ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2006, Vol. 35, No. 3 A

MATCHING SEMANTICALLY DESCRIBED WEB SERVICES USING ONTOLOGIES

Kim Christensen, Thorbjørn Højgaard Olesen, Lone Leth Thomsen

Department of Computer Science, Aalborg University
Fredrik Bajersvej 7, DK-9220 Aalborg, Denmark

Abstract. The theme for this article is Semantic Web Services. The motivation is the propagation of web services and the demand for dynamically exchanging web services in business processes (BPs). To achieve such dynamic exchange, it is necessary to be able to easily find web services that match a given set of requirements. Such requirements are best described with semantics, so it is necessary to develop a semantic UDDI repository (from now on denoted Sem-UDDI) for publishing of and searching for semantically described web services. Sem-UDDI is UDDI V2 compliant and uses a common interface based on UDDI categoryBags for publishing both OWL-S and WSDL-S described web services, and has a similar interface for searching. In order to sort the returned web services with respect to the search requirements, Sem-UDDI uses match score calculation rules based on commonalities between object properties in an OWL ontology concept hierarchy. Sem-UDDI is implemented as a layer to be put on top of a conventional UDDI repository, which gives companies the possibility of getting semantic functionality, while continuing to use their existing UDDI repository.

1. Introduction

According to [4], in the future web services will no longer be designed to function solely as distributed object oriented computing systems. Instead web services that utilise the Semantic Web[3] will make it possible for clients (users) to dynamically discover and use web services without a priori negotiations with the providers of the web service. This goal is certainly important to achieve for companies that deal with commercial web environments, be it business-to-business or business-to-consumer. This is due to the high level of market changes that exist in that particular area. Similarly, it is highly important to companies that provide web services, to be able to adapt their interfaces without interrupting and changing the programs their clients use. Furthermore, companies that want to compare prices or use alternate services within each transaction would like the possibility of flexible adaption of the interfaces presented by alternative services. This would be very beneficial in connection with interchanging services within their BPs. Currently, services that perform similar functions often have different interaction protocols and interfaces. This article proposes a solution to overcome the problem of determining whether one service seamlessly can be exchanged with another.

In the following we present our solution to annotating web services with semantics and the matching of such semantically described web services. Section 2 presents what a semantically described web service consists of, how the ontology should be represented, and the patterns that make it possible to publish and search for semantic web services described in either of the two standards, OWL-S[10] and WSDL-S[14]. Section 3 covers the matching of semantically described web services. Section 4 presents a brief explanation of how the solution is implemented. Section 5 relates our work to other projects covering the same area. Finally section 6 gives a conclusion on our proposed solution.

2. Combining OWL-S and WSDL-S

An ontology that describes the concepts used for annotating the operations of web services must be represented in order to use the patterns presented in the following.

2.1. Representation of an Ontology

The representation of an ontology for use here must be a so called *world* ontology. Furthermore, Burstein et el. [4] also argue that there is a need for a world ontology. A world ontology is characterised by describing virtually the entire world, i.e. it does not only contain information from one specific application domain, but covers all application domains. Having

world ontology ensures a non-redundant definition of concepts.

Using such an ontology, companies must describe their services by the concepts defined in this world ontology, an example of this type of ontology is shown in Figure 1. Such a world ontology is very comprehensive and should be created by gathering information from domain experts to ensure its correctness. Furthermore it would be beneficial if the ontology was shared publicly and only one single world ontology existed. Due to the enormous amount of work in standardizing such a world ontology, it would ideally be developed and administered by a W3C working group with members from different commercial companies. Currently, a world ontology does not exist, but the existence of taxonomies like NAICS[13] and UNSPSC[16] gives reason to believe that a world ontology will be developed.

There are a number of alternative approaches for representing the ontology, a discussion of these can be found in [6].

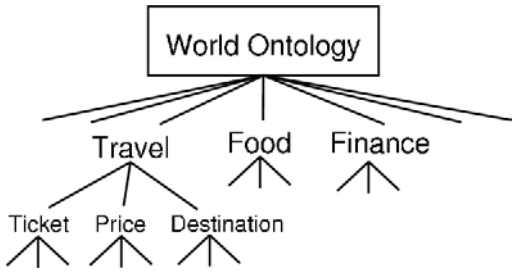

Figure 1. Shared-world Ontology

Given the assumption that such an ontology exists, it is possible to define patterns for publishing of and searching for semantically described web services. As shown in Figure 1, an ontology can be represented graphically in a tree structure. This is similarly shown in the matching example in Figure 8.

2.2. The Publish and Search Patterns

This section presents the patterns for publishing of and searching for semantically described web services. According to [4], the current trend is that web services are described using either OWL-S or WSDL-S. The patterns we present are thereby highly relevant due to the possibility of using both these standards in publishing and searching for semantically described web services.

2.2.1. Semantically Described Web Services

In this article, the meaning of a semantically described web service is a web service where every operation has a semantic description of its inputs and outputs. This information is required to determine whether a specific web service can be integrated seamlessly into a BP. The different parts of a BP have formal definitions, the most important of these are denoted service templates and service objects. For a full description, see [6].

Service Template (ST) An ST defines a web service that a company needs for its BP. An ST consists of a number of sub-components of which only the *inputs* and *outputs* are considered here. Naturally these are needed to define the flow of data within the BP.

Service Object (SO) An SO is a web service candidate that might be suitable to fit in a BP. This is determined by matching it with an ST. Therefore, an SO consists of the same sub-components as an ST.

2.2.2. Publish

The pattern for publishing a web service described in e.g. OWL-S requires the user to describe the web service by an OWL-S service profile which has references to concepts defined in the world ontology. The pattern is shown in Figure 2.

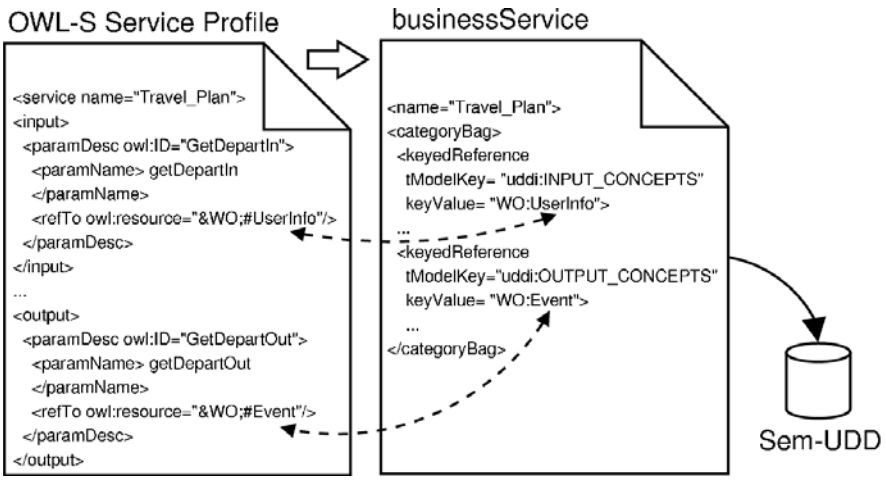

Figure 2. The publish pattern for OWL-S described web services

The information in the OWL-S service profile is used to create a businessService able to contain the information from the service profile. This data structure is sent to Sem-UDDI which handles the publish request by simply storing the businessService, making it available for future searches.

A businessService contains a categoryBag that holds a number of keyedReferences. Each keyedReference contains a tModelKey, a keyName, and a keyValue. The tModelKey contains a reference to a special tModel, which can be either uddi:INPUT_CONCEPTS or uddi:OUTPUT_CONCEPTS.

Note that Figure 2 contains no references to WSDL and the mapping between WSDL and the OWL-S service profile. Such a mapping exists and is called a service grounding, however it is not considered in this article since it has no relevance in connection with neither publishing nor searching.

The publish pattern just described is for OWL-S described web services, but the same pattern applies for WSDL-S.

2.2.3. Search

In order to obtain the semantics of the input and output types, the semantically described web services must be published in the manner described in Section 3.

If a service has been published in Sem-UDDI accordingly, it is possible to make a more specific search, compared to the conventional UDDI search method using NAICS or UNSPSC categories. The specific search is performed by using categoryBags containing the semantic input and output requirements.

To search for a semantically described web service a *find_ service* must be created from the ST. The structure of a find_service resembles the structure of a businessService, and is used when searching for web services. In the ST, a number of input and output parameters are defined. These parameters are the ideal parameters for the BP because if they match exactly only little work is needed for integrating the found web service.

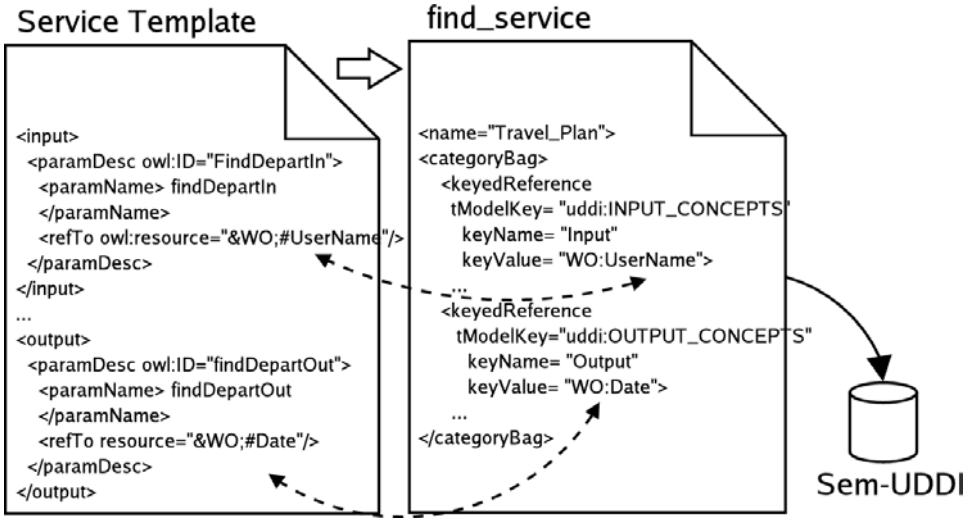

Figure 3. The search pattern for semantically described web services

The ST in Figure 3, which shows the overall search pattern, resembles the OWL-S service profile in Figure 2. In reality this is not the case as an ST only contains references to the input and output parameters, whereas an OWL-S service profile also contains references to concepts that define communication protocols etc. These are left out of Figure 2 for simplicity.

Some problems might arise using the search pattern though, imagine an example where averagely two web services are published per concept in the world ontology. If a search were performed using the concept *Date* (see Figure 8), only two web services would be matched for semantic similarity. If the search were extended to include the *TimePoint*, *CalendarDate* and the *Event* concepts, the matching would averagely be done on eight (four times two) web services. Six of these web services would be capable of fulfilling the specifications of the searched concept, because *CalendarDate* and *Event* subsume *Date*. The two web services described by *TimePoint* would however only be capable of partly fulfilling the specifications. The web services published with *TimePoint*, *CalendarDate* and *Event* as input concept would not be considered if only *Date* were used when searching.

This problem is solved by widening the search results to contain not only the searched concept itself, but also its parents and children. This is done by using *inferred* data, which is described in the following.

2.2.4. Searching with Inferred Data

The notion of an inferred search means that a search for a given concept will be widened when the search is performed in Sem-UDDI. This fact should though be hidden to the user, as the overall result of a search is a list of matching SOs ranked according to their match score.

In [8] methods for fetching the parents and children of a semantically described concept are suggested. These methods can be adapted to solve the problem with too few search results. This is done by adding extra keyedReferences to the categoryBag

created by the user. These extra keyedReferences are references to the immediate parents and children of the searched concept. Figure 4 shows the structure of the find_service after the addition of the inferred data for *Date*. If the immediate parents or children do not suffice, it is possible to follow the relations further out and thereby find additional concepts from the ontology. In General, it is difficult to estimate how large the increase in the number of web service hits is, as this depends on the number of parents and children and the number of published web services associated with each of them.

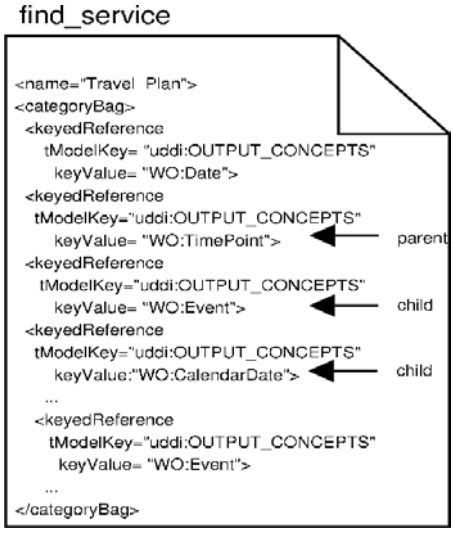

Figure 4. The new find_service containing inferred data

With this widened search in Sem-UDDI there are more results (SOs) to match than with a normal search using NAICS or UNSPSC categories.

3. Matching Rules

This section explains how matching between an ST and SO is performed, and describes Sem-UDDI's matching rules for both single and multiple parameters.

When a match must be done, the input and output parameters of the web service operation must be identified. This is done as explained in the previous section.

3.1. Matching STs and SOs

This section describes the matching of the semantics in STs and SOs. This includes a description of a number of functions that combined makes it possible to determine the degree to which an ST and SO are equivalent. The functions described in this section require the ontology describing the ST to be the same ontology that describes the SO. We consider such a world ontology to be a prerequisite. However a description of functions that match STs and SOs described by different ontologies can be found in [5]. Furthermore, the inspiration for calculating the geometric distance between different concepts is also found in [5].

In the example in Figure 6, an ST for the travel reservation service of the Request Conference BP must be matched against a number of more or less matching SOs. The ST must not violate internal functions already defined in the BP (the other puzzle bits). Thereby an ST for the Request Conference BP could look as described in Figure 5.

Request Conference ST	
Sub component	**Value**
Input	Date, Duration, Destination, Username, Address
Output	Itinerary

Figure 5. Composition of an ST

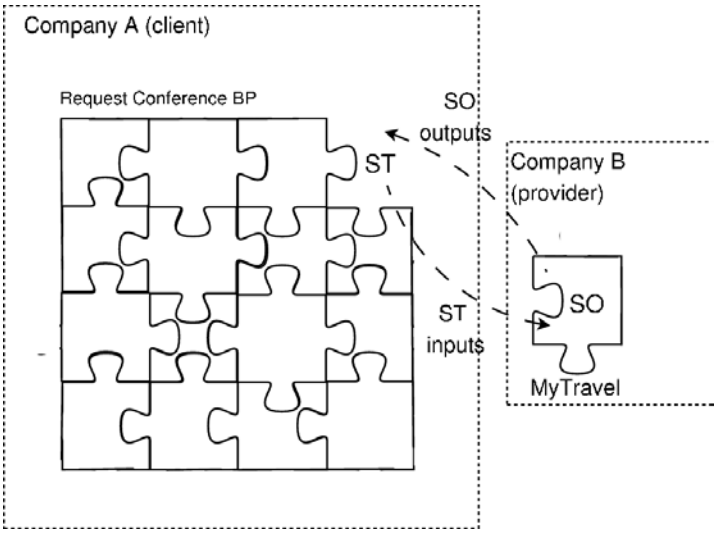

Figure 6. A BP that needs a travel service

When the ST is created as in Figure 5 it must be matched against a number of SOs. This matching must be done on the ontology concepts that describe them. To do this, a matching algorithm is needed.

The idea behind the matching algorithm is based on the notion of inheritance from object oriented programming. To determine whether two concepts are similar it is considered whether they subsume each other. If concept "A" subsumes concept "B", "A" has all the functionality of "B" and possibly even more. Thereby "A" is able to take over the role of "B". If the case is the opposite ("B" subsumes "A") then "A" cannot entirely take over the role of "B", but they have some properties in common.

The matching algorithm takes an ST and an SO as input parameters and returns a floating point number between 0 and 1, which represents the semantic similarity. Figure 7 shows how the algorithm works, it is based on Tversky's feature-based similarity model [15]. This model is based on the idea that common features tend to increase the similarity of two concepts. In this way the matching algorithm in Figure 7 computes the semantic similarity by how many common concepts they share.

```
double semantic_similarity(ST,SO) {
  if (ST.input.concept == SO.input.concept)
    return 1;
  else if (ST.input.concept < SO.input.concept)
    return 1;
  else if (ST.input.concept > SO.input.concept)
    return (SO.input.property.count()
           / ST.input.property.count());
  else {
    propertyList pl1, pl2, pl3, pl4;
    pl1 = ST.input.property.addAll();
    pl2 = SO.input.property.addAll();
    pl3 = intersect(pl1, pl2);
    pl4 = union(pl1, pl2);
    matchScore = sqrt(((pl3.count() / pl4.count())
                  *(pl3.count() / pl2.count())));
    return matchScore;
} }
```

Figure 7. The matching algorithm

The function `semantic_similarity()` returns a floating point number between 0 and 1, depending on the ratio of common properties between the ST and the SO. To make this computation, the function divides the problem in four cases:

1) The ST and SO concept are the same:

This is the simplest case. The two services have the exact same properties described, which means that the inputs of the ST and SO match exactly. Thereby the match score is evaluated to 1.

2) The SO concept subsumes the ST concept:

In this case all the properties (including the derived) of the ST also exist in the SO. Therefore the match score is evaluated to 1 as well.

3) The ST concept subsumes the SO concept:

This case handles the scenario where the SO concept is subsumed by the ST concept. This means that some of the properties of the ST could be missing in the SO. Therefore the match score is evaluated to $\frac{|\Pr op(SO)|}{|\Pr op(ST)|}$

4) The concepts do not subsume each other at all:

In the most complicated case the concepts do not subsume each other in any way. Therefore the geometric distance between the ST and SO must be calculated. This way of calculating the similarity was originally proposed in [5]. The geometric distance is calculated in the last `else`-clause in Figure 7 and is described mathematically by

$$\sqrt{\frac{|\Pr op(ST)\bigcap \Pr op(SO)|}{|\Pr op(ST)\bigcup \Pr op(SO)|} * \frac{|\Pr op(ST)\bigcap \Pr op(SO)|}{|\Pr op(SO)|}}$$

3.1.1. An Ontology Matching Example

To illustrate the idea of the matching functions an example of ontology is in place. This example builds upon the general example presented in this article. As mentioned earlier, the ST is obliged to take as input whatever the internal functions of the BP dictates. According to Figure 9 this is a `Date` in this case.

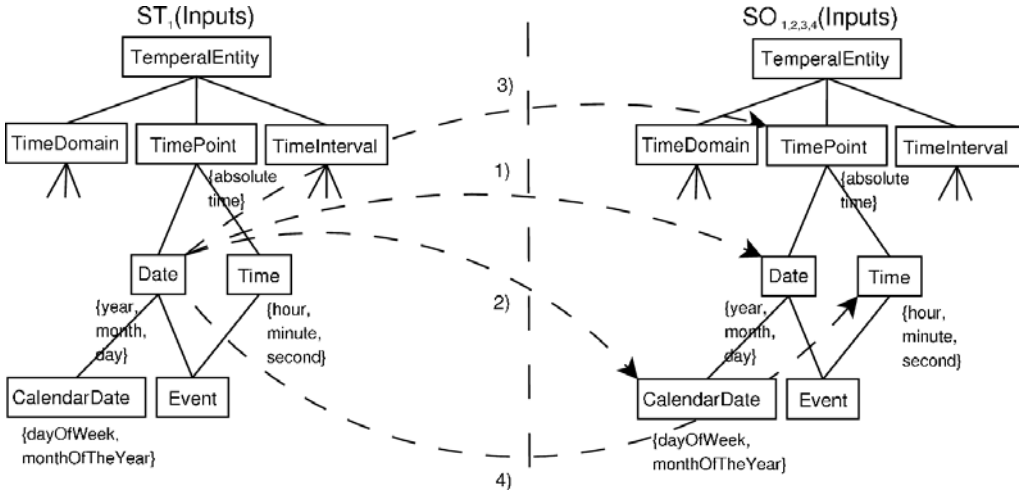

Figure 8. The Time ontology in a matching situation

	ST	Input	SO	Input	Score
1)	MyTravelST	Date	MyTravelSO	Date	1
2)	MyTravelST	Date	SASAirSO	Calendar-Date	1
3)	MyTravelST	Date	LHFlightSO	Time-Point	0.25
4)	MyTravelST	Date	BAFlySO	Time	0.189

Figure 9. The matching of one ST to four different SOs

If the matching algorithm above should be invoked on this example, the match scores for the ST and the SOs will look as in Figure 9. This example assumes that the number of input parameters for `MyTravelST` is one. Furthermore the properties of the respective concepts are described in curly brackets under the concept. The calculation of case 4 is made using the following four equations:

- s_1 = **Prop(Date)** = {absolute_time, year, month, day}, size = 4
- s_2 = **Prop(Time)** = {absolute_time, hour, minute, second}, size = 4
- s_3 = **Prop(Date)** $\cap$ **Prop(Time)** = {absolute_time}, size = 1
- s_4 = **Prop(Date)** $\cup$ **Prop(Time)** = {absolute_time, year, month, day, hour, minute, second}, size = 7

Using these four equations (s_1 is used as an intermediate result for s_3 and s_4), the match score for case 4 is:

$$matchScore = \sqrt{\frac{|s_3|}{|s_4|} * \frac{|s_3|}{|s_2|}} = \sqrt{\frac{1}{7} * \frac{1}{4}} = 0{,}189$$

The match score mentioned in this section considers the case where the similarity between two concepts should be calculated. The cases where parameters of a categoryBag are void, or not known in the world ontology, are presented in the following section.

3.2. Single Parameter Rules

Generally for single parameters, there are three possible cases to handle. These are the cases where the concept received in the categoryBag (containing the parameters from the operation) are *Known* or *Unknown* to the world ontology used, or if the parameter is *Void*.

The following example shows an ST and an SO with a maximum of one parameter for input and output. The example takes outset in a scenario where a company needs a web service that books a conference ticket for a conference that takes place at a certain time.

	Input parameters	Output parameters
ST	Time	Void
SO	TimeDuration	Ticket

Figure 10. Example of an ST and an SO with single parameters

The three possible cases (*Known*, *Unknown*, and *Void*) should be handled differently. In this example, the concept *TimeDuration* is *Unknown*, thus the match score between *Time* and *TimeDuration* will be set to 0. The *Void* case must be handled according to the rules in Figure 11, and thereby in this case the match score must be set to 1. If the concepts to be matched are both *Known*, the matching algorithm described in Figure 7 is used.

Pseudo code for the rules to calculate the match score for each type of concept is given in Figure 11. These rules only handle input concepts, but the rules are similar for output concepts.

```
if (ST.in == ''Void'')
  matchScore = 1;
else if (ST.in == ''Unknown'')
  matchScore = 0;
else if (ST.in == ''Known'') {
  if (SO.in == ''Known'')
    matchScore = semantic_similarity(ST.in, SO.in);
  else
    matchScore = 0;
}
```

Figure 11. The rules for calculating match scores

3.3. Multiple Parameter Rules

The match score algorithm presented in Section 3 leaves multiple parameters out of consideration, thus the following matching rules were designed due to the fact that the match score must cover an entire service and not just one ontology concept.

When matching is to be performed between an ST and an SO with more than one input parameter, all possible matching combinations of both the input and output parameters must be considered. This means that each input parameter in the ST must be matched against all possible input parameters in the SO, and similarly for output parameters.

The following example shows how the calculation of the match score is done with multiple parameters. This example also takes outset in the scenario where a company needs a web service that is able to book a ticket for a conference that takes place at a certain time.

	Input parameters	Output parameters
ST	Company, Time	Ticket
SO	Address, Company, Date	Event

Figure 12. Example of an ST and an SO with multiple parameters

This example shows how the match score of the input concepts is calculated for the case where the inputs of the ST are (*Known*, *Known*) and the inputs of the SO are (*Unknown*, *Known*, *Known*).

In general, this gives a total of $n \times m$ combinations (in this example 2×3), each with one match score. Each of the parameters in the ST is matched against all parameters in the SO. From these match scores, the highest is extracted and used to calculate the overall match score. The overall match score is calculated by adding these highest match scores and dividing the result with the number of parameters of either the ST or SO. The choice between these depends on the one having the most parameters.

		ST	
		Company	**Time**
SO	**Address**	0	0
	Company	1	0
	Date	0	0.189

Figure 13. Match score calculation example with multiple input parameters

The example only handles the input parameters, but the calculation regarding multiple output parameters is similar. The input match score is calculated as follows:

$$inputMatchScore = \frac{1 + 0{,}189}{3} = 0{,}396$$

Similarly the output match score is calculated:

$$outputMatchScore = \frac{0}{1} = 0$$

Thus the overall match score for the above example is calculated to be:

$$matchScore = \frac{1 + 0{,}189}{3} * 0{,}5 + 0 * 0{,}5 = 0{,}198$$

The match score in this case is weighted equally for input and output, therefore the weight is set to 0.5. The match score calculation is all quite straightforward, but some twists are worth mentioning. For instance, the *Void* case will naturally never occur if multiple parameters are defined. Furthermore, when two parameters are matched and one of them is *Unknown*, the match score is determined to be 0. The example only handles one single case (2×3 parameters), but the calculation rules are similar no matter how many parameters the ST or SOs consist of.

4. Implementation

The theories behind the publish and search patterns described in Section 2 along with the matching rules described in Section 3 have been implemented as a prototype of the semantic UDDI repository called Sem-UDDI.

Sem-UDDI is implemented as a layer to be put on top of an existing version 2 compliant UDDI repository, denoted as X-UDDI. The advantage of having Sem-UDDI as a layer instead of a self-contained UDDI repository is that companies can continue to use their existing repository while getting semantic functionality.

UDDI clients such as UDDI4J[7] or Microsoft Visual Studio[11] will communicate with Sem-UDDI like any other UDDI repository because it has the same interface as described in the UDDI version 2 API specification[2]. The API contains several functions divided in two categories, those used for inquiring and those used for publishing in a repository. Of all these, only the `find_service` function from the inquire API is of special interest, since the rest of the inquire functions and all of the publish functions are forwarded unchanged by Sem-UDDI to X-UDDI. Figure 14 shows how Sem-UDDI lies as an intermediate layer between the UDDI client and X-UDDI.

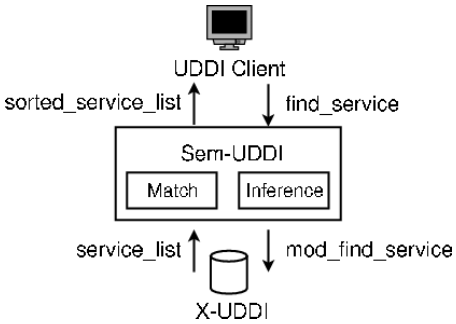

Figure 14. Sem-UDDI as a layer

When a UDDI client sends a `find_service` request, containing an ST, Sem-UDDI modifies the request to contain more ontology concepts with the purpose of increasing the search results. The idea behind and the method for finding additional related concepts to widen the search, i.e. the use of inferencing, are described in Section 3. For each of the concepts used as search requirements in the ST in the `find_service` request, the `Inference` module finds the parents and children concepts from the world ontology. These extra concepts are added, enclosed in `keyedReferences`, to the `categoryBag` of the `find_service` request. After modifying the ST in the `find_service` request, Sem-UDDI sends it further to X-UDDI which is responsible for the actual data storage and data retrieval. X-UDDI responds with a `service_list`, containing a list of web services (SOs) matching any of the concepts used in the ST in the `find_service` request as search requirements. For each of the SOs contained in the `service_list`, the `Match` module calculates a match score indicating similarity of the SO, compared to the ST. Finally, Sem-UDDI sorts the SOs with respect to the calculated match score, before sending the `service_list` back to the UDDI client.

5. Related Work

Sem-UDDI's publish and search patterns are inspired by the OWL-S approach described in [9] and

the WSDL-S approach described in [12]. Both use categoryBags to contain keyedReferences to ontology concepts for semantically describing input and output parameters. [9] suggests that additional information contained in the OWL-S service profile should be stored with the businessService in UDDI, however this is not considered by Sem-UDDI, since it only bases its searches on input and output requirements. Similarly, [12] suggests to annotate operations with semantic concepts, but this is not considered by Sem-UDDI either.

The matching algorithm used in Sem-UDDI is inspired by [5] that suggests using ontology property inheritance for calculating the match score for a Service Template and a Service Object. However, [5] considers only STs and SOs with a single parameter, where the matching rules in Sem-UDDI can handle STs and SOs with multiple input and output parameters.

The recent work in [1] takes outset in earlier work of the WSDL-S approach. One of the new ideas is to give web service developers the freedom to choose which semantic language to use for annotating input and output parameters, i.e. they can use e.g. OWL, UML, or WSMO[17]. Currently, Sem-UDDI can use OWL ontologies only, but could be extended to support additional languages, given that they are compatible with Sem-UDDI's search and publish patterns.

6. Conclusion

To be able to conduct business faster and more efficient, companies need to optimise their business processes to be able to handle dynamic exchanges of web services. This cannot be achieved with the conventional web service technologies available today, thus a new means for easily finding web services that match a given set of requirements must be found.

The considerations of how the optimal representation of an ontology should be, revealed that the need for a world ontology is significant. Furthermore, our work has shown that it is possible to design and implement a semantic UDDI repository that provides functionality for publishing of and searching for both non-semantically and semantically described web services. The repository was developed to be capable of handling both of the two standards OWL-S and WSDL-S for semantically describing web services. Even though OWL-S and

WSDL-S have different ways of describing web services semantically, a common interface combining the two approaches was developed. This interface was built upon the notion of ontologies written in OWL to define semantic concepts.

To compare semantic search requirements with published semantically described web services, a set of rules for calculating a match score was developed.

An important design choice was to develop the semantic UDDI repository as a layer to be put on top of a conventional UDDI repository. Hereby companies can get semantic functionality while still using their existing UDDI repositories. This also makes the semantic UDDI repository flexible, since it is not dependent on a specific UDDI implementation.

The current implementation of the semantic UDDI repository is functional, but limited in that when publishing a web service, it is only possible to define one operation for it.

The future of semantic UDDI repositories highly depends on other technologies like the Semantic Web and the propagation of web services in high value application areas. Currently, the Semantic Web is in a preliminary state, but web services are in constant growth. If this trend continues it is likely that the need for dynamically exchanging web services in business processes will become significant, and semantic UDDI repositories will ease the process of binding companies together.

References

[1] **R. Akkiraju et al.** Web Service Semantics - WSDL-S. *http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s,* 2005.

[2] **T. Bellwood et al.** UDDI Version 2.04 API Specification. *http://uddi.org/pubs/ProgrammersAPI_v2.htm*, 2002.

[3] **T. Berners-Lee.** Semantic Web Roadmap. *http://www.w3.org/DesignIssues/Semantic.html*, 1998.

[4] **M. Burstein, C. Bussler, T. Finin, M.N. Huhns, M. Paolucci, A.P. Sheth, S. Williams, M. Zaremba.** A Semantic Web Services Architecture. *http://ebiquity.umbc.edu/get/a/publication/208.pdf,* 2005.

[5] **J. Cardoso, A. Sheth.** Semantic e-Workflow Composition. LSDIS, Department of Computer Science, University of Georgia, *http://lsdis.cs.uga.edu/lib/download/TM02-004-Cardoso-Sheth.pdf*, 2003.

[6] **K. Christensen and T.H. Olesen.** Sem-Uddi: A Semantic UDDI Repository combining OWL-S and WSDL-S. *Department of Computer Science, Aalborg University, http://www.cs.aau.dk/~tho/sem-uddi.pdf*, 2005.

[7] **K. Jagger et al.** UDDI4J. *http://uddi4j.sourceforge.net*, 2005.

[8] Jena. Jena Semantic Web Framework - Inference Engine. *http://jena.sourceforge.net/inference*, 2004.

[9] **T. Kawamura, M. Paolucci, T.R. Payne, K.P. Sycara.** Importing the semantic web in uddi. *CAiSE*'02/*WES*'02: *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, *Springer-Verlag*, 2002, 225–236..

[10] **D. Martin et al.** OWL-S: Semantic Markup for Web Services. *http://www.w3.org/Submission/OWL-S*, 2005.

[11] Microsoft. Microsoft Visual Studio. *http://msdn.microsoft.com/vstudio*, 2005.

[12] **J. Miller, A. Sheth, K. Sivashanmugam, K. Verma.** Adding semantics to web service standards. 1*st International Conference on Web Services*, *LSDIS, Department of Computer Science, University of Georgia, June* 2003, 395–401.

[13] NAICS. North American Industry Classification System. *http://naics.org*, 2005.

[14] **A. Sheth et al.** Web Service Semantics - WSDL-S. *http://www.w3.org/2005/04/FSWS/Submissions/17/WSDL-S.htm*, 2005.

[15] **A. Tversky.** Wikipedia (Features of Similarity). *http://en.wikipedia.org/wiki/Amos_Tversky*, 2005.

[16] UNSPSC. United Nations Standard Product and Services Classification (UNSPSC) Homepage. *http://www.unspsc.org*, 2005.

[17] WSMO. Web Service Modelling Ontology. *http://www.wsmo.org*, 2005.

Received August 2006.