used for a proper formal definition of a QoS metric (*QoS metric model*). This metric facet is actually an upper ontology representing any abstract QoS metric. A specific QoS metric can be created by refining the *QoSMetric* class. Many specific QoS metrics (especially the general ones) can be part of a midlevel ontology created for QoS metric reuse. We prefer specialization to instantiation because it allows for a quicker reasoning process. We plan to develop a mid-level ontology defining cross-domain QoS metrics and a low-level ontology for defining QoS metrics for particular domains.

The *QoSMetric* is one of the most important classes of OWL-Q representing a QoS metric. The values of a QoS metric are provided by an *Actor*. A QoS metric belongs to a *Domain* of knowledge. It has only one name. It measures a $QoSAttribute \cup MeasurableAttribute$ on a specific *ServiceElement*. The value type of a *QoSMetric* is an instance of the *ValueType* class (analyzed in a separate facet) while the scale of the value is an instance of the *Scale* class. A *QoSMetric* is separated into static and dynamic metrics. A *StaticQoSMetric* is computed only once according to a *Trigger* in order to produce a value for a *StaticQoSAttribute*. A *DynamicQoSMetric* is computed repeatedly according to a *Schedule* to produce values of a *DynamicQoSAttribute* that change over time. It can be a simple QoS metric *measuredBy* a *MeasurementDirective* or a complex one. *ComplexMetric*s are derived from other metrics with the help of a *MetricFunction*. Last but not least a *QoSMetric* is related to other metrics according to two types of *Relationship*s: *Independent* and *Related*. When two metrics are related, we can specify the direction of their values or the impact of one's value to the other's value. According to the scale it uses, a metric can be categorized into absolute, interval, nominal, ordinal and ratio metrics. Ratio metrics directly reference a Unit of measurement as a *RatioScale* is actually equivalent to a *Unit*. Metrics can be positively or negatively monotonic. In this way, we know if one metric value is better than another one.

Scale Facet A measurement scale controls the value type and the type of operations allowed for a metric and belongs to a specific *Attribute*. It also specifies the way one value expression bound to one scale can be transformed to another value expression of another compatible scale (both scales belonging to the same metric). So specific scales can be compatible if they belong to the same scale type and there is a *ScaleTransformationFunction* that transforms their expressions into each other. *Scale* is a more general notion with respect to *Unit*. A scale can be categorized into five disjoint subclasses: *NominalScale*, *OrdinalScale*, *IntervalScale*, *RatioScale* and *AbsoluteScale* [15]. Nominal scales concern metrics that have as value type a set of numbers or strings. The members of this set cannot be compared (no ordering). Specific nominal scales can be compatible if there is a one-to-one mapping function between their corresponding value types. Ordinal scales apply to metrics that have an ordered set as value type. Metrics belonging to different ordinal scales cannot be added, multiplied, divided or abstracted in QoS constraints. We can transform one ordinal scale expression into another one with the help of monotonic functions. Interval scales preserve not only ordering but also differences. However, they do not preserve ratios. The operations of addition and substraction are allowed between different ordinal metrics. We can transform one interval scale expression into another one with the help of affine transformation functions of the form: $M = a * M^{'} + b$. Ratio scale preserve ordering, size of intervals and ratios. In a ratio scale there is always a zero element representing the total lack of the measured attribute. All arithmetics are allowed between different ratio metrics. We can transform one ratio scale expression into another one with the help of mapping functions of the form: $M = a * M'$. Finally, the following facts are true for an absolute scale: **a)** measurement is made simply by counting the number of elements in the measurement set; **b)** measured attribute takes the form: "num of occurrences of x in the entity"; **c)** all arithmetic analysis is meaningful; d) the set of acceptable transformations between different absolute scale expressions is the identity transformation function.

Unit Facet The Unit Facet formally describes the unit of a ratio scale of a ratio QoS metric. A *Unit* has one name, several abbreviations and synonyms (even in different languages). A *Unit* belongs to a *System of Units*, which system can be *SelfConsistent* or *NonSelfConsistent*, and is associated with the same *QoSAttribute* as the one that is measured by the QoS metric of the unit. A *Unit* is separated into *BasicUnit*s and *MultipleUnit*s. The *BasicUnit* class is separated into *UniqueAttributeUnit*s and *DerivedAttributeUnit*s, depending on the type of *Attribute* measured. A *MultipleUnit* is associated with a *BaseUnit* and converted to it by a constant (*magnitude*). It has a name composed of the name of its *BaseUnit* and a prefix. A *DerivedUnit* is proportional to some *Unit*s and inverse proportional to other *Unit*s. It also has a *magnitude* that is used to express its mathematical definition in relation to the other (inverse) proportional units. An unit is equivalent to another unit and can be converted to it with the help of their ratio scale and its ratio transformation functions.

Value Type Facet The *ValueType* ontology describes the types of values a QoS metric can take. The *ValueType*s can be *Scalar* or *NumericUnion*, or *ListBased* types. *Scalar* value types are simple value types that can be *Numeric* or *String. ConstrainedNumeric* value types represent *Numeric* value types that have (upper, low or one) limits (e.g. the Integers set [2,5] or the Integer value {2}). The *NumericUnion* class represents value types that are expressed as unions of *Numeric* value types (e.g. $[1, 2] \cup \{4\} \cup [9, 11]$). The *List-Based* class represents list value types that have a specific size and whose elements are of a specific *ValueType.* Subclasses of the *ListBased* class are: numeric or string lists, queues and timeseries.

3.3 Rules

The most significant change in OWL-Q is the incorporation of rules. It is well-known at the Semantic Web community that OWL supports very well reasoning about concepts but not about properties. For example, there is no way we can specify that a fact $p(x, y)$ can be true, where x, y are instances, if other property or instance facts are true. As another example, there is no way to specify that two or more property or class instance facts (or a mixture of them) cannot be both part of the semantic database. However, it is imperative in OWL-Q to reason about properties with rules because: **a)** relations between temporal properties like duration [14] should be expressed and reasoned about; **b)** operations or comparisons on metrics should be restricted according to the scale that they use;

c) integrity constraints between property facts and/or instance facts should be able to be enforced; **d)** compatibility or equivalency of scales and compatibility of metrics' value types should be expressed by OWL property facts fired by rules; **e)** rule-based algorithms like the metric matching one (see next section) have to be specified. So we are currently in the process of extending OWL-Q with rules, which are expressed in SWRL – the most widely used SW rules proposal at present. However, most reasoners only partially support SWRL and this is a major obstacle to our under-implementation semantic framework for QoS-based WS description and discovery.

4 QoS-based Web Service Discovery Framework

4.1 QoS Metric Matching Algorithm

All QoS-based WS discovery algorithms fail to produce accurate results because they rely on either syntactic or semantically-poor QoS metric descriptions. Hence, they cannot infer the equivalence of two QoS metrics based on descriptions provided by different parties. Different specifications occur for two reasons: **a)** different perception of the same concept; **b)** different type of system reading for the same metric. For example, equivalent response time metrics could be associated to different units (e.g. minutes vs. seconds) and to different value types(e.g. [0.0,10.0] vs. [0,600] respectively). As another example, a DownTime metric can be either obtained in the form of high-level reading from a system with advanced instrumentation or can be derived from a resource metric of a system's Status obtained from low-level reading of systems with basic instrumentation.

Provided that two QoS metric descriptions are expressed in OWL-Q, we have developed a rule-based QoS metric matching algorithm [5] that infers the equivalence of the two metrics. This algorithm is composed of three main rules, each corresponding to a different case in a two metrics comparison. The last rule is recursive and reaches the final point of checking the equivalence of two mathematical formulas in order to infer the equivalence of two metrics.

Unfortunately, equivalency of mathematical expressions, which is a problem area of symbolic computation, is undecidable. For this reason, we decided to use CSP solving that is decidable although computationally expensive. The trick for this transformation/change is the simple observation that symbolic expression equality can be seen alternatively as unsatisfiability of a CSP containing a constraint enforcing that the difference of the two expressions is not zero. In other words, if the CSP does not have any solution, then the constraint cannot be enforced and the negation of its formula is always true. The latter infers the equality of the expressions compared, which is our goal. For example, suppose that we want to check if two expressions $(x+1)^2$ and x^2+2x+1 are equal. We can easily transform the previous problem to a CSP: $[(X : -\infty \ldots +\infty), ((x+1)^2 - x^2 - 2x - 1 != 0)]$ and try to solve it. There is no solution to this CSP, so the constraint is unsatisfiable, the difference of the two expressions is always zero and thus these expressions are equal.

Due both to changes on the OWL-Q Ontology and to the above reasoning, we have modified our metric matching algorithm as follows:

$$match\,(M_1, M_2) \Leftarrow rrm\,(M_1, M_2) \vee rcm\,(M_1, M_2) \vee ccm\,(M_1, M_2) \tag{1}$$

$$\begin{aligned} &sm\,(M_1, M_2) \Leftarrow svm\,(M_1.scale, M_2.scale, M_1.type, M_2.type) \\ &\wedge M_1.object = M_2.object \wedge M_1.measures = M_2.measures \end{aligned} \tag{2}$$

$$rrm\,(M_1, M_2) \Leftarrow ResourceMetric\,(M_1) \wedge ResourceMetric\,(M_2) \wedge sm\,(M_1, M_2) \tag{3}$$

$$\begin{aligned} &rcm\,(M_1, M_2) \Leftarrow ResourceMetric\,(M_1) \wedge CompositeMetric\,(M_2) \wedge sm\,(M_1, M_2) \\ &\wedge M_2.derivedFrom \cap CompositeMetric = \oslash \wedge \neg\exists\, V \in M_2.derivedFrom\ match\,(M_1, V) \end{aligned} \tag{4}$$

$$\begin{aligned} &ccm\,(M_1, M_2) \Leftarrow CompositeMetric\,(M_1) \wedge CompositeMetric\,(M_2) \\ &\wedge sm\,(M_1, M_2) \wedge msm\,(M_1.derivedFrom, M_2.derivedFrom) \\ &\wedge \neg solveCSP(M_1.derivedFrom, M_2.derivedFrom, \\ &M_1.measuredBy - M_2.measuredBy! = 0) \end{aligned} \tag{5}$$

where M_1 and M_2 are Metrics, $svm\,(M_1.scale, M_2.scale, M_1.type, M_2.type)$ is a rule that infers if the scales and value types of metrics M_1 and M_2 are compatible, $msm\,(M_1.derivedFrom, M_2.derivedFrom)$ is a rule that matches one by one the M_1's list of derivative metrics with the corresponding metrics list of M_2, and $solveCSP\,(List_1, List_2, equation)$ is a logic procedure that solves the CSP defined by the two first metric lists and the equation given by third argument. When the latter procedure finds a solution, it returns true, otherwise it returns false. More details about all other clauses and symbols can be found in [6].

Therefore, the above algorithm infers that two metrics M_1 and M_2 match if one of the three body rules of rule (1) is satisfied. The first (rule (3)) and the second (rule (4)) of the three body rules have not been altered and we are not going to further describe them.

The last of the three body rules, rule (5), compares and possibly aligns one by one the metrics from which M_1 is derived with the corresponding metrics of M_2 and updates appropriately the measurement formulas of M_1 and M_2. Then from the derivation lists of M_1 and M_2 and their measurement formulas a (possibly non-linear) CSP is defined and solved. More details about the algorithm can be found in [6].

Composite-to-Composite Metric Matching Example. Assume that a WS provider defines composite metric $Avail_1$ that measures the QoS Property of *Availability* of his whole WS and is derived from two Resource metrics $Downtime_1$ and $Uptime_1$ based on the formula: $1 - Downtime_1/(Downtime_1 + Uptime_1)$. In addition, assume that a WS requester defines composite metric $Avail_2$ that also measures the QoS Property of *Availability* and is derived from two Composite metrics $Downtime_2$ and $Uptime_2$ based on the formula: $Uptime_2/(Uptime_2 + Downtime_2)$. Further assume that all metrics have as value type the interval $[0.0, 1.0]$ and that the following facts are true: $smatch(M_1, M_2)$, $rcm(Downtime_1, Downtime_2)$, $rcm(Uptime_1, Uptime_2)$. We want to see if composite metrics $Avail_1$ and $Avail_2$ are matched based on the satisfiability of rule (5). The first three clauses of this rule are trivially true. The fourth clause infers that: $rcm(Downtime_1, Downtime_2)$, $rcm(Uptime_1, Uptime_2)$. So $Downtime_1$ and $Downtime_2$ are mapped to a new metric $Downtime$ and $Uptime_1$ and $Uptime_2$ are mapped to $Uptime$. In this way, it stands that: $M_1.derivedFrom = M_2.derivedFrom = [Downtime, Uptime]$, $M_1.measuredBy = 1 - Downtime/(Downtime + Uptime)$, $M_2.measuredBy = Uptime/(Uptime + Downtime)$. The last clause of the rule will create and solve a CSP that has the following definitions: $Downtime, Uptime :: [0.0, 1.0]$ and constraints: $1 - Downtime/(Downtime + Uptime) - Uptime/(Uptime + Downtime)! = 0$. This CSP is unsatisfiable so finally the fact $match\,(M_1, M_2)$ is inferred.

4.2 QoS Metric Alignment Algorithm

The Alignment process is executed when any QoS specification S is published or queried on the underlying QoS-based WS discovery system. Its goal is to align S with all already processed offers O_i and demands D_j by finding their common QoS metrics based on the QoS metric matching algorithm. After metric alignment, S is transformed to a CSP which is checked for consistency (i.e. if it has a solution). If the CSP is inconsistent, then neither S nor its CSP are stored in our *Repository* (R) and S's owner is informed. In case of an inconsistent demand, the discovery algorithm is also not executed. The alignment process relies on the concept of the *Metric Store* (MS), which is part of R. MS stores all unique QoS metrics encountered so far. So when a new QoS spec arrives, we don't need to examine if any of its metrics matches with any metric of all offers or demands but with any metric in the MS. In this way, there is a minimization of all possible metric-to-metric comparisons. In addition, all unique metrics of this new QoS spec are added to the MS. If this QoS spec is inconsistent, its metrics are not removed from the MS. More details about this algorithm and how the transformation of a QoS spec S to a CSP is carried out can be found in [6].

4.3 QoS-based Web Service Discovery Algorithm

One of the most prominent QoS-based WS discovery algorithm [3] expresses each QoS-based WS description as a CSP. Then it separates the QoS-based advertisements into two categories: the ones that satisfy completely the QoS-based request and the others that do not satisfy the request. However, this algorithm presents four major drawbacks: **1)** it performs syntactic metric matchmaking producing false negative and false positive results; **2)** QoS spec matchmaking relies on the concept of *conformance*, which is not absolutely correct (see next paragraph); **3)** it does not provide advanced categorization of results; **4)** it does not return any result when QoS requests are over-constrained, where over-constrained problem specifications happen very often in the real-world.

Matchmaking of QoS offers and demands is based on the concept of conformance [3], which is mathematically expressed by the following equivalency:

$$conformance\left(O_i, D\right) \Leftrightarrow sat\left(P_i \wedge \neg P^D\right) = \text{false} \qquad (6)$$

To explain, an offer O_i matches a demand D when there is no solution to the offer's CSP P_i that is not part of the solution set of the demand's CSP P^D. This definition is slightly wrong as it excludes from the result set those QoS offers that provide better solutions than that of the demand's. For example, suppose that a WS provider and requester use the same metric X, measuring the QoS Property of Availability, that has as value type the set $(0.0, 1.0) \uparrow$, where $\uparrow$ denotes that this type is positively monotonic i.e. greater values are better than lower ones. Further assume that the WS provider's CSP has the constraint: $X \geq 0.96$ while the WS requester's CSP has the constraint: $0.95 \leq X \leq 0.999$. Based on the above definition, the provider's offer does not match the request as it contains solutions greater than that of the request's, although these solutions are better. Thus, a more correct definition of matchmaking is the following: an offer O_i matches a demand D when its CSP P_i has solutions that are either contained in the solution set of the demand's CSP P^D or are better that the demand's solutions.

Based on the deficiencies of [3] and the new definition of matchmaking, we have proposed two QoS-based WS discovery algorithms [6]. The first one is only restricted to unary constraints but is more effective and easy to implement while the other is more generic but harder to implement. These algorithms presuppose that the offers set $\{O_i\}$ and the demand D are already aligned and transformed to corresponding CSPs P_i and P^D respectively. Due to space limitations of this paper, we are going to analyze only the second algorithm.

Generic Discovery Algorithm This algorithm checks if the whole solution of the offer is worse than all solutions of the demand by assigning a preference or value to each CSP solution. So it is more closed to the definition of conformance we have previously given in this section.

The big question is how the assignment of preferences to solutions takes place. The technique we use is based on utility functions and weights on CSP variables [3]. Each CSP variable (a map of a metric) is given a (user) weight or preference (taking values from the set $[0.0, 1.0]$) to reflect the significance of this variable to the preference/value of the solution. In addition, each possible value of this variable is given also a preference ($\in [0.0, 1.0]$) by the variable's utility function. The preference of a CSP solution is given by the following sum on all variables X_j: $p_s = \sum_{X_j}(w_{X_j} \cdot uf_{X_j}(v_{X_j}))$, where w_{X_j} is the weight of the variable X_j, $uf_{X_j}()$ is its utility function and v_{X_j} is its value.

Based on the above technique, a partial ordering of all solutions of a CSP can be inferred. This is the appropriate mean in order to define matchmaking: an offer's CSP P_i matches the CSP P^D of the demand if its worst solution has a preference of greater or equal value with respect to the preference of the worst solution of the demand. This definition leads to two main observations: **a)** CSOPs for offers and demands have to be solved in order to find the preference of the worst solution; **b)** constraints are only used to reduce the domain of the variables. The second observation hides an important conclusion: constraint relaxation is inherent to the optimization of CSPs based on preference functions. To explain, a matching offer may have a (worst) solution that violates constraints of the demand affecting one or more variables of less significance. However, this solution surely provides better values for variables of higher significance/preference. It is like relaxing some constraints of the demand in order to match this offer. The next paragraph provides a sketch of the QoS-based WS discovery algorithm, while the last one provides a simple example of its application.

Algorithm. [**Matchmaking**] We compute the preferences $p^D_{s_1}$ and $p^D_{s_2}$ of the demand's CSP P^D worst s^D_1 and best s^D_2 solution respectively by solving two CSOPs (minimization and maximization) [5]. For each offer's CSP P_i, we compute the preferences $p^i_{s_1}$ and $p^i_{s_2}$ of its worst s^i_1 and best s^i_2 solution respectively in the same manner as above. Then, we consider four cases:

1. If $(p^i_{s_2} \leq p^D_{s_1})$, then the offer is put in the *fail* match list.
2. If $(p^i_{s_2} > p^D_{s_1} \wedge p^i_{s_1} < p^D_{s_1})$, then the offer is put in the *partial* match list.
3. If $(p^i_{s_1} \geq p^D_{s_1} \wedge p^i_{s_2} \leq p^D_{s_2})$, then the offer is put in the *exact* match list.
4. If $((p^i_{s_1} \geq p^D_{s_1} \wedge p^i_{s_2} > p^D_{s_2}) \vee (p^i_{s_1} \geq p^D_{s_2}))$, then the offer is put in the *super* match list.

The first case expresses the fact that the offer's best solution is not better than the worst solution of the demand and justifies the classification of the offer as *failed*. The second case expresses the fact that the offer has some *bad* solutions but also some *good* solutions so it is considered as a *partial* result. The third case concerns offers that contain a subset of the solutions of the demand and justifies their classification as *exact*. The last case is about offers that contain not only solutions of the demand but also better ones. That's why they are classified as *super* results/matches.

[**Selection**] In this process, either the best two categories of results (if not empty) or the third category are ordered based on the weighted sum of the preferences of their worst and best solutions [5].

Example. To demonstrate our QoS-based WS discovery algorithm, we supply a simple example of its application to a small set of four QoS offer CSPs P_i and one demand CSP P^D. Assume that all CSPs have the following three definitions: $X_1 :: (0.0, 86400.0] \downarrow$, $X_2 :: (0, 100000] \uparrow$ and $X_3 :: (0.0, 1.0) \uparrow$. Based on these variable definitions, assume that each CSP has the following constraints: P_1 : $[X_1 \leq 10.0, X_2 \leq 100, X_2 \geq 50, X_3 \geq 0.9]$, P_2 : $[X_1 \leq 4.8, X_2 \leq 50, X_2 \geq 40, X_3 \geq 0.95]$, P_3 : $[X_1 \leq 16, X_2 \leq 40, X_2 \geq 30, X_3 \geq 0.98]$, P_4 : $[X_1 \leq 16, X_2 \leq 50, X_2 \geq 40, X_3 \geq 0.98]$, and P^D : $[X_1 \leq 15.0, X_2 \geq 40, X_2 \leq 60, X_3 \geq 0.99]$. Moreover, assume that the WS requester does not provide weights to the constraints of his demand and associates the following weights to the three metrics/variables: $X_1 \leftarrow 0.3$, $X_2 \leftarrow 0.3$, $X_3 \leftarrow 0.4$, while $a = 0.7$ and $b = 0.3$ [5].

In addition, assume that the following utility functions are applied to the CSOPs: $uf_{X_1} = (16 - X_1)/16$, $uf_{X_2} = (X_2 - 30)/70$, $uf_{X_3} = (X_3 - 0.9)/0.1$ [5].

For each offer CSP P_i we have the following preferences: $[P_1 : p^1_{s_1} = 0.1982, p^1_{s_2} = 1.0]$,$[P_2 : p^2_{s_1} = 0.4528, p^2_{s_2} = 0.7857]$, $[P_3 : p^3_{s_1} = 0.32, p^3_{s_2} = 0.7428]$, $[P_4 : p^4_{s_1} = 0.3628, p^4_{s_2} = 0.7428]$. The demand's CSP P^D has the following preferences: $P^D : [p^D_{s_1} = 0.4216, p^D_{s_2} = 0.8285]$. So the discovery algorithm will produce the following results lists: $Super = [\,]$, $Exact = [O_2]$, $Partial = [(O_1), (O_3), (O_4)]$, $Fail = [\,]$.

As it can be seen, offer O_2 is in the *Exact* match list although it violates the last constraint of the demand. The reason for this is that the preference of its worse solution is greater than the preference of the worse solution of the demand. To put it in another way, O_2 provides a far better lowest value for the X_1 attribute with respect to the worse lowest value for the X_3 attribute. Another observation is that O_1 pays the penalty of providing the minimum possible value for the X_3 attribute and is considered a *partial* result.

4.4 QoS-based Web Service Discovery Engine

We are currently in the development phase of our QoS-based WS discovery engine by using the Pellet reasoner for ontology reasoning and the ECLiPSe (http://eclipse.crosscoreop.com) system for solving linear constraints, while the Java programming language is used as a bridge between them. Pellet is chosen because it supports the tasks of ontology validation and reasoning, OWL 1.1 datatype reasoning and partial SWRL inferencing. ECLiPSe is chosen as it supports advanced linear constraint solving and extends the common facilities of Prolog. Additionally, it can be extended to support non-linear constraint solving through external solvers. More details about the architecture and the functionality of the main components of the discovery engine can be found in [6].

5 Future Work

As future work, we plan to evaluate our metric matching and discovery algorithms in order to show their performance and accuracy. We also intend to exploit advanced techniques for solving over-constrained problems like semi-ring based constraint satisfaction [16]. We also plan to extend OWL-Q with the description of the context of both the WS and the WS requester so as to achieve Context-aware QoS-based WS discovery. Our ultimate and final goal is to accomplish QoS-based and context-aware WS composition.

References

1. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with owls-mx. In: AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2006) 915–922

2. Zhou, C., Chia, L.T., Lee, B.S.: Daml-qos ontology for web services. In: ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04), Washington, DC, USA, IEEE Computer Society (2004) 472
3. Cortés, A.R., Martín-Díaz, O., Toro, A.D., Toro, M.: Improving the automatic procurement of web services using constraint programming. Int. J. Cooperative Inf. Syst. **14**(4) (2005) 439–468
4. Van Hentenryck, P., Saraswat, V.: Strategic directions in constraint programming. ACM Computing Surveys **28**(4) (1996) 701–726
5. Kritikos, K., Plexousakis, D.: Semantic qos metric matching. In: ECOWS '06: Proceedings of the European Conference on Web Services, Washington, DC, USA, IEEE Computer Society (2006) 265–274
6. Kritikos, K., Plexousakis, D.: Semantic qos-based web service discovery algorithms. In: ECOWS '07: Proceedings of the European Conference on Web Services, Washington, DC, USA, IEEE Computer Society (2007) (accepted).
7. Ran, S.: A model for web services discovery with qos. SIGecom Exch. **4**(1) (2003) 1–10
8. Maximilien, E.M., Singh, M.P.: Conceptual model of web service reputation. SIGMOD Rec. **31**(4) (2002) 36–41
9. Tosic, V., Pagurek, B., Patel, K.: Wsol - a language for the formal specification of classes of service for web services. In Zhang, L.J., ed.: ICWS, CSREA Press (2003) 375–381
10. Keller, A., Ludwig, H.: The wsla framework: Specifying and monitoring service level agreements for web services. Technical Report RC22456 (W0205-171), IBM (2002)
11. Tian, M., Gramm, A., Nabulsi, M., Ritter, H., Schiller, J., Voigt, T.: Qos integration in web services. Gesellschaft fur Informatik DWS 2003, Doktorandenworkshop Technologien und Anwendungen von XML (October 2003)
12. Oldham, N., Verma, K., Sheth, A., Hakimpour, F.: Semantic ws-agreement partner selection. In: WWW '06: Proceedings of the 15th international conference on World Wide Web, New York, NY, USA, ACM Press (2006) 697–706
13. Kritikos, K., Plexousakis, D.: Requirements for qos-based web service description and discovery. compsac **2** (2007) 467–472
14. Hobbs, J.R., Pan, F.: An ontology of time for the semantic web. ACM Trans. Asian Lang. Inf. Process. **3**(1) (2004) 66–85
15. Fenton, N.E.: Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, Boston, MA, USA (1996)
16. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. J. ACM **44**(2) (1997) 201–236

Optimizing the Retrieval of Pertinent Answers for NL Questions with the E-Librarian Service

[1]Serge Linckels, [2,1]Harald Sack, [1]Christoph Meinel

[1]Hasso Plattner Institute for Software Systems Engineering (HPI)
University of Potsdam, Postfach 900460, D-14440 Potsdam, Germany
{linckels,meinel}@hpi.uni-potsdam.de
[2]Institut für Informatik, Friedrich-Schiller-Universität Jena
Fürstengraben 1, D-07743 Jena, Germany
sack@minet.uni-jena.de

Abstract. There is a growing discrepancy between the creation of digital content and its actual employment and usefulness in a learning society. Technologies for recording lectures have become readily available and the sheer number and size of such objects produced grows exponentially. However, in practice most recordings are monolithic entities that cannot be integrated into an active learning process offhand. To overcome this problem, recorded lectures have to be semantically annotated to become full-fledged e-learning objects facilitating automated reasoning over their content. We present a running web-based system — the e-Librarian Service CHESt — that is able to match a user's question given in natural language to a selection of semantically pertinent learning objects based on an adapted best cover algorithm. We show with empirical data that the precision of our e-Librarian Service is much more efficient than traditional keyword-based information retrieval; it yields a correct answer in most of the cases (93% of the queries), and mostly with a high precision, i.e., without supplementary hits. We also describe some ideas to improve the retrieval performance by user feedback.

1 Introduction

The World Wide Web (WWW) is the largest knowledge base that ever existed. The availability of material with educational content in the WWW increases dramatically. However, its usage in an educational environment is poor, mainly due to two facts [16, 6, 18, 22]. First, there is currently no reliable mechanism to prove the correctness of the data. Second, there is way too much information, in particular redundant and not relevant information, so that finding appropriate answers in an efficient way is a rather difficult task being reliant on the user's interaction. The user is charged with the awkward, time consuming and diverting task of filtering the pertinent information out of the noise. Turning large knowledge bases as the WWW into useful educational resources requires to identify correct, reliable, and machine understandable resources, as well as to develop simple but highly efficient search tools with the ability to perform logical inferences over these resources. This idea is fully in the stream of current Semantic Web thinking.

In this paper we describe a running system[1] — the e-Librarian Service CHESt [12, 13] — that is able to understand a user's questions given in natural language (NL) and to retrieve semantically pertinent resources. We call such resources *Learning Objects* (LOs). By LO we refer to an entity about a precise subject that may be used for learning, education or training [20], e.g., a multimedia sequence including machine processable metadata that semantically describe its content. Our E-Librarian Service can be perceived as a specialization of passage retrieval techniques; see [14] for an overview.

It has been realized that digital libraries do benefit from having its content understandable and available in a machine processable form, and it is widely agreed that ontologies will play a key role in providing the infrastructure to achieve this goal. One of the basic building blocks of our e-Librarian Service is a common domain ontology, which has a double use. First, it is used for the translation of the NL user questions into a formal language, i.e., Description Logics (DLs). DLs are a family of knowledge representation formalisms that allow to represent the knowledge of an application domain in a structured way and to reason about this knowledge [1]. The semantic interpretation, i.e., the translation of a NL user question into a DL is described in [12]. Second, the domain ontology is used to describe the LOs in the knowledge base with additional semantic metadata. We developed a tool that helps to semi-automatically generate the semantic metadata based on the textual content of the LOs.

Our e-Librarian Service implements a retrieval algorithm that is based on the concept covering problem. Among all the LOs that have some common information with the user query, our algorithm is able to identify the most pertinent match(es), keeping in mind that the user in general expects an exhaustive answer while preferring a concise answer with only little or no information overhead. The evaluation of our algorithm shows that in an educational environment our e-Librarian Service is much more appropriate than a traditional keyword-based search engine, because it delivers much less information overhead while simultaneously providing a higher precision.

The paper is structured as follows. After this introduction, section 2 discusses related work and projects. The main contribution of the paper is the algorithm for retrieving semantically pertinent LOs from a given knowledge base. The algorithm is presented in section 3, and explicitly discussed and evaluated in section 4. Section 5 provides an outlook and discussion how the system can be improved by user contributions and feedback, while section 6 concludes the paper with a brief summary of achieved results.

2 Related Work

Instead of the traditional Question Answering (QA) as being subject in linguistics and information retrieval [17], our approach is not targeted to compute a

[1] `http://www.linckels.lu/chest`

coherent answer being expressed in NL. We simply provide a set of interrelated resources (LOs), which contain the information that is necessary to answer the user's question. The user has to read the provided LO(s) to obtain an answer.

We address three different approaches related to document matching and retrieval based on DL inferences. First, an approach for matching documents based on non-standard inferences in the DL sub-languages $\mathcal{ALNS}, \mathcal{ALN}^*$, and $\mathcal{ALE}$ is presented in [9]. A matching problem modulo equivalence and modulo subsumption is of the form $C \equiv^? D$ and $C \sqsubseteq^? D$ respectively, where C is a description and D a pattern. A solution or matcher of these problems is a substitution σ such that $C \equiv \sigma(D)$ and $C \sqsubseteq \sigma(D)$, respectively. The solution is based on computing homomorphisms between description trees. Although this is an excellent solution for dealing with complex descriptions such as for comparing complete documents, it is less appropriate for our purpose. In our case, LOs are described by simple semantic annotations with few role-imbrications. The resulting description trees are rather flat and comprise rarely more than two levels.

Second, the concept covering problem [7] is based on DLs with structural subsumption. The proposed algorithm for identifying the best cover relies on the computation of minimal transversals in a hypergraph. The algorithm has been implemented in the project MKBEEM (Multilingual Knowledge Based European Electronic Marketplace). That solution is very pertinent for our e-Librarian Service because it always finds the best cover, i.e., the best matching LOs w.r.t. the user's question (see section 3.2).

Another definition of the concept covering problem that eliminates the limitation of DLs to provide structural subsumption has been presented in [5]. There, the concept covering problem is based on the concept abduction problem (CAP) [19], which is able to provide an explanation if subsumption does not hold. It is stated as follows: S (supply) and D (demand) are two descriptions in a DL $\mathcal{L}$, and satisfiable in a terminology $\mathcal{T}$. A CAP, identified by $< \mathcal{L}, S, D, \mathcal{T} >$, is finding a concept $H \in \mathcal{L}$ (hypotheses) such that $\mathcal{T} \models S \sqcap H \sqsubseteq D$, and moreover $S \sqcap H \not\equiv \bot$. The algorithm was implemented in a project for semantic-based discovery of matches and negotiation spaces in an e-marketplace. One of the weaknesses of this solution is that does not always return an optimal cover.

We decided to base our e-Librarian Service on the concept covering problem as presented in [7] because for our application DLs with structural subsumption provide sufficient expressiveness. Furthermore, our system must always return an optimal cover. Finally, the solution is simple and adapted to our LO descriptions.

3 The LO Retrieval Problem

In this section we describe the retrieval aspect of our e-Librarian Service that can be perceived as a specialization of *passage retrieval* techniques. Passage retrieval techniques have been extensively used in standard IR settings, and have proven effective for document retrieval when documents are long or when there are topic changes within a document, thus making it an appealing candidate for the present work [14]. By *retrieval* we refer to answering a user's question by identifying only the semantically most pertinent LOs according to the given question. In addition, the system must be able to quantify the quality of the yielded results, i.e., to measure the semantic distance between the user's query and the identified LOs. This measure is also used to rank similar results.

Our solution is based on the *concept covering problem* and on the quantification of the *semantic difference*. The novelty of our approach is that it always proposes a solution to the user, even if the system concludes that there is no exhaustive answer. By quantifying the missing and supplementary information, the system is able to compute and visualize the quality and pertinence of the yielded LO(s).

3.1 Least Common Subsumer and Semantic Difference

The least common subsumer (lcs) [2] stands for the least concept description (w.r.t. subsumption) that subsumes a given set of concept descriptions.

Definition 1 (Least Common Subsumer). *Let $\mathcal{L}$ be a DL and C, D, E be $\mathcal{L}$-concept descriptions. The concept E is a lcs of C, D iff it satisfies:*

- *$C \sqsubseteq E$ and $D \sqsubseteq E$, and*
- *E is the least $\mathcal{L}$-concept description with this property, i.e., if E' is an $\mathcal{L}$ concept description satisfying $C \sqsubseteq E'$ and $D \sqsubseteq E'$, then $E \sqsubseteq E'$.*

Definition 2 (Semantic Difference). *[21] Let $\mathcal{L}$ be a DL and $C, D \in \mathcal{L}$ two concept descriptions with $C \sqsubseteq D$. Then the semantic difference $C - D$ is defined by:*

$$C - D = max_{\sqsubseteq}\{E \in \mathcal{L} : E \sqcap D \equiv C\}.$$

This definition of semantic difference requires that the second argument subsumes the first one. However, the semantic difference $C - D$ between two incomparable descriptions C and D can be given by computing the least common subsumer of C and D:

$$C - D = C - lcs(C, D).$$

3.2 Finding Pertinent Documents

Although the principle of the concept covering problem (see section 2) is the most pertinent solution for our E-Librarian Service, we think that a user might not be satisfied if the delivered answer to his/her precise question is a concatenation of different — normally not related — resources from the knowledge base. First, there is no transition between the different LOs in the answer. Second, we risk that there is mean to much information because the original concept covering problems adds all LOs to the answer until the answer is covered completely.

We learned from experiments [11] that users prefer few but precise answers even if these answers are not complete, rather than a set of different concatenated documents. This assertion is confirmed by pedagogical analyzes, e.g., [10, 6, 8, 4] that students are searching for one — the best — answer, and do not consider different delivered search results. They would rather reformulated their query until they receive only a few results, or until they find the perfect result.

Our modified concept covering problem defines a cover as a concept description C w.r.t. a terminology $\mathcal{T}$ that shares some information with another concept description Q w.r.t. $\mathcal{T}$.

Definition 3 (Cover). *Let $\mathcal{L}$ be a DL with structural subsumption, $\mathcal{T}$ be an $\mathcal{L}$-terminology and $C_{\mathcal{T}} = \{C_i \not\equiv \bot, i \in [1,n]\}$ the set of concept descriptions occurring in $\mathcal{T}$. Then $C_j \in C_{\mathcal{T}}$ is a cover of a $\mathcal{L}$-concept description $Q \not\equiv \bot$ if $Q - lcs_{\mathcal{T}}(Q, C_j) \not\equiv Q$.*

The best cover can be defined based on the remaining information in the query (denoted as $Miss$) and in the cover (denoted as *Rest*). The Miss is the part of the query that is not part of the cover, and the Rest is the information that is part of the cover but not required by the query.

Definition 4 (Miss and Rest). *Let Q, C be be two $\mathcal{L}$-concept descriptions.*

- *The Miss of Q w.r.t. C, denoted as $Miss(Q,C)$ is defined as follows: $Miss(Q,C) = Q - lcs_{\mathcal{T}}(Q,C)$.*
- *The Rest of Q w.r.t. C denoted as $Rest(Q,C)$ is defined as follows: $Rest(Q,C) = C - lcs_{\mathcal{T}}(Q,C)$.*

The best cover can be assumed as being the cover with the smallest Miss and Rest. Therefore, we have to quantify the Miss and the Rest, i.e., measure the size of a $\mathcal{L}$-concept description.

Definition 5 (Size of a Concept Description). *The size of a $\mathcal{L}$-concept description, denoted as $|\cdot|$ is inductively defined by:*

- $|\bot| = |\top| = 0$,
- $|A| = |\neg A| = 1$,
- $|\exists r.C| = |\forall r.C| = 2 + |C|$,
- $|C \sqcap D| = |C \sqcup D| = |C| + |D|$,
- $|\neg C| = |C|$.

Definition 6 (Best Cover). *Let C, D be two $\mathcal{L}$-concept descriptions. A cover C is called a best cover w.r.t. Q using a terminology $\mathcal{T}$ iff:*

- *C is a cover w.r.t. Q using $\mathcal{T}$, and*
- *there does not exists any cover C' of Q using $\mathcal{T}$ such that*

$$(|Miss(Q,C')|, |Rest(Q,C')|) < (|Miss(Q,C)|, |Rest(Q,C)|)$$

where $<$ stands for the lexicographic order.

By choosing a lexicographical order we give preference to a minimized Miss, e.g., for (Miss,Rest), the couple (1,2) < (2,1) because the first couple has a smaller Miss than the second one. In fact, the e-Librarian Service aims to give an exhaustive answer in the first place, i.e., to yield an answer that covers the user's query as much as possible, even if there is more information in the answer than required. Only in the second place, the Rest is considered in order to rank the results that have the same Miss.

3.3 Algorithm for the LO Retrieval Problem

Our best cover algorithm is called LOFind (see figure 1). As input a query Q is expected that was translated into a $\mathcal{L}$-concept description, and a $\mathcal{L}$-terminology $\mathcal{T}$, i.e., a set of semantic descriptions of LOs. The output of LOFind is the set E of best covers w.r.t. Q using $\mathcal{T}$.

```
Require: a query $Q \not\equiv \bot$, a set of concept descriptions $C_\mathcal{T} = \{C_i \not\equiv \bot, i \in [1, n]\}$
Ensure: a set of best covers $E = \{C_j \in C_\mathcal{T}, j \in [0..n]\}$
 1: $E \leftarrow \emptyset$
 2: $MinMiss \leftarrow +\infty$
 3: for each $C_i \in C_\mathcal{T}$ do
 4:   if $Q - lcs(Q, C_i \not\equiv Q)$ then
 5:     if $|Miss(Q, C_i)| < MinMiss$ then
 6:       $E \leftarrow C_i$
 7:       $MinMiss \leftarrow |Miss(Q, C_i)|$
 8:     else if $|Miss(Q, C_i)| = MinMiss$ then
 9:       $E \leftarrow E \cup C_i$
10:     end if
11:   end if
12: end for
```

Fig. 1. The algorithm LOFind

The algorithm works as follows. Let us suppose that $C_\mathcal{T}$ is the set of semantic descriptions of the LOs in our knowledge base. Then, each LO is tested if it is a cover (line 4). If so, then it will only be maintained, if either the size of its Miss is smaller than (line 5) or equal to (line 8) the smallest Miss found up to now. In the first case, the current LO replaces all the former best cover-candidates (lines 6 + 7). In the second case, the current LO is added to the best cover-candidates found up to now (line 9).

3.4 Illustrating Example

$\mathsf{LO}_1 \equiv \mathsf{Protocol}$
$\mathsf{LO}_2 \equiv \exists\mathsf{howWorks} \sqcap \mathsf{TCP/IP}$
$\mathsf{LO}_3 \equiv \mathsf{Protocol} \sqcap \exists\mathsf{hasTask.ErrorHandling}$
$\mathsf{LO}_4 \equiv \mathsf{Protocol} \sqcap \exists\mathsf{hasTask.FlowControl}$
$\mathsf{LO}_5 \equiv \mathsf{FlowControl}$

Fig. 2. Example of a terminology of LO definitions.

For the sake of simplicity, let us suppose that there are 5 LOs in the knowledge base. The corresponding semantic descriptions are shown in figure 2. We use the

DL sub-language $\mathcal{EL}$ that has structural subsumption and allows conjunction ($\sqcap$), existential restriction ($\exists r.C$), and the top concept ($\top$). The content of the LOs deals with the following topics:

LO_1: information about protocols in general,
LO_2: explanation how the protocol TCP/IP works,
LO_3: explanation that error handling is a task of a protocol,
LO_4: explanation that flow control is a task of a protocol,
LO_5: explanation of flow control.

Step 1: Expanding the Terminology. Expanding the terminology means, making explicit some implicit knowledge. The expanded terminology uses the example taxonomy about networking (see figure 3) and is shown in figure 4.

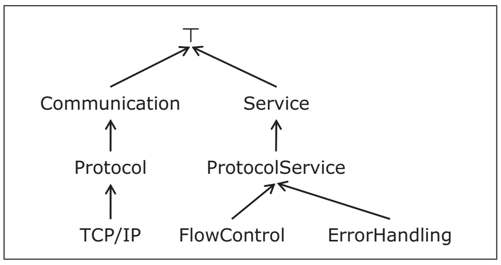


Fig. 3. Sample of a taxonomy about networking.

$\mathsf{LO}_1 \equiv \mathsf{Protocol} \sqcap \mathsf{Communication}$
$\mathsf{LO}_2 \equiv \exists \mathsf{howWorks} \sqcap \mathsf{TCP/IP} \sqcap \mathsf{Protocol} \sqcap \mathsf{Communication}$
$\mathsf{LO}_3 \equiv \mathsf{Protocol} \sqcap \mathsf{Communication} \sqcap \exists \mathsf{hasTask}.(\mathsf{ErrorHandling} \sqcap \mathsf{ProtocolService} \sqcap \mathsf{Service})$
$\mathsf{LO}_4 \equiv \mathsf{Protocol} \sqcap \mathsf{Communication} \sqcap \exists \mathsf{hasTask}.(\mathsf{FlowControl} \sqcap \mathsf{ProtocolService} \sqcap \mathsf{Service})$
$\mathsf{LO}_5 \equiv \mathsf{FlowControl} \sqcap \mathsf{ProtocolService} \sqcap \mathsf{Service}$

Fig. 4. Example of an expanded terminology.

Step 2: Computing the Covers. Let us suppose that the user has entered the NL question "*What are the tasks of TCP/IP?*", and that the question was translated into the following $\mathcal{EL}$-concept description: Q $\equiv \mathsf{TCP/IP} \sqcap \exists \mathsf{hasTask}$. In the expanded form the user's question can be denoted as:

$$Q \equiv \mathsf{TCP/IP} \sqcap \mathsf{Protocol} \sqcap \mathsf{Communication} \sqcap \exists \mathsf{hasTask}.$$

The aim is now to identify the LOs within the expanded terminology that cover the expanded query, i.e., that have something in common with Q; these are: LO_1, LO_2, LO_3, and LO_4.

Step 3: Computing the Best Cover. Now, for each cover the according Miss and Rest have to be computed. The best cover is the one with minimal Miss and Rest, with a preference to the minimal Miss.

	size of the Miss	size of the Rest
LO_1	$\lvert\mathsf{TCP/IP} \sqcap \exists\mathsf{hasTask}\rvert = 3$	$\lvert\top\rvert = 0$
LO_2	$\lvert\exists\mathsf{hasTask}\rvert = 2$	$\lvert\exists\mathsf{howWorks}\rvert = 2$
LO_3	$\lvert\mathsf{TCP/IP}\rvert = 1$	$\lvert\mathsf{ErrorHandling} \sqcap \mathsf{ProtocolService} \sqcap \mathsf{Service}\rvert = 3$
LO_4	$\lvert\mathsf{TCP/IP}\rvert = 1$	$\lvert\mathsf{FlowControl} \sqcap \mathsf{ProtocolService} \sqcap \mathsf{Service}\rvert = 3$

Conclusion: LO_3 and LO_4 are the best covers and are delivered as an answer to the user's query. Both LOs have the same Miss and Rest, 1 and 3, respectively so that their rank is the same. It is interesting to mention that the concept TCP/IP does not appear in one of the best covers, although it appears in the query and in LO_1. This shows that the best cover is not computed on a statistical evaluation of keywords, but that it is in fact the result of the logical inference.

Other covers, usually those where the size of the Miss is greater by one than the size of the Miss of the best cover, are yielded as second choice, here: LO_2.

4 Evaluation

Our algorithm was compared in a benchmark test with a traditional keyword-based search engine. Unfortunately, no similar measurements are available for the related projects referred in section 2.

4.1 Knowledge Base and Set of Questions

We used the online tele-TASK archive[2] that contains hundreds of recorded university lectures, as knowledge base. We selected the lecture series about Internetworking, which is a set of 30 units with a total of 38 hours of recorded lectures. We split the 30 lecture units into 1000 smaller LOs. A set of 123 NL questions about the topic Internetworking has been created. We tried to work out questions as students would ask, e.g., "*What is an IP-address composed of?*", "*How does a datapacket find its way through a network?*", "*What is a switch good for?*", "*Do internetprotocols guarantee an error-free communication?*". We also indicated for each question the relevant answer(s) that should be delivered.

[2] `http://www.tele-task.de/`

4.2 Evaluation Constraints

We call an answer from the e-Librarian Service a *perfect hit* if it covers the query completely, i.e., where the Miss and the Rest compute to zero. We call an answer from the e-Librarian Service a *sufficient hit* if it covers the query completely, but the answer contains more information than necessary, i.e., where the Miss equals zero and the Rest computes to some positive value.

For the evaluation we only considered the best covers with minimal Miss, not the second choices. This means that if the e-Librarian Service did not deliver an exhaustive answer as best cover but only as second choice, then we considered the answer to be wrong.

The results achieved with our e-Librarian Service have been compared with the results of a traditional keyword-based search engine. The keyword-based search engine is working in the usual way by browsing the textual content of the LOs. The textual content was generated by converting the PowerPoint-slides into pure text. A LO is considered to be a potential answer, if at least one (relevant) keyword from the user's query can be found. The keyword-based search engine does not consider stop words, i.e., words with no semantic relevance.

4.3 Benchmark Results

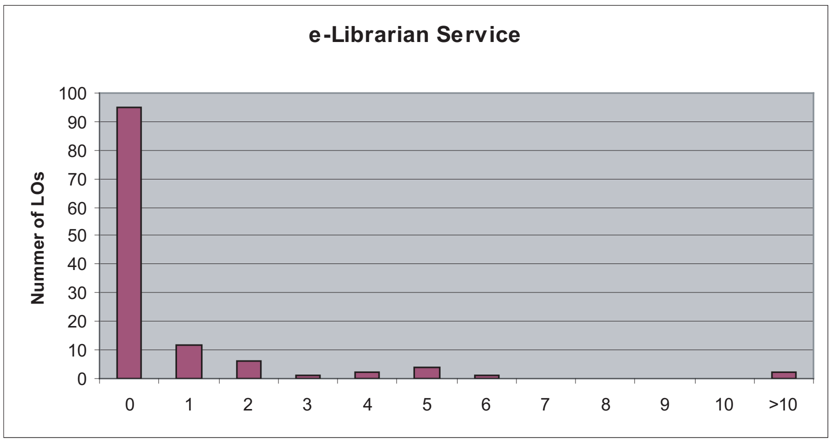
The benchmark test was performed on a standard Windows XP computer with a 1.4 GHz CPU and 512 MB of RAM. The e-Librarian Service has been implemented as a Java application. The processing time of the first question is about 200 ms, while for the rest it is less than 10 ms. The outcomes of the benchmark test are the following.

First, the e-Librarian Service scored better than the keyword search regarding the pertinence of the results. In most cases the e-Librarian Service yielded the correct answer:

	perfect hits	sufficient hits	total queries
e-Librarian Service	93 (76%)	112 (91%)	123 (100%)
Keyword search	9 (7%)	103 (84%)	123 (100%)

These numbers emphasize the pertinence of our e-Librarian Service as an appropriate tool for an educational environment; in most cases the learner gets a satisfying, even perfect, answer from the system. The fact that some answers contain little more information than necessary is no problem at all and can even have a positive effect for the learner.

Second, the precision of our solution is confirmed by the fact that in average less than 0.7 LOs are delivered in addition to the perfect answer (compared to 6 LOs for the keyword-based search). Figure 5 shows the number of supplementary LOs being delivered in addition to the expected answer. This important outcome points out that the e-Librarian Service usually is achieving the correct answer with no additional information (for 93 out of 123), and in a few cases one (12 out of 123) or two (6 out of 123) supplementary LOs. The keyword-based search engine in general delivers a lot more of secondary LOs.



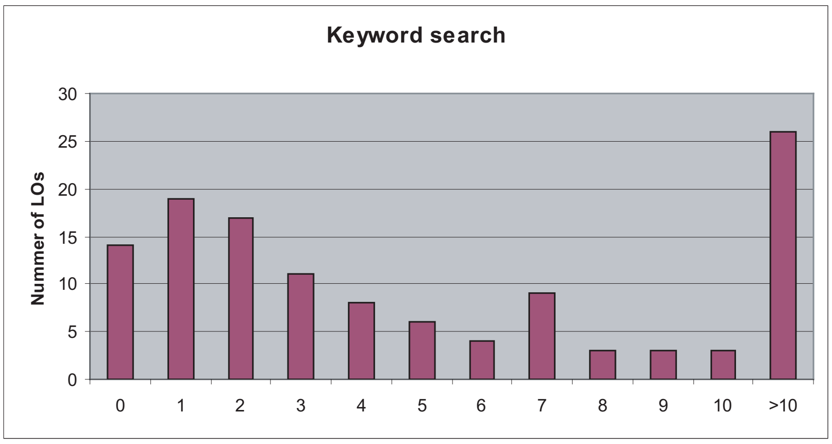


Fig. 5. Number of supplementary LOs yielded with the optimal answer.

This result is an important evidence for the pertinence of our tool in an educational environment; the user asks a precise question (or enters a keyword phrase) and expects few but concise answers. However, the keyword-based search leaves the user with the awkward task of filtering the pertinent answers out of the noise.

Third, in information retrieval the performance of a retrieval algorithm is measured by *recall* and *precision* [3]. Let use emphasize that for each question in the test set, there are only few relevant documents to be retrieved (in average 1.29 relevant answers per question). For this reason, we refer only to an average recall-level rather than to the 11 standard recall-levels. For an average recall-level, the precision of the e-Librarian Service is 84.41%, compared to 40.42% for the keyword-based search. These numbers confirm the previous outcome that our algorithm has a very high precision about the pertinence of the yielded answers; its average precision is more than twice as much than the precision achieved with the keyword-based search.

5 Improving Search Result Quality with User Feedback

As shown in the previous section, our e-Librarian Service is able to provide sufficient, even perfect, answers for most user questions. To further improve the quality of the search results, we decided to make use of the user's intellectual capabilities. The user has the possibility to vote for appropriate answers. Furthermore, we discuss possible diversification of user feedback and address the problem of general scalability of the e-Librarian Service.

5.1 Direct User Feedback

Direct user feedback can be achieved in different forms. The most simple way is to let the user determine whether a given result set of LOs really is appropriate according to his/her question or not. As usual, the user enters a query and the e-Librarian Service returns a list of LOs ordered by their computed rank. For each result a check box is displayed and the user has the possibility to indicate the appropriateness (and therefore indirectly also the quality) of the answer by leaving a mark in the check box. The e-Librarian Service has to keep track of user feedback and to channel that data into the rank computation of the LO result sets. Of course, different users might have different opinions about the accuracy of given answers.

The e-Librarian Service faces the problem to provide both an *objective answer* as well as a feedback-driven and therefore more or less *subjective answer*. For keeping track of the user feedback, an index data structure is maintained to provide efficient access. In the index, the users' questions (translated into DL formulas) are mapped with appropriate LOs and connected with a feedback-based computed rank (feedback rank) of each LO w.r.t. the user's question. In the simplest approach, the feedback rank corresponds to the number of users giving a positive feedback. The index is of the following form:

$$< user\_question, \ \{LO, \ feedback\_rank\} > .$$

For each user question the index provides access to the most appropriate LOs according to the user given feedback.

To avoid the aforementioned problem of objective and subjective answers, the e-Librarian Service displays both the (objective) best covers and the (subjective) feedback-based results. Thus, the user has the possibility to see objectively computed results and results according to the opinion of other users. If both results fit in the way that they both display the same top-rank result, the quality of our algorithm is confirmed.

User feedback might also serve as a personalization feature. For registered users the e-Librarian service is able to provide answers that have already been confirmed by the user's personal feedback. For this reason, the index data structure has to be extended to include also a set of user names for all users that have given feedback for a distinct LO:

$$< user\_question,\ \{LO,\ feedback\_rank, \{user\_names\}\} > .$$

In the same way, a more distinguished feedback is possible by giving the user the possibility to quantify the result's accuracy of fit within a given range of numbers. Instead of marking a simple check box (range: 0/1), the user has to enter a number corresponding to the appropriateness of the result set, e.g., -3 =does not fit at all … $+3$ =fits perfectly. Now, the index data structure has to provide the average user feedback for each LO as well as a set of user names including each user's personal feedback:

$$< user\_question,\ \{LO,\ avg.\ feedback\_rank, \{user\_name,\ user\_feedback\}\} > .$$

The probability that any two users are asking the same complex question obviously is rather low. Thus, in addition to an index entry corresponding to the complex (composite) user questions, supplementary index entries can be created for all single context literals of the DL formula that represent the user's question (by *concept literal* we refer to any atomic DL formula or its negation). There, we have to take the following into account: A complex user question might perfectly match with a given answer. But, for a single concept literal within the user's question this answer might indeed be appropriate but not perfect. Therefore, feedback-hits for complex questions have to get full feedback score, while feedback-hits for single concept literals (within the user's question) do only get a partial feedback score.

5.2 Diversification of User Feedback

Besides taking into account simple user feedback data, the question, if a given LO is well suited to provide the right answer to a user's question also depends on the user's expectations. Different users asking the same question might expect different answers. This comes, because different users prefer different levels of complexity, of difficulty, and of elaborateness [15]. Moreover, different users come from different background, have different motivations, and thus, different context.

Simple user feedback can be extended in different dimensions by providing facilities to express the users customized requirements and by giving the user the possibility to quantify those characteristics for given LO result sets. The user must be able to specify, if (s)he prefers complex and precise LOs or if a short overview about the requested topic is sufficient for his/her purposes. The other way around, the user should also be able to provide feedback data about the characteristics of a given LO. In this case, for each result set of LOs several switches have to be implemented (checkboxes, text fields, or sliders) to give the user the possibility to indicate his/her opinion about the diverse qualities of the presented LOs.

If the e-Librarian Service keeps track of the user's actions, also statistics can be gathered about LO usage. If a user has already accessed and used a given LO, this information can be used to customize the computation of the best cover w.r.t. the previous knowledge of the user. Anyway, connecting the logical inference capabilities of the e-Librarian Service with sophisticated user feedback information seems to be a promising approach to augment the quality of the computed search results and will be subject of further research.

6 Conclusion

In this paper we have proposed the e-Librarian Service CHESt based on a retrieval algorithm that returns only semantically pertinent LOs from a multimedia repository w.r.t. a user's query given in NL. We have applied two non-standard inferences of DLs — the least common subsumer (lcs), and the difference operation — to compute the best cover of the user's query. The e-Librarian Service has been developed in the context of the Web University project[3], which aims at exploring novel internet- and IT-technologies in order to enhance university teaching and research. Our solution is particularly interesting for education in a self-directed learning environment, where it fosters autonomous and exploratory learning [11].

A similar e-Librarian Service for learning fractions in mathematics with a different retrieval algorithm has already been tested successfully in school [11]. We were able to measure a relevant improvement in the students' scores. This is mainly attributed to the fact that the students were more motivated by using our system — because they quickly found the pertinent answer to their question(s) — and therefore put more effort into learning and acquiring new knowledge.

Currently, we are working to improve the quality of the achieved results by implementing approaches concerning the integration of user feedback and social networking information as described in section 5.

References

1. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
2. Franz Baader, Ralf Küsters, and Ralf Molitor. Computing Least Common Subsumers in Description Logics with Existential Restrictions. In *16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 96–101, 1999.
3. Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press / Addison-Wesley, 1999.
4. François-Marie Blondel. La recherche d'informations sur internet par des lycéens, analyse et assistance à l'apprentissage. In Peyrin J.P. Vries E., Pernin J.P., editor, *Hypermédias et Apprentissages 5 : Actes du cinquième colloque*, pages 119–133, 2001.

[3] http://www.hpi.uni-potsdam.de/~meinel/research/web_university.html

5. Simona Colucci, Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, and Azzurra Ragone. Semantic-based automated composition of distributed learning objects for personalized e-learning. In *European Semantic Web Conference (ESWC)*, pages 633–648, 2005.
6. Raya Fidel, Rachel K. Davies, Mary H. Douglass, Jenny K. Holder, Carla J. Hopkins, Elisabeth J. Kushner, Bryan K. Miyagishima, and Christina D. Toney. A visit to the information mall: Web searching behavior of high school students. *Journal of the American Society for Information Science*, 50(1):24–37, 1999.
7. Mohand-Saïd Hacid, Alain Leger, Christophe Rey, and Farouk Toumani. Computing concept covers: A preliminary report. In *Workshop on Description Logics*, 2002.
8. Christoph Hölscher and Gerhard Strube. Web search behavior of internet experts and newbies. *Computer Networks*, 33(1-6):337–346, 2000.
9. Ralf Küsters. *Non-Standard Inferences in Description Logics*, volume 2100 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.
10. Tessa Lau and Eric Horvitz. Patterns of search: Analyzing and modeling web query refinement. In ACM Press, editor, *Proceedings of the Seventh International Conference on User Modeling*, 1999.
11. Serge Linckels, Carole Dording, and Christoph Meinel. Better results in mathematics lessons with a virtual personal teacher. In *ACM SIGUCCS*, pages 201–209, 2006.
12. Serge Linckels and Christoph Meinel. Resolving ambiguities in the semantic interpretation of natural language questions. In *Intelligent Data Engineering and Automated Learning (IDEAL)*, volume 4224 of *LNCS*, pages 612–619, 2006.
13. Serge Linckels, Stephan Repp, Naouel Karam, and Christoph Meinel. The virtual tele-task professor: semantic search in recorded lectures. In *Technical Symposium on Computer Science Education (ACM SIGCSE)*, pages 50–54, 2007.
14. Xiaoyong Liu and W. Bruce Croft. Passage retrieval based on language models. In *Conference on Information and Knowledge Management (CIKM)*, pages 375–382, 2002.
15. Ulrike Lucke, Djamshid Tavangarian, and Denny Voigt. Multidimensional educational multimedia with <ml>^{3}. In *World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education (ELEARN)*, 2003.
16. Philippe Martin. *Web Intelligence*, chapter Knowledge Representation, Sharing and Retrieval on the Web, pages 263–297. Springer-Verlag, 2003.
17. Dan Moldovan, Sanda Harabagiu, Roxana Girju, Paul Morarescu, Finley Lacatusu, Adrian Novischi, Adriana Badulescu, and Orest Bolohan. LCC tools for question answering. In *Text REtrieval Conference (TREC) TREC*, 2002.
18. Raquel Navarro-Prieto, Mike Scaife, and Yvonne Rogers. Cognitive strategies in web searching. In *Conference on Human Factors & the Web*, 1999.
19. Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, and Marina Mongiello. Abductive matchmaking using description logics. In *18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 337–342, 2003.
20. Institute of Electrical and Electronics Engineers Learning Technology Standards Committee. IEEE standard for learning object metadata (draft). IEEE standard 1484.12.1, 2002.
21. Gunnar Teege. Making the difference: a subtraction operation for description logics. In *Principles of Knowledge Representation (KR)*, pages 540–550, 1994.
22. Christine Youngblut. Educational uses of virtual reality technology. Technical Report IDA Document D-2128, Defense Advanced Research Projects Agency, `http://www.hitl.washington.edu/scivw/youngblut-edvr/D2128.pdf`, Jan 1998.

ISWC+ASWC 2007

The 6th International Semantic Web Conference and
the 2nd Asian Semantic Web Conference

November 11~15 2007
BEXCO, Busan KOREA