

Web Services Discovery Utilizing Semantically Annotated WSDL

Jorge Cardoso¹, John A. Miller², and Savitha Emani²

¹ SAP Research CEC Dresden
Chemnitzer Strasse 48
01187 Dresden, Germany

jorge.cardoso@sap.com

² LSDIS Lab, Department of Computer Science
University of Georgia
Athens, GA 30602 – USA

jam@cs.uga.edu, emani@cs.uga.edu

Abstract. To make semantic Web services accessible to users, providers use registries to publish them. Unfortunately, the current registries use discovery mechanisms which are inefficient, as they do not support discovery based on the semantics of the services and thus lead to a considerable number of irrelevant matches. Semantic discovery and matching of services is a promising approach to address this challenge. This paper presents an algorithm to match a semantic Web service request described with SAWSDL against semantic Web service advertisements. The algorithm is novel in three fundamental aspects. First, the similarity among semantic Web service properties, such as inputs and outputs, is evaluated using Tversky's model which is based on concepts (classes), their semantic relationships, and their common and distinguishing features (properties). Second, the algorithm, not only takes into account services' inputs and outputs, but it also considers the functionality of services. Finally, the algorithm is able to match a semantic Web service request against advertisements that are annotated with concepts that are with or without a common ontological commitment. In other words, it can evaluate the similarity of concepts defined in the context of different ontologies.

Keywords: We Semantic Web, Web services, Ontologies.

1 Introduction

Semantic Web services are the new paradigm for distributed computing. They have much to offer towards the integration of heterogeneous, autonomous and large scale distributed systems. Several standards such as WSDL [1, 2], UDDI [3], and SOAP [4] have been developed to support the use of Web services. Significant progress has been made towards making Web services a pragmatic solution for distributed computing on the scale of the World Wide Web. With the proliferation of Web services and the evolution towards the semantic Web comes the opportunity to automate various Internet related tasks. Applications should be able to automatically or semi-automatically

discover, invoke, compose, and monitor Web services offering particular services and having particular properties [5].

Given the dynamic environment in e-businesses, the power of being able to discover Web services on the fly, to dynamically create business processes is highly desirable. The discovery of Web services has specific requirements and challenges compared to previous work on information retrieval systems and information integration systems. Several issues need to be considered. The discovery has to be based, not only on syntactical information, but also on data, as well as functional and QoS semantics [6].

Discovery is the procedure of finding a set of appropriate Web services that meets user requirements [7]. The discovery of Web services to model Web processes differs from the search for tasks/activities to model traditional processes, such as workflows. One of the main differences is in terms of the number of Web services available to the composition process. On the Web, potentially thousands of Web services are available which make discovery a difficult procedure. One cannot expect a designer to manually browse through all the Web services available and select the most suitable one. Therefore, one of the problems that needs to be overcome is how to efficiently discover Web services [6].

Currently, the industry standards available for registering and discovering Web services are based on UDDI specifications [3]. An important challenge is that of finding the most appropriate Web service within a registry [7]. This challenge arises due to the discovery mechanism supported by UDDI. In an attempt to disassociate itself from any particular Web service description format, UDDI specification does not support registering the information from the service descriptions in the registry. Hence the effectiveness of UDDI is limited, even though it provides a very powerful interface for keyword and taxonomy based searching. Suggestions [8] have been made to register WSDL descriptions, which are the current industry's accepted standard, in UDDI. However, since WSDL descriptions are syntactic, registering them would only provide syntactical information about the Web services. The problem with syntactic information is that the semantics implied by the information provider are not explicit, leading to possible misinterpretation by others. Therefore, discovering Web services using UDDI is relatively inefficient since the discovery mechanism only takes into account the syntactic aspect of Web services by providing an interface for keyword and taxonomy based searching.

The key to enhance the discovery of Web services is to describe Web services semantically [9] and use semantic matching algorithms (e.g. [6, 10-12]) to find appropriate services. Semantic discovery allows the construction of queries using concepts defined in a specific ontological domain. By having both the advertisement description and request query explicitly declare their semantics, the results of discovery are more accurate and relevant than keyword or attribute-based matching. Adding semantics to Web service descriptions can be achieved by using ontologies that support shared vocabularies and domain models for use in the service description [7]. Using domain specific ontologies, the semantics implied by structures in service descriptions, which are known only to the writer of the description, can be made explicit. While searching for Web services, relevant domain specific ontologies can be referred to, thus enabling semantic matching of services.

In this paper, we will review the state-of-the-art in the discovery of Web services. We then present a new algorithm for Web service discovery that is novel in three

fundamental aspects. First, the similarity among semantic Web service properties, such as inputs and outputs, are determined based on a feature-based model, Tversky's model. Using Tversky's model, we consider that similarity is a judgment process that requires two services to be decomposed into aspects in which they are the same and aspects in which they are different. Evaluating the similarity is based on concepts (classes), their semantic relations, and their common and distinguishing features (properties). Second, the algorithm, not only takes into account services' inputs and outputs, but it also considers the functionality of services. This allows for increasing the precision of search. Providers can express in a better way the objective of their services and customers can give a better characterization of the services they are looking for. Finally, the algorithm is able to match a semantic Web service request against advertisements that are annotated with concepts that are with or without a common ontology commitment. In other words, it can evaluate the similarity of concepts defined in the context of different ontologies. This last characteristic is important since in some situations it is perfectly acceptable to find similar services (or even equivalent services) annotated with semantic concepts that exist in the context of different ontologies.

The remainder of this paper is structured as follows. Section 2 provides an overview on how Web services can be semantically annotated or described so that they can be considered semantic Web services. We present an approach to add semantics to WSDL. The tool Radiant is used to exemplify the essential functionalities needed for an annotation tool. In section 3, we present our semantic Web service matching function (called SM-T) to discover services. It also describes a ranking algorithm that uses the matching function previously presented and that can be used by discovery mechanisms. Section 4 explains how the SM-T function can be integrated in the METEOR-S Web Services discovery infrastructure. This system supplies an infrastructure of registries for semantic publication and discovery of Web services. Section 5 discusses the related work in this area and the last section presents our conclusions.

2 Semantic Web Service

Many believe that a new Web will emerge in the next few years, based on the large-scale research and development ongoing on the semantic Web and Web services. The intersection of these two, semantic Web services, may prove to be even more significant. Academia has mainly approached this area from the semantic Web side, while industry is beginning to consider its importance from the Web services side [13]. Three main approaches have been developed to bring semantics to Web services:

- The first approach uses OWL-S, a Web Service description language that semantically describes the Web using OWL ontologies. OWL-S services are then mapped to WSDL operations and inputs and outputs of OWL-S are mapped to WSDL messages.
- The second approach, WSMO, is a meta-model for semantic Web services devised to facilitate the automation of discovering, combining and invoking electronic services over the Web. WSMO elements include: Ontologies, Web services, Goals and Mediators.

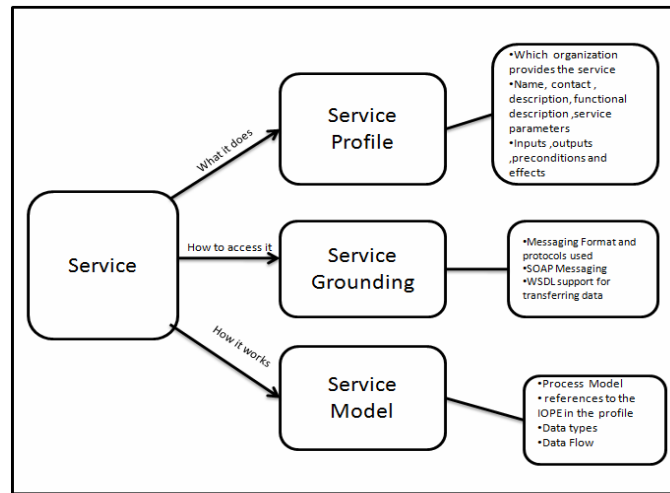


Fig. 1. OWL-S: Service ontology overview

- The third approach to creating semantic Web services is by mapping concepts in a Web service description (WSDL specification) to ontological concepts. The WSDL elements that can be marked up with metadata are operations, messages, preconditions and effects, since all the elements are explicitly declared in a WSDL description.

The approaches will be discussed in the following subsections.

2.1 OWL-S

OWL-S [14] (formerly DAML-S) is a standard ontology or language which gives providers a computer-interpretable description of a Web service. It supplies a set of classes and properties which describes capabilities of a Web service in an unambiguous, computer form. This ontology uses OWL as the web compatible representational language. As OWL-S gives a markup to the Web services it helps in automated discovery, composition and interoperation of services. OWL-S employs an upper level ontology to describe Web services. It consists of three parts expressed in accordance with OWL ontologies: the service profile (What does the service provide for prospective clients?), the service model (How is it used?), and the service grounding (How does one interact with it?), each of these perspectives provide essential information about the service (Figure 1).

The Service Profile used to discover a Web service gives complete information on whether a particular service meets the requirement of a user or not. This information involves what the service capabilities are, its limitations and the quality of service. It gives detailed information about the name, contact, description of the service, specification of parameters (properties) according to the process ontology, Inputs, Outputs, Preconditions and Effects (IOPE). The Service Model gives a layout of how a consumer should pass requests and how the service accomplishes the task. When services

are composed the consumer can use the description in different ways: to analyze whether the service meets the requirements in detail, to compose multiple services for a specific task, to synchronize and coordinate different participants and to monitor the execution of the services. The services are modeled as processes; the IOPEs declared in the service profile are referenced here. If the processes are connected with each other then the dataflow between these processes is specified. The Service Grounding specifies the communication protocol, message formats and other details used to access the web service. Concrete messages are specified in grounding i.e., how the inputs and outputs are of a process are realized as messages in some transmittable format. WSDL is used to support initial grounding mechanism as a set of endpoints for messages along with SOAP binding where HTTP is the communication protocol that is used.

2.2 WSMO

The Web Service Modeling Ontology (WSMO [15]) comprises an ontology of core elements for semantic Web services, described in WSML (Web Services Modeling Language), a formal description language, and also an execution environment called WSMX (Web Service Execution Environment). In WSMO, ontologies provide the terminology used by other WSMO elements to describe the relevant aspects of the domains of discourse. Goals symbolize user desires which can be satisfied by executing a Web service and Mediators express elements that surmount interoperability problems between distinct WSMO elements. WSMO and OWL-S, both accept the same view towards having service ontologies to construct semantic Web services. WSMO has its own family of languages, WSML, which is based on Description Logics and Logic Programming.

As WSMO provides ontological specifications for the elements of Web services it is designed on the basis of few principles: it identifies the resources with the help of URIs, it is based on an ontology model and supports ontology languages designed for the semantic Web, each resource is defined independently, it handles heterogeneity, it separates between client and the available services, it provides and differentiates between description and implementation, it describes Web services that provide access to a service (actual value obtained after a Web service is invoked).

WSMO uses different approaches to discover Web services which require different annotation and description of goals and services. Web service discovery is done by matching goal descriptions with semantic annotations of Web services. This type of discovery happens in an ontological level. Two main processes are required for this discovery: the user input will be generalized to more abstract descriptions and services and their descriptions should be abstracted to classes of services.

2.3 Adding Semantics to WSDL

It has been recognized [5] that due to the heterogeneity, autonomy and distribution of Web services and the Web itself, new approaches should be developed to describe and advertise Web services. The most notable approaches rely on the use of semantics to describe Web services. This new breed of Web services, termed semantic Web services, will enable the automatic annotation, advertisement, discovery, selection,

composition, and execution of inter-organization business logic, making the Internet become a common global platform where organizations and individuals communicate with each other to carry out various commercial activities and to provide value-added services. The academia has mainly approached this area from the semantic Web side, while industry is beginning to consider its importance from the point of view of Web services [13]. As we have already seen, three main approaches have been developed to bring semantics to Web services: SAWSDL (formally WSDL-S), OWL-S [14], and WSMO [15]. Since our work has been carried out with the research group that has defined SAWSDL, we will focus our study on this specification.

2.3.1 WSDL

WSDL [2] is primarily an interface description language for Web services, just as IDL was for CORBA. As an interface, it describes capabilities that Web services implementing the interface should provide. The main thing to describe about an interface is the set of operations. In WSDL, the meaning of an operation is given by the operation name, the input parameter names and types, the output parameter names and types as well as the possible faults that can be thrown. In addition, further information can be obtained from the interface itself and in WSDL 2.0 one interface can extend another (interface inheritance).

A WSDL document describes a Web service as a collection of ports. Messages specify data being exchanged between the services and port types are collection of operations. As such a WSDL document has certain elements to define data types, messages, operations, port types, binding, ports and services. Figure 2 shows a complete example of how a WSDL looks like.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net.
RI's version is JAX-WS RI 2.1.2-hudson-182-RC1.-->
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net.
RI's version is JAX-WS RI 2.1.2-hudson-182-RC1. -->

<definitions xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://stock/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns=http://schemas.xmlsoap.org/wsdl/
    targetNamespace="http://stock/" name="estockincService">

<types>
  <xsd:import namespace=http://stock/ "
schemaLocation="http://localhost:8080/WebService/estockincServi
ce?xsd=1" />
</types>

<message name="stockquoterequest">
```

Fig. 2. An example WSDL Document

```

    <part name="parameters" element="tns:stockquoterequest" />
</message>
<message name="stockquoterequestResponse">
    <part name="parameters" ele-
ment="tns:stockquoterequestResponse" />
</message>

<portType name="estockinc">
    <operation name="stockquoterequest">
        <input message="tns:stockquoterequest" />
        <output message="tns:stockquoterequestResponse" />
    </operation>
</portType>

<binding name="estockincPortBinding" type="tns:estockinc">
    <soap:binding trans-
port="http://schemas.xmlsoap.org/soap/http" style="document" />
    <operation name="stockquoterequest">
        <soap:operation soapAction="" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
        </output>
    </operation>
</binding>

<service name="estockincService">
    <port name="estockincPort" bind-
ing="tns:estockincPortBinding">
        <soap:address
location="http://localhost:8080/WebService/estockincService" />
    </port>
</service>
</definitions>

```

Fig. 2. (continued)

Although the intent of WSDL is to give the syntax of a Web service interface, some level of semantics or meaning is necessary for the interface and its operations to be usable. The real issue is not whether WSDL descriptions themselves have any semantics, but rather how complete and precise are the semantics, and whether the semantics can be effectively and automatically processed.

2.3.2 SAWDL

WSDL as it stands is most useful if standards (naming conventions and even standard predefined interfaces) are used. Then automation is possible if exact matching is used and you are sure everyone has fully followed the standard. Automation tools for discovery and composition may blindly find and connect components. Unfortunately, this brittle solution has only worked in the past in narrow domains or with controlled organizations and is unlikely to scale to the Web.

Table 1. Allowable SAWSDL annotations

	Model Reference	Lifting SchemaMapping	Lowering SchemaMapping
<interface>	Yes	No	No
<operation>	Yes	No	No
<complexType>	Yes	Yes	Yes
<simpleType>	Yes	Yes	Yes
<element>	Yes	Yes	Yes
<attribute>	Yes	No	No
<fault>	Yes	No	No

One could jump to an approach that provides a much richer and more formalized description of Web services (e.g., OWL-S [14]), but maybe a simple augmentation of WSDL may suffice (or at least provide substantial improvement). This is the idea behind WSDL-S [16] and the even simpler Semantic Annotations for WSDL (SAWSDL). As of August 2007, SAWSDL has been accepted as a W3C recommendation or standard for augmenting WSDL and associated XML Schema documents with semantic annotations. Although SAWSDL was designed for WSDL 2.0, which itself was accepted as a W3C recommendation in July 2007, SAWSDL also works with WSDL 1.1 as it is the one currently in predominate use. SAWSDL focuses on the Interface portion of WSDL 2.0 (or PortType in WSDL 1.1) and its sub-elements. Semantics is attached to the principal elements within an interface description, simply by annotating them with concepts from a semantic model (e.g., classes within an OWL ontology). These annotations are innocuous in that they can be easily filtered out, leaving the original WSDL.

There are three types of annotations provided by SAWSDL: model references, lifting schema mappings and lowering schema mappings. The model references tell what an element means in the ontological world, while the mappings allow data to be transformed up (lifted) to the ontological world and returned back down (lowered). Note that these mappings are really descriptions as well, since they need not be applied directly at run time. For example, when one service may need to invoke another, a semantic discovery and composition tool could use these mappings to determine what services can talk to each other. In composition, the mappings could be composed providing transformations from one XSD to another and never actually going up to the ontological world. In Table 1, the SAWSDL annotations are cross referenced with the elements they annotate.

Let us now consider how this information can be used to discover Web services. Note that this information is also useful in the composition of Web services, but that is not the focus of this paper (see [17] for its use in composition). One may reasonably discover Web services by either looking for operations or interfaces. The other elements annotated by SAWSDL are too low level, but of course come into play when looking for operations or interfaces. Let us begin by considering the discovery of operations. The following is a fragment of SAWSDL from the Rosetta Ontology [18].

```
<wsdl:operation name="order"
pattern="http://www.w3.org/2006/01/wsdl/in-out"
sawSDL:modelReference="http://www.w3.org/2002/ws/sawSDL/s
pec/ontology/purchaseorder#RequestPurchaseOrder">
```



```

    <wsdl:input element="OrderRequest" />
    <wsdl:output element="OrderResponse" />
</wsdl:operation>

```

The annotation of the operation named `order` is a model reference to the `RequestPurchaseOrder` class in the `purchaseorder` ontology. This ontology is loosely based on the RosettaNet standard for e-commerce, which includes well-defined operations and sub-operation in their Partner Interface Process (PIP) specifications. In other words, essential functionality is prescribed. One could view this as a high-level description of functionality or in some cases simply as a categorization of functionality. Other aspects of an operation include the inputs and outputs and even preconditions and effects (preconditions and effects are part of WSDL-S, but are initially left out of SAWSDL for simplicity). Next we look at annotations related to the order operation's input.

```

<xs:element name="OrderRequest"
sawSDL:modelReference="http://www.w3.org/2002/ws/sawSDL/spec/ontology/purchaseorder#OrderRequest"
sawSDL:loweringSchemaMapping="http://www.w3.org/2002/ws/sawSDL/spec/mapping/RDFOnt2Request.xml">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="customerNo" type="xs:integer" />
      <xs:element name="orderItem" type="item"
        minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Here the `OrderRequest` element is annotated with `OrderRequest` from the ontology. This reference opens up the richer typing structures of a language like OWL versus XSD (e.g., classes, subclasses, named references and restrictions) as well as inferencing capabilities (e.g., subsumption). Finally, we examine annotations related to the order operation's output.

```

<xs:element name="OrderResponse" type="confirmation" />
  <xs:simpleType name="confirmation"
sawSDL:modelReference="http://www.w3.org/2002/ws/sawSDL/spec/ontology/purchaseorder#OrderConfirmation">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Confirmed" />
      <xs:enumeration value="Pending" />
      <xs:enumeration value="Rejected" />
    </xs:restriction>
  </xs:simpleType>

```

Here the `OrderResponse` element is annotated with `OrderConfirmation` from the ontology. Similar annotations can be provided for faults, while this is likely to be more important for composition than discovery.

Although operation discovery is fundamental, practically speaking one often wishes to invoke multiple operations from a Web service, so in this sense interface discovery is also important. In this paper, we mainly leave this aspect for future work, but of course some of the obvious issues are the following: discovery of a set of operations, temporal dependencies between the operations and statefulness. From a two party point of view these issues are of concern to a conversation protocol, if generalized to multiple parties they are of concern to a choreographer (e.g., following the emerging WS-CDL standard). From the point of view of one of the parties, they can orchestrate their interactions with the other parties (or partners) via a process specification (e.g., following the WS-BPEL standard).

2.3.3 Using Radiant to Add Semantics to WSDL

Radiant [19] is a tool that can be used for marking up Web service descriptions with ontologies. Radiant is a part of an ongoing project, METEOR-S, in an effort to create semantic Web processes, at the LSDIS lab – University of Georgia. This tool provides support for WSDL-S, a joint UGA-IBM specification and SAWSDL. WSDL-S and SAWSDL allow users to easily add semantics to Web services by using the extensibility elements of WSDL. Radiant provides an intuitive UI for annotation of WSDL files using ontologies. All the annotations described in the WSDL-S/SAWSDL specifications are supported by this tool. The framework includes algorithms to match and annotate WSDL files with relevant ontologies using domain ontologies to categorize Web services into domains. A key enabling capability is to achieve annotation with as much automation as possible without losing quality (see [19] to understand how automation is achieved). Figure 3 shows a screenshot of the interface used for annotation. In this figure, the interface provides the user with capabilities of a specifying WSDL file (on the left side) and an ontology (on the right side) used for mapping. The user may then simply drag an element (a class or property) from the ontology on drop it an element in the WSDL file.

While many other efforts have talked about adding semantics to Web services, practical implications of actually annotating Web services with the use of real world applications and ontologies have not been discussed in great detail. Manifestly, there is a lack of real world systems and solutions. The following steps can be followed to annotate Web services using Radiant

1. Start the Eclipse Workbench¹.
2. Open the “Help” menu.
3. Open the “Software Updates” submenu
4. Select “Find and Install”
5. Select the “Search for new features to install” radio button and click next
6. Click “New Remote Site”
7. Enter “<http://lstdis.cs.uga.edu/Radiant/UpdateSite>” without the quotes in the URL box.
8. Enter “Radiant” without quotes for the name field.

¹ <http://www.eclipse.org/>

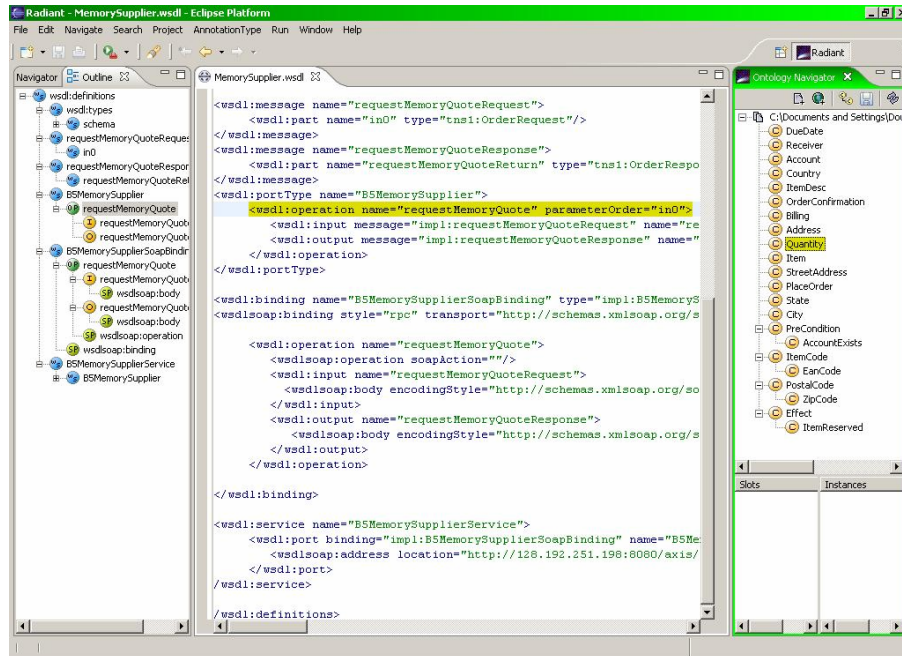




Fig. 3. Radiant tool to annotated WSDL-S and SAWSDL documents

9. Make sure there is a check in the box next to Radiant and click next.
10. Click Finish.
11. When the new dialog box opens, put a check next to Radiant and click next.
12. Select the "I accept terms in the license agreement" radio button and click next.
13. Then follow any onscreen dialogs and the plug-in will be installed.
14. Click on window drop down menu select open perspective and select Radiant.

The Eclipse screen is divided into three parts one is the navigator/outline part, the uddi, wsdl viewer and editor, ontology navigator.

1. Create a new project and open an existing WSDL document.
2. On the ontology navigator load the ontology by clicking on  or .
3. From the Annotation type drop down menu select the annotation type.
4. Click on outline to get the tree view of the WSDL document and select the concept for annotation. Drag the element to the appropriate section of the WSDL tree. The annotations are added to the document automatically.

3 Matching Algorithm for Semantic Web Services

This section presents an algorithm for matching semantic Web services, called SM-T (Semantic Matching Web services using Tversky's model). The algorithm presented computes the degree of match between two output concepts, two input concepts, and two functionality concepts of a service request and advertisement, represented by an

ontology. Given a service request and several advertisements for available Web services, this algorithm can be used to find the more suitable Web services. Web services can be annotated using Radiant [20], as explained previously, and MWSOI [7] and Lumina [21] can use the SM-T algorithm as part of its discovery infrastructure to discover Web services.

We exploit the fact that the input, output, and functionality concepts which are matched may have (in addition to their name) properties (e.g., in the form of attributes) associated with them, and we also take into account the level of generality (or specificity) of each concept within the ontology as well as their relationships with other concepts. Notice that in contrast to semantic-based matching, syntactic-based matching cannot use this information.

Matching input, output, and functionality concepts differs slightly from calculating their semantic similarity. One difference is that the functions to compute the semantic similarity of ontological concepts are usually symmetric, while matching functions are asymmetric [6]. For example, let us assume that SUMO Finance Ontology² in Figure 4 is used to semantically annotate or describe a set of Web services (only an extract of the ontology is shown). The METEOR-S SUMO Finance Ontology was created by converting SUMO financial ontology from KIF to OWL.

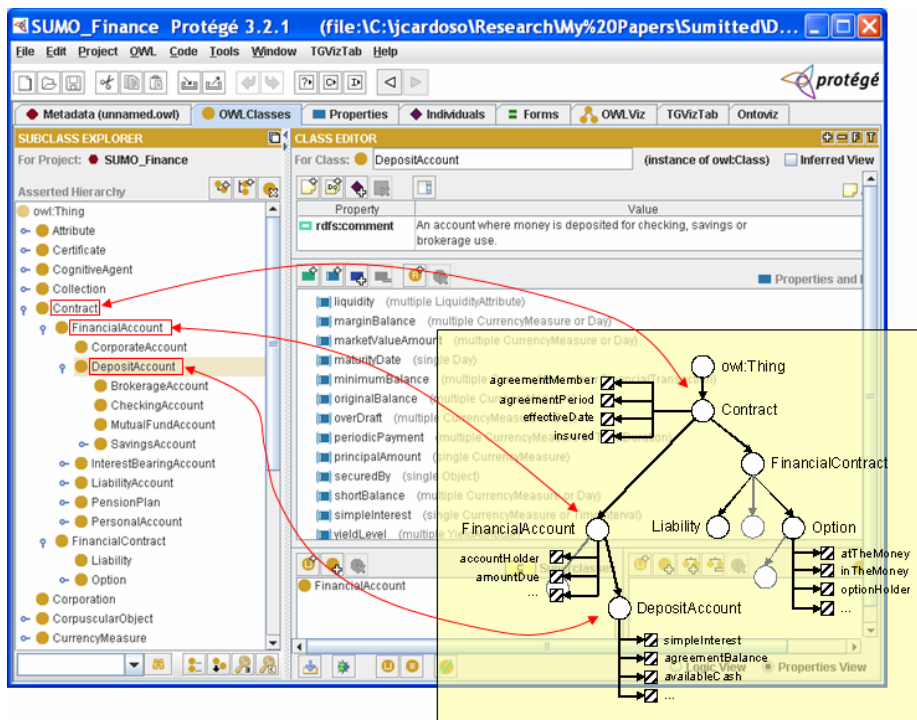


Fig. 4. Example of the SUMO Finance ontology used to semantically annotate a set of Web services

² http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/ontologies/SUMO_Finance.owl

Let us assume that we have a semantic Web service request R with the input concept *FinanceAccount* (c_1) and an advertisement A with the input concept *Contract* (c_2). In this scenario, request R matches advertisement A (i.e., $match(c_1, c_2)=true$), since *FinanceAccount* is a subclass of *Contract*. Our rationale is that if A is able to deal with the input *Contract* it must also be able to deal with the input *FinanceAccount*. We can think that when the Web service is invoked there will be some kind of cast (as in C programming) from *FinanceAccount* to *Contract*. This idea and concept have been first introduced in [22].

Now, let us assume that we have a semantic Web service request R with the input concept *Contract* (c_2) and an advertisement A with the input concept *FinanceAccount* (c_1). In this scenario, it is possible that the semantic Web service A cannot be invoked with the input *Contract* since A may need properties that only exist in the class *FinanceAccount*. Therefore, $match(c_2, c_1)=false$. As we can see from these two scenarios, the function $match$ is asymmetric, since $match(c_1, c_2) \neq match(c_2, c_1)$.

3.1 Formal Definition of a Semantic Web Service

One way to handle functionality of a Web service operation is through preconditions, postconditions and effects. These specifications are usually detailed and precise enough to work at runtime and may be unwieldy for discovery. Usually, they should be specified in a rule language like SWRL or RIF. For discovery, however, there are advantages to sticking with description logic (e.g., OWL). Like other concepts in semantic Web services, a functionality concept is given meaning according to where it stands in a hierarchy and by considering its sub-functions. Of course, a fully detailed specification of sub-functions along with control and data flow could degenerate into a complete specification of the code for the service. What we are looking for is a concise, high-level description that facilitates comparison between services. Consequently, we assign a concept from an ontology to describe the overall functionality of the Web service operation. This functional concept must specialize its parent concept and generalize all of its child concepts.

The functional concept can include component functional concepts (children) which one can think of as carrying out the steps required for the overall functional concept. Again, programmatic level details should not be included, as they would get in the way (similar to the situation in the early and mid phases of software design following a software engineering methodology). The need for annotating inputs and outputs follows the same rationale.

Since we are dealing with input parameters, output parameters, and the functionality of semantic Web services operations (represented with c_i , c_o and c_f , respectively), we define a Web service operation as a finite sequence of ontological concepts as:

$$sws(c_i, c_o, c_f)$$

The number of elements can be other than 3 if we consider more or fewer concepts to be used in a match. The functionality and QoS of Web services [6] can also be considered when matching requests with advertisements. The functions and algorithm that we present can be easily extended to include the notion of functionality, since functionality can be treated in a similar way as inputs or outputs. What the reader

needs to keep in mind is that we always use the Tversky's model [23] to match requests with advertisements, independently of the elements (e.g. inputs, outputs, functionality, QoS, etc) being considered.

3.2 Comparing Semantic Web Services Based on a Single Common Ontology

In this scenario, Web service input, output, and functionality concepts are related to one global and unique ontology providing a common vocabulary for the specification of semantics. Comparing a concept with the ontology is translated into searching for the same or similar concepts within the ontology.

There are several functions that can be adapted and used to compute the degree of match between two input, output, or functionality concepts belonging to the same ontology. The following four main techniques have been identified [24]:

1. **Ontology based approaches.** These approaches [25-27] use an ontology and evaluate the semantic relations among concepts. The most basic metric simply computes the distance between two concepts in an ontology. This corresponds to calculating the distance of nodes in a graph.
2. **Corpus based approaches.** These approaches [28-30] use a corpus to establish the statistical co-occurrence of words. The rationale is that if two words constantly appear together we may conclude that some relation exists between them.
3. **Information theoretic approaches.** These approaches [23, 31-33] consider both a corpora and an ontology, and use the notion of information content from the field of information theory. By statistically analyzing corpora, probabilities are associated to concepts based on word occurrences. The information content for each concept is computed in such a way that infrequent words are more informative than frequent ones. By knowing the information content of concepts it is possible to calculate the semantic similarity between two given concepts.
4. **Dictionary based approaches.** These approaches [34, 35] use a machine readable dictionary to discover relations between concepts. For example, one approach determines the sense of a word in a given text by counting the overlaps between dictionary definitions of the various senses.

Most of these approaches are not suitable to compute the degree of matching between input and output concepts of the semantic Web services. All these metrics are symmetric (except [23]). This means that $f(c_1, c_2) = f(c_2, c_1)$. As explained previously, when matching inputs, outputs and functionality, the matching function needs to be asymmetric.

Furthermore, ontology-based approaches are rather limited since only the taxonomy of the ontology is used to find similarities between concepts. Corpus and dictionary-based approaches require associating a probability with each concept and finding a specific meaning of a word according to the context in which it is found in a dictionary, respectively. These approaches are not simple to implement for Web services. Questions raised include which corpus and dictionaries to use and how to deal with the heterogeneity of Web service discourse domains.

In our opinion, Tversky's model [23] needs to be considered when matching semantic Web services, since it has been considered one of the most powerful similarity

models to date [36]. It is also known as a feature-counting metric or feature-contrast model. This model is based on the idea that common features tend to increase the perceived similarity of two concepts, while feature differences tend to diminish perceived similarity. The model takes into account the features that are common to two concepts and also the differentiating features specific to each. More specifically, the similarity of concept c_1 to concept c_2 is a function of the features common to c_1 and c_2 , those in c_1 but not in c_2 and those in c_2 but not in c_1 . For instance, a truck (Sport Utility Vehicle) and a sedan are similar by virtue of their common features, such as wheels, engine, steering wheel, and gears, and are dissimilar by virtue of their differences, namely the number of seats and the loading capacity.

Based on Tversky's model, we introduce the matching functions $S_i^=(c_R, c_A)$, $S_o^=(c_R, c_A)$ and $S_f^=(c_R, c_A)$ which analyze the number of properties (which may be inherited) shared among two input, output or functionality concepts c_R and c_A (R stands for a Web service request, A stands for a Web service advertisement, i stands for input, o stands for output, and f stands for functionality) conceptualized within the same ontology. In our functions $S^=$, function $p(c)$ retrieves all the properties associated with concept c and function $|s|$ calculates the number of elements in set s . The equal symbol between two concepts (e.g. $c_R=c_A$) indicates that the concepts are the same. The symbol '>' between two concepts (e.g. $c_R>c_A$) indicates that concept c_R is a specialization of concept c_A . Finally, the symbol '<' between two concepts (e.g. $c_R<c_A$) indicates that c_R is a generalization of concept c_A ($c_R<c_A$).

$$S_i^=(c_R, c_A) = \begin{cases} 1, & c_R = c_A \\ 1, & c_R > c_A \\ \frac{|p(c_R)|}{|p(c_A)|}, & c_R < c_A \\ \frac{|p(c_R) \cap p(c_A)|}{|p(c_A)|}, & c_R \neq c_A \end{cases}$$

$$S_o^=(c_R, c_A) = S_f^=(c_R, c_A) = \begin{cases} 1, & c_R = c_A \\ \frac{|p(c_A)|}{|p(c_R)|}, & c_R > c_A \\ 1, & c_R < c_A \\ \frac{|p(c_R) \cap p(c_A)|}{|p(c_R)|}, & c_R \neq c_A \end{cases}$$

Since functions $S_i^=(c_R, c_A)$, $S_o^=(c_R, c_A)$ and $S_f^=(c_R, c_A)$ are very similar we will only describe function $S_i^=$. Four distinct cases can occur:

Case 1: In the first case, since the two input concepts are equal ($c_R=c_A$) their similarity is maximal and therefore the degree of match is one.

Case 2: In the second case, concept c_R is a specialization of concept c_A ($c_R > c_A$). As a result, a Web service with input concept c_A is able to process concept c_R . For example, let us consider the ontology from **Fig. 4**. If a Web service request specifies concept *FinanceAccount* as input and an advertisement specifies concept *Contract* as input then the advertised service is able to process the input concept *FinanceAccount*. This is because the concept c_R is a subclass of concept c_A and it has at least the same set of properties as c_A . In this case, the similarity is also one.

Case 3: In the third case, if the request concept c_R is a generalization of advertisement concept c_A ($c_R < c_A$), then c_A has probably some properties that do not exist in c_R . Therefore, it is possible that a Web service advertisement with input c_A is not able to process the input concept c_R due possibly to missing properties. For example, if a Web service request R specifies concept *Record* as input and an advertisement A specifies concept *FinanceAccount* as input then Web service A may not be able to process the input concept *Contract*. This is because A may need the property *Degree* and *Competencies* of the input concept to work properly.

Case 4: In the last case, concepts c_R and c_A are not equal and do not subsume each other in any way ($c_R \neq c_A$). In this scenario, we evaluate the matching by analyzing how many common properties exist between the two concepts and how many properties are different. Also, we analyze the percentage of input advertisement properties that were satisfied.

As an example, let us illustrate the use of function $S_i^=(c_R, c_A)$ for the four cases – 1), 2), 3) and 4) – that can occur when matching a request c_R with an advertisement c_A . In our example, the Web services' input is annotated with concepts from the ontology illustrated in Figure 4. The four cases that may occur are listed in Table 2 and are evaluated as follows:

- In case 1), both c_R and c_A are associated with the same concept (*FinanceAccount*). Since the request matches the advertisement perfectly. The result is 1.
- In case 2), the request c_R is associated with the concept *FinanceAccount* and the advertisement c_A is associated with the concept *Contract*. Since the concept *Contract* is a generalization of concept *FinanceAccount*, the properties of the concept *FinanceAccount* (the set {*agreementMember*, *agreementPeriod*, *effectiveDate*, *insured*, *accountHolder*, *amountDue*}) is a superset of the properties of the concept *Contract* (the set {*agreementMember*, *agreementPeriod*, *effectiveDate*, *insured*}). All the properties of c_A exist in c_R . As a result, the similarity is evaluated to 1.
- In case 3), the request c_R is associated with the concept *FinanceAccount* and the advertisement c_A is associated with the concept *DepositAccount*. Since the concept *FinanceAccount* is a superclass of concept *DepositAccount*, the properties of the concept *FinanceAccount* (the set {*agreementMember*, *agreementPeriod*, *effectiveDate*, *insured*, *accountHolder*, *amountDue*}) is a subset of the properties of the concept *DepositAccount* (the set {*agreementMember*, *agreementPeriod*, *effectiveDate*, *insured*, *accountHolder*, *amountDue*, *simpleInterest*, *agreementBalance*, *availableCash*}). In this case, when the request c_R matches the advertisement c_A some properties of c_A are left unfulfilled (the properties *simpleInterest*, *agreementBalance*, and *availableCash*). To indicate this mismatch the matching is set to the

ratio of the number of properties of c_R and the number of properties of c_A , which in this case is $|p(c_R)|/|p(c_A)| = 6/9 = 0.67$.

- In the last case (4), the request c_R is associated with the concept *FinanceAccount* and the advertisement c_A is associated with the concept *Option*. The concept *FinanceAccount* has the set of properties {*agreementMember*, *agreementPeriod*, *effectiveDate*, *insured*, *accountHolder*, *amountDue*} and the concept *Option* has the set of properties {*agreementMember*, *agreementPeriod*, *effectiveDate*, *insured*, *atTheMoney*, *inTheMoney*, *optionHolder*}. Since the concepts do not have a parent/children relationship, we compute the percentage by the advertisement's properties that are fulfilled with a property from c_R . The similarity is evaluated as follows:

$$S_i^-(c_R, c_A) = \frac{|p(c_R) \cap p(c_A)|}{|p(c_A)|} = \frac{4}{7}$$

The result of evaluating the function indicates a low degree of matching between the concepts *FinanceAccount* and *Option*. Only one of the three advertisement's properties are satisfied by request properties. The following table shows the results for the four cases presented.

Table 2. An example of matching inputs with a common ontology commitment

Request c_R	Advertisement c_A	$S_i^-(c_R, c_A)$
<i>FinancialAccount</i>	<i>FinancialAccount</i>	1
<i>FinancialAccount</i>	<i>Contract</i>	1
<i>FinancialAccount</i>	<i>DepositAccount</i>	0.67
<i>FinancialAccount</i>	<i>Option</i>	0.57

As we can see, the concept *DepositAccount* is closer to the concept *FinanceAccount* than the concept *Option*. This result corroborates our perception and visual analysis of the ontology and its concepts.

3.3 Comparing Semantic Web Services Based on Multiple Ontologies

In this scenario, different Web services are described by different ontologies. Since there is no common ontology commitment, there is no common vocabulary which makes the comparison of different concepts a more complicated task.

Web service parameters (such as inputs, outputs, and functionality) are identified by words (classes) and there are two major linguistic concepts that need to be considered: synonymy and polysemy. Polysemy arises when a word has more than one meaning (i.e., multiple senses). Synonymy corresponds to the case when two different words have the same meaning. To tackle with the existence of these linguistic concepts we will use a feature-based similarity measure that compares concepts based on their common and distinguishing features (properties).

The problem of determining the similarity of concepts defined in different ontologies is related to the work on multi-ontology information system integration. Most of the similarity measures previously presented [25-35] cannot be directly used to match

Web services since they are symmetric, and more importantly, they can only be used when the concepts to be compared are defined in the same ontology.

Nonetheless, the Tversky's feature-based similarity model [23] is interesting since it takes into account the features or properties of concepts and not the taxonomy that defines the hierarchy of concepts. When matching inputs and outputs, the features of concepts need to be considered, especially when we compare concepts from different ontologies we cannot rely on their taxonomy. One can argue that, in scenarios with different ontologies, we need to take into account the context of ontologies when comparing concepts. In our approach, the context of a concept is transparently represented by its inherited properties.

Based on Tversky's model, we introduce matching functions $S_i^\#(c_R, c_A)$, $S_o^\#(c_R, c_A)$ and $S_f^\#(c_R, c_A)$ for semantic Web services with no common ontology commitment based on the number of properties shared among two input or output concepts c_R and c_A conceptualized within the same ontology. The function computes the geometric distance between the similarity of the domains of concept c_R and concept c_A and the ratio of matched input properties from the concept c_A . Our similarity functions are defined as follows:

$$S_i^\#(c_R, c_A) = \sqrt{\frac{\Pi(p(c_R), p(c_A))}{|p(c_R) \cup p(c_A)| - \Pi(p(c_R), p(c_A))} * \frac{\Pi(p(c_R), p(c_A))}{|p(c_A)|}}$$

$$S_o^\#(c_R, c_A) = S_f^\#(c_R, c_A) = \sqrt{\frac{\Pi(p(c_R), p(c_A))}{|p(c_R) \cup p(c_A)| - \Pi(p(c_R), p(c_A))} * \frac{\Pi(p(c_R), p(c_A))}{|p(c_R)|}}$$

Function Π establishes a mapping between the properties of two concept classes. Figure 5 illustrates two ontologies involved in a mapping.

For example, when matching the class concepts *DepositAccount* and *Deposit* we need to establish a mapping between the properties of the two classes. The mapping is

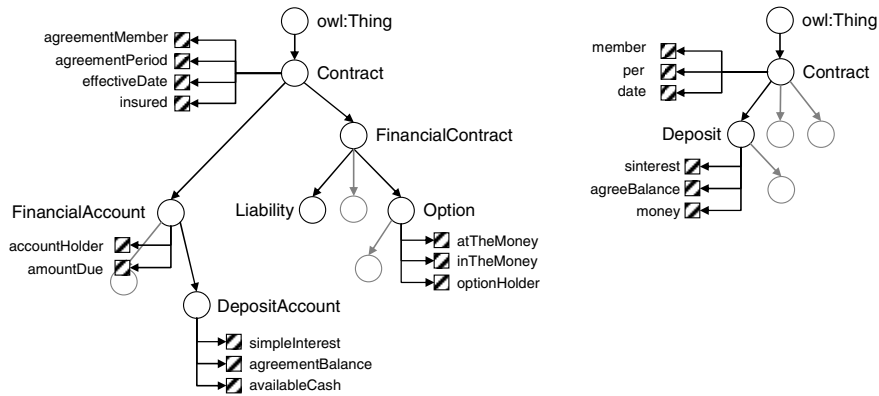


Fig. 5. Two ontologies involved in a mapping

computed with the function $\Pi(p(\text{DepositAccount}), p(\text{Deposit}))$, which is equivalent to $\Pi(\{\text{agreementMember}, \text{agreementPeriod}, \text{effectiveDate}, \text{insured}, \text{accountHolder}, \text{amountDue}, \text{simpleInterest}, \text{agreementBalance}, \text{availableCash}\}, \{\text{member}, \text{per}, \text{date}, \text{sinterest}, \text{agreedBalance}, \text{money}\})$. Possible mappings that can be established are the following:

$$\begin{aligned} \Pi_{i,1}: & (\text{simpleInterest}, \text{sinterest}) \\ \Pi_{i,2}: & (\text{agreementBalance}, \text{agreeBalance}) \\ \Pi_{i,3}: & (\text{availableCash}, \text{money}) \end{aligned}$$

Function Π establishes the best mapping between two sets of properties, pl_1 and pl_2 , and it is defined as follows:

$$\Pi(pl_1, pl_2) = \begin{cases} \text{Max}(\Pi(pl_1 - p_1, pl_2 - p_2) + ss(p_1, p_2)), & ss(p_1, p_2) = 1 \text{ and } pl_1 \neq \emptyset \wedge pl_2 \neq \emptyset \\ \Pi(pl_1 - p_1, pl_2 - p_2), & ss(p_1, p_2) = 0 \text{ and } pl_1 \neq \emptyset \wedge pl_2 \neq \emptyset \\ 0, & pl_1 = \emptyset \vee pl_2 = \emptyset \end{cases}$$

Function $ss(p_1, p_2)$ determines if two properties are considered to be equal using function g . If two properties match syntactically then function ss returns 1, otherwise it returns 0. Properties match syntactically only if function g determines that the syntactic similarity is greater than a constant β .

$$ss(p_1, p_2) = \begin{cases} 1, & g(p_1, p_2) \geq \beta \\ 0, & g(p_1, p_2) < \beta \end{cases}$$

Function $g(p_1, p_2)$ is a function that computes the syntactic similarity of two words. In our approach, we use “string-matching” as a way to calculate similarity. Function g can be implemented using several existing methods such as equality of name, canonical name representations after stemming and other preprocessing, q -grams, synonyms, similarity based on common sub-strings, pronunciation, soundex, abbreviation expansion, stemming, tokenization, etc. Other techniques borrowed from the information retrieval area may also be considered. A very good source of information on retrieval techniques can be found in [37]. Constant β determines the sensibility of the matching. As β gets closer to 1, the matching function returns less false positives. As β gets closer to 0, it returns more false positives.

For example, let us consider the request query with $c_R = \text{“DepositAccount”}$ and an advertisement with $c_A = \text{“Deposit”}$. When computing $\Pi(p(\text{“DepositAccount”}), p(\text{“Deposit”}))$ of these inputs, we obtain value 2. This number represents the two valid mappings obtained:

$$\begin{aligned} \Pi_{i,1}: & (\text{simpleInterest}, \text{sinterest}) \\ \Pi_{i,2}: & (\text{agreementBalance}, \text{agreeBalance}) \end{aligned}$$

Mapping $\Pi_{i,1}$ is found since the results of $ss(\text{“simpleInterest”}, \text{“sinterest”})$ and $ss(\text{agreementBalance}, \text{agreeBalance})$, using the q -grams methodology [38] as an

implementation of g with $\beta = 0.5$, is greater than 0.58 (e.g., $g(\text{"agreementBalance"}, \text{"agreeBalance"})=0.58$). Please refer to [38] to understand this result from applying q -grams. As a result, in both cases ss is evaluated to 1.

All the other mappings are not part of Π . For example, if we compute $ss(\text{"agreementBalance"}, \text{"money"})$ we obtain a result of 0 (function g has a value of 0), which means that we do not consider the properties to be syntactically equal.

The result of computing $S_i^\#(c_R, c_A)$ is done in the following way. The concept *DepositAccount* has 9 properties (i.e., *agreementMember*, *agreementPeriod*, *effectiveDate*, *insured*, *accountHolder*, *amountDue*, *simpleInterest*, *agreementBalance*, *availableCash*) and concept *Deposit* has 6 properties (i.e., *member*, *per*, *date*, *interest*, *balance*, *cash*). Furthermore, $\Pi(p(\text{"DepositAccount"}), p(\text{"Deposit"}))=2$. Applying function $S_i^\#(c_R, c_A)$ we obtain:

$$S_i^\#(c_R, c_A) = \sqrt{\frac{2}{(9+6)-2} * \frac{2}{6}} = \sqrt{\frac{2}{13} * \frac{1}{3}} = \sqrt{\frac{2}{39}} = 0.2265$$

This result corroborates our intuition since only two of the six properties of the concept *Deposit* are satisfied by the properties of concept *DepositAccount*. Furthermore, the concepts *DepositAccount* and *Deposit* are rather distinct since only two properties are shared between the two.

3.4 Ranking Algorithm

In this section we present the actual algorithm for ranking Web service advertisements, following the functions presented previously.

```

REQ(ci, co, cf) = Web service request
ADVj(cji, cjo, cjf) = List of advertisement

For all j get ADVj(cji, cjo, cjf)
If same_ontology(ci, cji) i = Si=(ci, cji)
else i = Si#(ci, cji)

If same_ontology(co, cjo) o = So=(co, cjo)
else o = So#(co, cjo)

If same_ontology(cf, cjf) f = Sf=(cf, cjf)
else f = Sf#(cf, cjf)

match[j] = (i+o+f) / 3;

forall
Sort match[j]
    
```

The algorithm uses the function *same_ontology* that determines if two concepts are defined in the same ontology. Once the matching degree of the input, output, and functionality between a Web service request and a Web service advertisement is calculated, we define the overall degree of the match as the arithmetic mean of the input match degree, output match degree, and functionality match degree. Of course, a weighted function can be implemented if one of the dimensions (inputs, outputs, and functionality) is more important than the others to a service provider or consumer.

4 Using SM-T with METEOR-S WSDI and Lumina

The SM-T algorithm can be integrated in the implementation of METEOR-S Web Services Discovery Infrastructure (MWSDI) [7] and Lumina [21]. One of the authors of this paper was one of the architects of MWSDI and Lumina. Both projects utilize the METEOR-S Discovery API that matches a semantic Template with closely matching Web services that, for example, could be plugged into an abstract process with little or no human intervention. The METEOR-S Discovery API is built on of jUDDI discovery engine and maps semantic information to the business, service and tModel components of UDDI. It thus provides a semantically enhanced UDDI.

4.1 UDDI

UDDI [39], sponsored by OASIS, is an XML-based registry for business and Web services world-wide to list services in the internet. The focus of UDDI is it dynamically allows businesses or enterprises to publish and discover Web services. That is UDDI provides a foundation for both publicly available Web services as well as those which present internally in an organization. UDDI model has persistent data structures called entities expressed in XML and stored in UDDI nodes. The information model is made of the following entity types:

- *businessEntity*: represents an business
- *businessService*: the set of Web services that are provided by a business
- *bindingTemplate*: provides information on how to use a Web service
- *tModel*: gives a technical model categorizing Web service type
- *publisherAssertion*: provides the relationship between business entities
- *subscription*: reports changes in the business entities

The programming interface of UDDI has two parts: inquiry and publishing. To inquire for a Web service through the UDDI several methods are available. The combinations of these search methods can be used through the registry to get optimized results. The methods can be used according to the business of interest by keyword search which gives a set of summarized results for further or deeper search, look for services based on a particular category a business offers and tModel search which returns a set of tModels from different services according to the search criteria. As we go deeper, we can search for the operations a business service offers.

4.2 Approaches to Discovery

Service registries need to provide suitable discovery mechanisms to consumers. We can categorize matchmaking approaches according to various criteria. One possible classification is to take into account what elements are used to match a service advertisement and a service request. We present four approaches: IO matching, multilevel matching, graph-based approaches, and syntactic matching.

IO matching. One of the first works in the field of service discovery (semantic Web service discovery) is described in [40] and [6, 41]. Paolucci [40] follow the idea that “an advertisement matches a request when all the outputs of the request are matched by the outputs of the advertisement, and all the inputs of the advertisement are matched by the inputs of the request”. Cardoso also takes into account the semantic and syntactic similarity of concepts using Tversky model. Thus, these methods takes into account only the inputs and outputs of services during matchmaking. Cardoso and Sheth [6] go a step further and include the QoS of services during the matching process.

Multilevel Matching. Using this matching strategy, presented by Jaeger [42], the matchmaking process is performed at many levels, that is, between inputs/outputs, service categories and other custom service parameters (e.g., related to QoS issues). Such approach reflects the intuition that ideal service discovery should exploit as much of the available functional and non-functional service information as possible.

A Graph-Based Approach. Trastour [43] proposes a semantic graph matching approach. A service description (request or advertisement) is represented as a directed graph (RDF-graph), whose nodes are instances of concepts (i.e., individuals) and arcs are properties (i.e., concept roles) relating such instances. The root node of each graph is the individual representing the service advertisement/request itself. The other nodes refer to concepts borrowed from domain ontologies (capabilities, constraints, etc.). The matchmaking between two graphs, one representing a service request and another representing a service advertisement, is performed with a recursive algorithm.

Syntactic matching. While the IO matching, multilevel matching, and graph-based matching rely on exploiting the subsumption relations in various ontologies in order to assess the similarity of services, service capabilities, this is not sufficient to enable an effective discovery. One extension that can be made is to use similarity measures and information retrieval (IR) techniques. The objective is to use implicit semantics of services, besides the explicit semantics that are described by the domain ontologies. The core idea in this approach is that IR similarity measures could be applied when logic-based (subsumption) matching fails. For example, TFIDF (Term Frequency/Inverse Document Frequency) term weighting schemes [44] can be used to evaluate the semantic distance/closeness between concepts, words or documents.

4.3 Lumina

The focus of Lumina works closely with MWSDI [7] to provide a user friendly GUI for specifying semantic templates and discovering matching services. MWSDI is an infrastructure that addresses the challenge of integrating a large number of registries

from diverse domains. MWSDI supplies an infrastructure of registries for semantic publication and discovery of Web services. The primary motivation was the expected growth in the number of registries and the lack of semantics in Web service representation. The system provides a scalable architecture to access such registries. In addition, it provides semantic publication and discovery capabilities by using a domain specific ontology for each registry. Two algorithms are made available for semantic publication and discovery using WSDL descriptions. Both these algorithms map inputs and outputs of Web services to ontological concepts. Subsequently, searching can be carried out using constructed templates using the ontological concepts.

MWSDI was implemented with an underlying peer-to-peer network which gives the scalability and flexibility required for creating an infrastructure for diverse Web service registries.

Lumina may be viewed as Radiant's companion. While Radiant annotates and publishes semantic Web services, Lumina is used for discovering these published services. It allows to search for services, individual operations or interfaces (i.e., combinations of operations). In order to create a semantic template, the GUI provides input text boxes and selections that can be filled in by data entry, mouse clicking or dragging a class or property from an ontology. Figure 6 illustrates how to fill in a semantic template using Lumina.

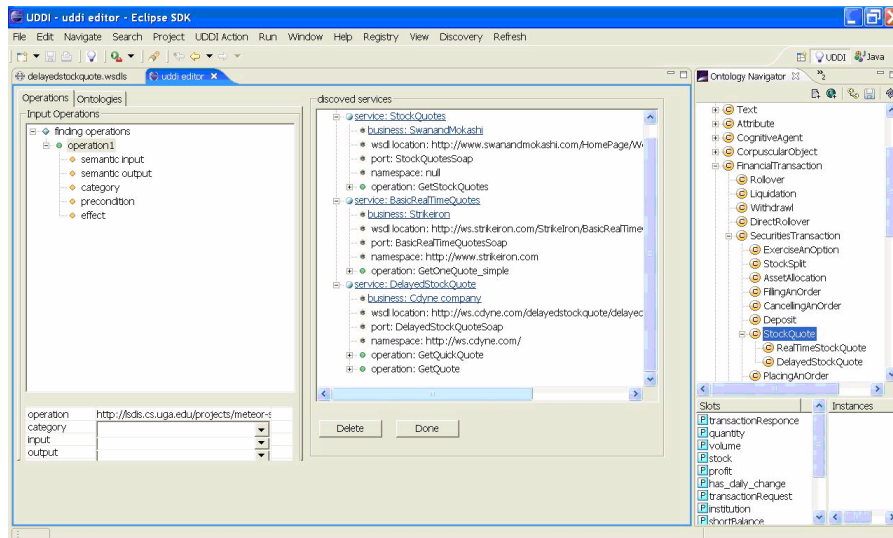


Fig. 6. Semantic template using Lumina

Lumina was designed to support WSDL-S and this provides a means for specifying inputs, outputs, functionality/category, preconditions and effects. Later a simplified SAWSDL mode was added that does not support preconditions and effects.

SM-T, MWSDI and Lumina basically follow the same approach concerning their vision of Web services. They all treat a Web service as an abstract interface (black box) consisting of multiple operations which each having its own set of inputs and set

of outputs as well as functionality. Annotating the inputs, outputs and functionality of Web service operations gives a significant improvement in discovery and is better than the approach used by current UDDI registries. This is because current UDDI implementations are only based on the syntactic matching of properties. Semantic approaches have already shown in several domains to improve search precision. Section 3.2 and section 3.3 show that the SM-T algorithm is able to compare concepts beyond a simple syntactic match. Let us assume that a user issues a request to a UDDI registry for a service with an input *FinanceAccount* (see Section 3).

Let us also assume that the registry has only an advertisement with the input *Contract*. In such a case, the registry informs the user that no Web service matching the search criteria was found. This search was based solely on a syntactic analysis. Now, let us assume that Web services descriptions are annotated with ontological concepts. The two Web services' inputs, *FinanceAccount* and *Contract*, are annotated with concepts with the same name in the ontology shown in Figure 4 (the names are the same for simplicity reasons, they could be different). Using this ontology, a semantically enhanced UDDI registry can use the ontological information to improve the search.

In such a situation, the results of the match would return a Web service since the concept *Contract* is a generalization of concept *FinanceAccount*. That is, the properties of the concept *FinanceAccount*,

{agreementMember, agreementPeriod, effectiveDate, insured, accountHolder, amountDue}

are a superset of the properties of the concept *Contract*,

{agreementMember, agreementPeriod, effectiveDate, insured}.

Since all the properties of *Contract* exist in *FinanceAccount*, there is a match and a reference to the Web service is returned to the user. This example shows that different concepts from the same ontology can be matched by our algorithm even when their properties do not match semantically. The example shown in section 3.3 also illustrates that two concepts from two different ontologies can be matched by our algorithm even if their properties do not match syntactically.

From the business perspective SM-T, MWSOI, and Lumina are all about grouping services and distributing them in different registries based on domain knowledge, for locating the right services easily. On the other hand, from the technical perspective, SM-T, MWSOI, and Lumina can provide a scalable infrastructure for accessing multiple registries and semantic enhancements to current service discovery mechanisms. We believe that to develop processes in the current network economy [45], architectures and algorithms like SM-T, MWSOI, and Lumina will drive the evolution of businesses' interactions using Web services. This infrastructure will also help Web services by changing the focus from a static to a more dynamic business settings. To discover Web services using Lumina the following steps can be followed:

- Download Lumina and install it as the eclipse plug in. Radiant has to be installed before installing Lumina.
- The screen is divided into six parts: Navigator/Outline, UDDI editor WSDL editor, Information list, Discovered results and Ontology navigator.

- Follow the same steps as for Radiant to load the ontology, create a new project and WSDL.
- Click on the Registry drop down menu and select registry. A window pops up. Add a new registry and connect.
- Click on the Publish menu and publish a business followed by the WSDL.
- In the UDDI editor select the operation, input, and output according to what you want to discover the web service and drag and drop the concept on them from the ontology navigator.
- At the information list the selected operations or IOPEs will be displayed select on them and click on discover.
- The web services discovered will be displayed on the discovered results pane.

5 Related Work

The discovery of services “boils down” to determining the similarity of services’ properties which are typically annotated with ontological concepts. In the literature we can find four distinct approaches to calculate the semantic relations among concepts. In [25-27], ontology based approaches are presented. The most basic metric simply computes the distance between two concepts in an ontology. Corpus based approaches are described in [28-30]. These approaches use a corpus to establish the statistical co-occurrence of words. Information theoretic approaches [23, 31-33] consider both a corpora and an ontology, and use the notion of information content from the field of information theory. By statistically analyzing corpora, probabilities are associated to concepts based on word occurrences. Dictionary based approaches [34, 35] use a machine readable dictionary to discover relations between concepts. For example, one approach determines the sense of a word in a given text by counting the overlaps between dictionary definitions of the various senses.

Some of the above approaches, to calculate the semantic relations among concepts, have been used to deploy discover algorithm for semantic Web services. The OWL-S/UDDI Matchmaker [46] introduces semantic search into the UDDI directory by embedding an OWL-S Profile in a UDDI data structure, and augmenting the UDDI registry with an OWL-S matchmaking component. The matching algorithm recognizes four degrees of match between two concepts defined in the same ontology: (1) *exact*, (2) *plug in*, (3) *subsume*, and (4) *fail*. The function used by the algorithm is asymmetric and is based on the existence of relationships between concepts. When no direct relationship exists among two concepts the algorithm simply returns *fail*. Unlike the algorithm presented in this paper, the OWL-S/UDDI Matchmaker searches for services based on inputs and outputs within the IOPEs (Input, Output, Precondition, and Effect) of the profile which must belong to the same ontology. Our approach allows evaluating the similarities of IOPE that are annotated with concepts from distinct ontologies.

The METEOR-S [20] Web Service Annotation Framework (WSAF) allows semi-automatically matching WSDL concepts (such as inputs and outputs) to DAML and RDF ontologies using text-based information retrieval techniques (for example, synonyms, n-grams and abbreviation). The strength of matches (SM) is calculated using a scoring formula which involves element (ElemMatch) and structure level schema (SchemaMatch) matching. The ElemMatch function performs the element level matching

based on the linguistic similarity of the names of the two concepts. The SchemaMatch function examines the structural similarity between two concepts. A concept in an ontology is usually defined by its properties, superclasses and subclasses. Since concept labels are somewhat arbitrary, examining the structure of a concept description can provide more insight into its semantics. In WSAF, the XML representation of WSDL is matched against the concepts of a given ontology. The best match between WSDL and ontological concepts are returned to users as a suggestion of potential mappings. It should be noticed that the work presented in [20] cannot be easily adapted to our problem. There are several reasons for this. First, the weight values for calculating the MS function were set without empirical testing and validation. Also, the weights are not defined for a set of ElemMatch and SchemaMatch values. For example, if $0.5 < \text{ElemMatch} < 0.65$ then no weights are suggested. Furthermore, the function that computes the ElemMatch of a WSDL concept and an ontological concept is not defined when the MatchScore is other than zero, but is less than one, using the NGram or Synonym matching algorithms.

In [47], the authors present a hybrid approach to Semantic Web service matching. The hybrid matchmaker, called OWLS-MX, is to be used to find service requests specified in OWL-S. OWLS-MX can be seen as an extension of the OWL-S/UDDI Matchmaker presented in [46]. Their approach is somewhat similar to our in that they “complement logic based reasoning with approximate matching based on syntactic information retrieval (IR) based similarity computations”. The IR based methods used include: the extended Jacquard similarity coefficient, the cosine similarity value, and the Jensen-Shannon information divergence based similarity value. Our approach differs in the sense that we have used *q-grams* for syntactic matching. But this is only a minor difference since, as we have explained previously in section 4.3, in our approach other syntactic matching functions can be used such as: soundex, abbreviation expansion, stemming, tokenization and other techniques borrowed from the information retrieval (see [37].), including the matching function used by OWLS-MX. The major difference in our work lies on the use of the Tversky’s model. While OWLS-MX mainly compares concepts syntactically when the logic-based comparison fails, in our approach we compare syntactically, not the concepts themselves, but the properties of the concepts. For example, if the concepts ‘car’ and ‘automobile’ are compared using OWLS-MX and the concepts are not related with a parent-child relationship (i.e., an *exact*, *plug-in*, or *subsumes* relationship is not found), the algorithm will answer fail, meaning that there is no match. Using SM-T, the algorithm will try to syntactically match the properties of the concepts. Therefore, if the concept ‘car’ has the properties: ‘engine’, ‘body’ and ‘wheels’ and the concept ‘automobile’ has the properties: ‘bigengine’, ‘car_body’ and ‘fourwheels’, the SM-T algorithm will indicate that there is a partial match (this will be expressed with a normalized value).

6 Conclusions

In this paper we have described a semantic matching algorithm to be used by UDDI registries enhanced with semantics. Our algorithm can work with Web services described with WSMO and OWL-S, or annotated with SAWSDL (previously WSDL-S). Compared to previous work [46], we do not limit the classification of the accuracy of matching a request with an advertisement using a four value schema (i.e. *exact*, *plug*

in, *subsume*, and *fail*). The accuracy of matching is assessed with a continue function with the range [0..1]. Furthermore, and compared to [46], we allow the matching of semantic Web services both with and without a common ontology commitment. This aspect is important since it is not realistic to assume that Web services will always be defined by the same ontology. In some cases, similar services may be defined by different ontologies. Furthermore, we take into account functionality.

Our algorithm relies on Tversky's feature-based similarity model to match requests with advertisement. This model takes into account the features or properties of ontological concepts and not the taxonomy that defines the hierarchy of concepts. We believe that when matching inputs, outputs and functionality, the analysis of features of concepts is fundamental when matching concepts from different ontologies, since they typically have distinct taxonomies. The matching process can be easily extended to include non-functional capabilities of services.

References

1. Chinnici, R., et al.: Web Services Description Language (WSDL) Version 1.2, W3C Working Draft 24 (2003)
2. Chinnici, R., et al.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language (2006), <http://www.w3.org/TR/wsd120/>
3. UDDI. Universal Description, Discovery, and Integration (UDDI v3.0) (2005), <http://www.uddi.org/>
4. SOAP. Simple Object Access Protocol 1.2 (2003), <http://www.w3.org/TR/soap12-part1/>
5. Cardoso, J., Sheth, A.P.: Introduction to Semantic Web Services and Web Process Composition. In: Cardoso, J., Sheth, A.P. (eds.) *Semantic Web Process: powering next generation of processes with Semantics and Web services*, pp. 1–13. Springer, Heidelberg (2005)
6. Cardoso, J., Sheth, A.: Semantic e-Workflow Composition. *Journal of Intelligent Information Systems (JIIS)* 21(3), 191–225 (2003)
7. Verma, K., et al.: METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management (ITM)*, Special Issue on Universal Global Integration 6(1), 17–39 (2005)
8. Curbera, F., Ehnebuske, D., Rogers, D.: Using WSDL in a UDDI Registry, Version 1.07, UDDI Best Practice, May 21 (2002), <http://www.uddi.org/pubs/wsd1bestpractices-V1.07-Open-20020521.pdf> (Retrieved October 12, 2006)
9. Sheth, A., Meersman, R.: Amicalola Report: Database and Information Systems Research Challenges and Opportunities in Semantic Web and Enterprises. *SIGMOD Record* 31(4), 98–106 (2002)
10. Smeaton, A., Quigley, I.: Experiment on Using Semantic Distance Between Words in Image Caption Retrieval. In: 19th International Conference on Research and Development in Information Retrieval SIGIR 1996, Zurich, Switzerland (1996)
11. Rodríguez, A., Egenhofer, M.: Determining Semantic Similarity Among Entity Classes from Different Ontologies. *IEEE Transactions on Knowledge and Data Engineering* 15(2), 442–456 (2002) (in press)
12. Klein, M., Bernstein, A.: Searching for Services on the Semantic Web Using Process Ontologies. In: *International Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA (2001)

13. Cardoso, J., et al.: Academic and Industrial Research: Do their Approaches Differ in Adding Semantics to Web Services. In: Cardoso, J., Sheth, A. (eds.) *Semantic Web Process: powering next generation of processes with Semantics and Web services*, pp. 14–21. Springer, Heidelberg (2005)
14. Martin, D., et al.: Bringing Semantics to Web Services: The OWL-S Approach. In: Cardoso, J., Sheth, A.P. (eds.) *SWSWPC 2004*. LNCS, vol. 3387. Springer, Heidelberg (2005)
15. Roman, D., et al.: WWW: WSMO, WSML, and WSMX in a nutshell. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) *ASWC 2006*. LNCS, vol. 4185. Springer, Heidelberg (2006)
16. Akkiraju, R., et al.: *Web Service Semantics - WSDL-S (2006)*, <http://www.w3.org/Submission/WSDL-S> (Retrieved October 10, 2006)
17. Sivashanmugam, K., et al.: Framework for Semantic Web Process Composition. *International Journal of Electronic Commerce (IJEC)*, Special Issue on Semantic Web Services and Their Role in Enterprise Application Integration and E-Commerce 9(2), 71–106 (2004-2005)
18. Farrell, J., Lausen, H.: *Semantic Annotations for WSDL (2006)*, <http://www.w3.org/2002/ws/sawSDL/spec/SAWSDL.html>
19. Gomadam, K., et al.: Radiant: A tool for semantic annotation of Web Services. In: 4th International Semantic Web Conference ISWC 2005, Galway, Ireland (2005)
20. Patil, A., et al.: MWSAF - METEOR-S Web Service Annotation Framework. In: 13th Conference on World Wide Web, New York City, USA (2004)
21. Cardoso, J., Sheth, A.: Semantic Web Services, Processes and Applications. In: Jain, R., Sheth, A. (eds.) *Semantic Web and Beyond: Computing for Human Experience*. Springer, Heidelberg (2006)
22. Paolucci, M., et al.: Semantic Matching of Web Services Capabilities. In: Horrocks, I., Hendler, J. (eds.) *ISWC 2002*. LNCS, vol. 2342. Springer, Heidelberg (2002)
23. Tversky, A.: Features of Similarity. *Psychological Review* 84(4), 327–352 (1977)
24. Zavaracky, A.: Glossary-Based Semantic Similarity in the WordNet Ontology, in Department of Computer Science, University College Dublin, Dublin (2003)
25. Wu, Z., Palmer, M.: Verb Semantics and Lexical Selection. In: 32nd Annual Meeting of the Association for Computational Linguistics (ACL 1994), Las Cruces, New Mexico (1994)
26. Rada, R., et al.: Development and Application of a Metric on Semantic Nets. *IEEE Transactions on Systems, Man, and Cybernetics* 19(1), 17–30 (1989)
27. Leacock, C., Chodorow, M.: Combining local context and WordNet similarity for word sense identification. In: Fellbaum, C. (ed.) *WordNet: An Electronic Lexical Database*, pp. 265–283. MIT Press, Cambridge (1998)
28. Turney, P.D.: Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In: 12th European Conference on Machine Learning. Springer, Heidelberg (2001)
29. Keller, F., Lapata, M.: Using the Web to Obtain Frequencies for Unseen Bigrams. *Computational Linguistics* (2003)
30. Church, K.W., Hanks, P.: Word association norms, mutual information, and Lexicography. In: Vancouver, B.C. (ed.) 27th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Vancouver (1989)
31. Lin, D.: An information-theoretic definition of similarity. In: 15th International Conf. on Machine Learning. Morgan Kaufmann, San Francisco (1989)
32. Resnik, P.: Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In: 14th International Joint Conference on Artificial Intelligence (1995)

33. Jiang, J., Conrath, D.: Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In: International Conference on Computational Linguistics (ROCLINGX), Taiwan (1997)
34. Lesk, M.: Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: 5th annual international conference on Systems documentation. ACM Press, New York (1986)
35. Banerjee, S., Pedersen, T.: Gloss Overlaps as a Measure of Semantic Relatedness. In: Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico (2003)
36. Richardson, R., Smeaton, A.: Using WordNet in a Knowledge-Based Approach to Information Retrieval. Dublin City University/School of Computer Applications, Dublin, Ireland (1995)
37. Belew, R.K.: Finding Out About: A Cognitive Perspective on Search Engine Technology and the WWW, p. 356. Cambridge University Press, Cambridge (2000)
38. Salton, G.: Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer. Addison-Wesley, Massachusetts (1988)
39. UDDI. UDDI Spec. Technical Committee, UDDI Version 3.0.2, (2004), http://uddi.org/pubs/uddi_v3.htm
40. Paolucci, M., et al.: Semantic matching of Web services capabilities. In: First International Semantic Web Conference on the Semantic Web, Sardinia, Italy. LNCS. Springer, Heidelberg (2002)
41. Cardoso, J.: Quality of Service and Semantic Composition of Workflows, in Department of Computer Science, p. 215. University of Georgia, Athens (2002)
42. Jaeger, M.C., Tang, S.: Ranked matching for service descriptions using DAML-S. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084. Springer, Heidelberg (2004)
43. Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A Semantic Web approach to service description for matchmaking of services. In: The first Semantic Web Working Symposium, California, USA (2001)
44. Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string distance metrics for name-matching tasks. In: Kurumatani, K., Chen, S.-H., Ohuchi, A. (eds.) IJCAI-WS 2003 and MAMUS 2003. Springer, Heidelberg (2003)
45. Sheth, A.P., v.d Aalst, W., Arpinar, I.B.: Processes Driving the Networked Economy. IEEE Concurrency 7(3), 18–31 (1999)
46. Srinivasan, N., Paolucci, M., Sycara, K.: An efficient algorithm for OWL-S based semantic search in UDDI. In: Cardoso, J., Sheth, A. (eds.) Lecture Notes in Computer Science. Springer, Heidelberg (2005)
47. Klusch, M., Fries, B., Sycara, K.: Automated Semantic Web Service Discovery with OWLS-MX. In: Alonso, E., Kudenko, D., Kazakov, D. (eds.) AAMAS 2000 and AAMAS 2002. ACM Press, New York (2006)