Discovering Web Services Using Semantic Keywords

Mauricio Espinoza and Eduardo Mena

Abstract— **With the increasing growth in popularity of Web services, the discovery of relevant services becomes a significant challenge. In order to enhance the service discovery is necessary that both the web service description and the request for discovering a service explicitly declare their semantics. Some languages and frameworks have been developed to support rich semantic service descriptions and discover using ontology concepts. However, the manual creation of such concepts is tedious and error-prone and many users accustomed to automatic tools might not want to invert his time in obtaining this knowledge.**

In this paper we propose a system that assists to both service producers and service consumers in the discovery of semantic keywords which can be used to describe and discover web services respectively. First, our system enhances semantically the list of keywords extracted from the elements that comprise the description of a web service and the user keywords used for discover a service. Second, an ontology matching process is used to discovers matchings between the ontological terms of a service description and a request for service selection. Third, a subsumption reasoning algorithm tries to find service description(s) which match the user request.

I. Introduction

WEB services have received much attention and interest in the software industry due to their potential in facilitating seamless business-to-business or enterprise application integration [13]. Given the dynamic environment in e-business, the power of being able to find Web services on the fly to create business processes is highly desirable.

Currently, the industry standards for Web services are WSDL (Web Services Description Language) [17] and UDDI (Universal Description Discovery and Integration) [15] specifications. Web services are described using WSDL definitions and advertised in UDDI registries. Unfortunately, discovering Web services using UDDI is relatively inefficient since the discovery mechanism does not support searches at a semantic level. For instance, if a user is looking for a software vendor service, a service called "system application" would not match with the request although its meaning is perfectly compatible with the meaning of the requested service. The goal of current Semantic Web Services initiatives such as WSDL-S [19], OWL-S [16] or WSMO [18] is adding semantics to Web services to match the expectations of the consumer of Web services. More precisely, semantic Web services are described in terms of concepts provided by a domain ontology. These concepts denote entities in the domain of the Web service (e.g., Software, Operating System) as well as functionalities that can be performed by services in the given domain (e.g., OrderSoftware, BuySoftware). However, the manual creation of such concepts is tedious and error-prone and many users accustomed to automatic tools might not want to spend his time in obtaining this knowledge.

This work is supported by the CICYT project TIN2004-07999-C02-02.

Mauricio Espinoza is with the IIS Department at the University of Zaragoza, María de Luna 1, 50018 Zaragoza, Spain (mespinoz@unizar.es). Work supported by a Santander Central Hispano-University of Zaragoza grant.

Eduardo Mena is a faculty member of the IIS Department at the University of Zaragoza, María de Luna 1, 50018 Zaragoza, Spain (emena@unizar.es).

In this paper we propose a system that discovers semantic keywords which can be used by both service producers and service consumers to describe and discover Web services, respectively. First, our system takes as input a list of keywords extracted from the elements that comprise the description of a Web service and the user keywords used to discover a service. This keywords are enhanced semantically by consulting the knowledge represented by many (heterogeneous and distributed) ontologies. Second, an ontology matching process which relies in a measure based on statistics techniques and ontological similarity is used to discover matchings between the concepts of a service description and the enhanced keywords of a request for service selection. Third, a subsumption reasoning algorithm tries to find service description(s) that match the user request. Each of these matches are individually scored, thus the result is a set of candidate services which are ordered by their degree match.

The rest of this paper is as follows. In Section II we overview our approach and describe the main steps performed by the system. In Section III we describe how the semantics of a set of keywords is discovered. In Section IV we show how the semantic keywords obtained with our system can be used to semantically describe and discover Web services. Related work can be found in Section V. Finally, conclusions and future work appear in Section VI.

II. Overview of the System

As motivating example, let us suppose that there exists a software vendor service (specialized in computer assisted design software), which allows users to performs two actions: 1) to get a software product offer (getSoftwareOffer) and, 2) to buy some of his software products (buySoftware). In order to perform the first operation "getSoftwareOffer", the Web service may expect the type of *CAD software*, and a *delivery address* as its *inputs*. This operation returns an email confirmation and an offer with properties as *software number*, the *operating system* (in which the application will be executed), and *price* (its outputs). On the other hand the operation "buySoftware" will require a sufficient credit balance for invoking the service (its preconditions), whereas, charging the credit card for the appropriate amount of money and delivered the software product are the *effects* after a successful invocation of the service.

Now, let us suppose that an enterprise needs to buy some software products including a system to support tasks of computer assisted design. According to the enterprise policies, for purchasing software is required to collect offers from various software vendors based on the characteristics of the wanted software product. It is quite conceivable that the consumer might have his/her own understanding of the domain in discourse and hence expressing his knowledge using some keywords for describe his requirements. Notice that in order to obtain more accurate and relevant results in the Web service discovery, both the service provider and the service consumer need to add semantics to their corresponding descriptions. However, few domain ontologies for Web service descriptions exist and building of ontology concepts from scratch is a challenging task [11].

In this paper we propose a system that assists to users in the discovery of semantic keywords which can be used to describe and discover Web services. Figure 1 shows the main steps of our system. In the figure the process are illustrated by a rectangle and the person icons representing the logical role of the person who executes the corresponding process. We summarized the main steps in the following.

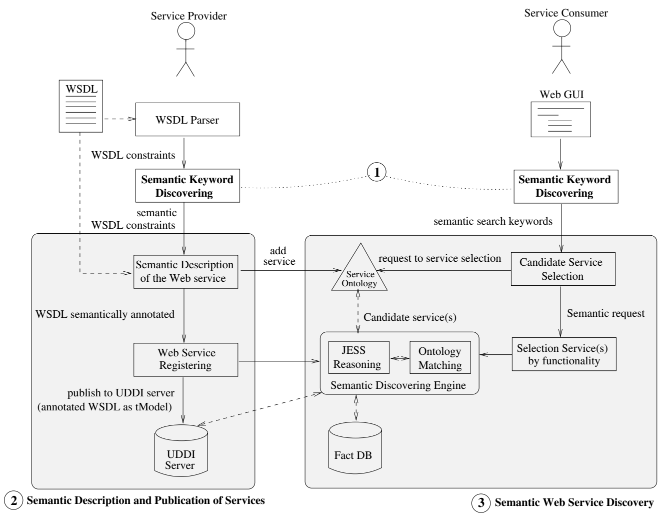

Service Provider
Service Consumer
WSDL
WSDL Parser
Web GUI
WSDL constraints
Semantic Keyword Discovering
1
Semantic Keyword Discovering
semantic WSDL constraints
semantic search keywords
Semantic Description of the Web service
add service
request to service selection
Service Ontology
Candidate Service Selection
WSDL semantically annotated
Candidate service(s)
Semantic request
Web Service Registering
JESS Reasoning
Ontology Matching
Selection Service(s) by functionality
Semantic Discovering Engine
publish to UDDI server (annotated WSDL as tModel)
UDDI Server
Fact DB
2 Semantic Description and Publication of Services
3 Semantic Web Service Discovery

Fig. 1. Main steps of our system for discovery of semantic Web services.

- *Semantic Keyword Discovering.*
  The system starts with a Web service description (*WSDL file*) provided by the *service provider*. The *WSDL parser* step parses the WSDL and returns a list of keywords that corresponds to elements that describe the service (e.g. inputs, outputs, preconditions, effects, or service name). The *semantic keyword discovering* process takes as input the list of keywords extracted from the previous step and discovers their possible meanings by consulting the knowledge represented by many (heterogeneous and distributed) ontologies. These keyword senses are semantically enriched with the synonym terms found during the ontology matching process: A synonymy measure based on statistics techniques and ontological similarity is used to integrate senses that are similar enough. Finally a disambiguation method is used to select the most probable intended meaning of each keyword by considering its context. A more detailed description of this process is presented in Section III.
- *Semantic Description and Publication of Web Services.*
  In order to support semantic discovery of Web services, the *semantic description of the Web service* step, annotates automatically the service description with the semantic keywords obtained by the system. The service provider can decide to confirm the result (a WSDL semantically annotated) and proceed directly to Web service registering step (described below). Alternatively, he can modify the semantic annotations that he considers convenient. Furthermore, the elements used to describe the service (keywords semantically enhanced) are added to the *service ontology*. When the ontology receives a service description, it creates a new concept and fills its properties with the characteristics extracted of the service, then it asserts this new concept using a OWLJessKB [6] reasoner, which is on charge of classifying it in ontology. These keywords will be queried later by the service consumer in order to obtain a first list of candidate services. Finally, the *Web service registering* step parses the semantic service description (using Jena API [5]) and converts it into a collection of JESS [12] facts, which are stored as triples ($< subject, predicate, object >$) in the Fact DB. Additionally, the Web service is published in a UDDI registry and can be dynamically discovered using some ontological concepts[1]. A more detailed description of these steps that semantically describe and publish a Web service is presented in Section IV-A.
- *Semantic Web Services Discovery.*
  In order to discover a Web service semantically described, the service consumer expresses his requirements and preferences by providing a list of keywords. In the *semantic keyword discovering* step, each keyword provided by the user to perform the search is semantically enhanced by ontological terms. Notice that the same process is used to add semantics to the elements that describe a Web service. In the first phase of the discovery of services, the *candidate service(s) selection* step matches Web services considering the characteristics that they provide by consulting the *service ontology*. The candidate services returned are stored in the *semantic discovering engine*. The *selection service(s) by functionality* step transforms the requirements of the user into a semantic template (as suggested by [8]) which captures the abstract functionality of a Web service request. This information is send to *semantic discovering engine* in order to obtains services that are semantically related to concepts in the template. A *ontology matching* process which relies in a measure based on statistics techniques and ontological similarity is used to discover matchings between the concepts used to describe a service and the concepts used to discover a service. Finally, a *subsumption reasoning algorithm* tries to find service description(s) which match with the service request. Each of these matches are individually scored, thus the result is a set of candidate services which are ordered by their degree match. In the Section IV-B

[1]The details about how to publish a semantically annotated WSDL into UDDI registry is out the scope of this paper.

we detail the steps performed by the *semantic discovery engine* in order to discover a semantic Web service.

III. Discovering the Semantics of Keywords

In this section we explain the semantic keyword discovering process which is used in two situations by our system: 1) to describe semantically some elements of a Web Service, and 2) to enhance the keywords used to find appropriate services. 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.

In a previous work [3], we proposed a system that takes as input a list of plain keywords provided by the user, discovers their semantics in run-time and obtains a list of senses extracted from different ontology pools; it deals with the possible overlapping among senses. For efficiency purposes, the system uses sampling and other statistic techniques, as well as parallel processing, whenever possible. Due to space limitations, in the following we summarize the main steps of our approach (see Figure 2). Notice that the whole process can be limited in time. A more detailed description of this process can be found in [3].

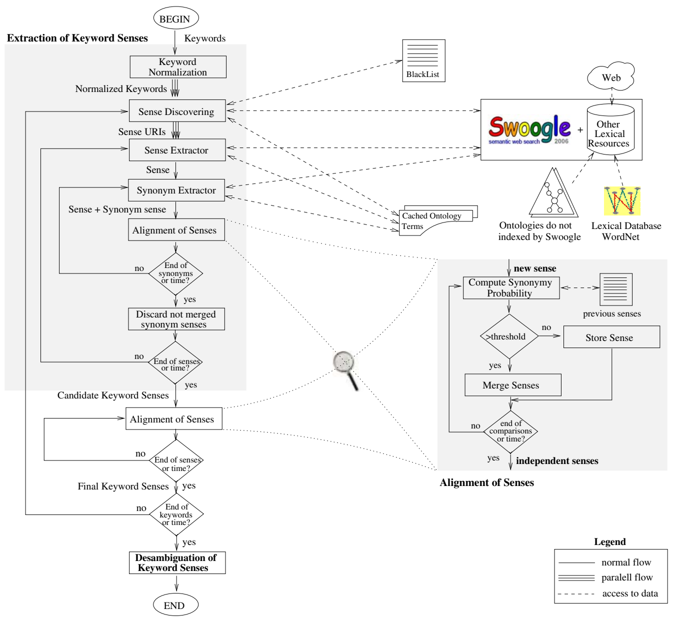


Fig. 2. Obtaining the possible senses of a set of keywords.

A. Extraction of Keyword Senses.

First, the user keywords are normalized by a preprocessing step (e.g., rewriting them in lowercase, removing hyphens, etc.), and in order to discover the semantics of the user keywords, the system accesses to the shared knowledge stored in different ontology pools available on the Web. The extracted senses are semantically enriched with the ontological senses of their synonyms (which are obtained from the ontology pool), whenever the system evaluates that the synonym senses matches to the semantics of the corresponding keyword sense.

In the Figure 3, we show the first three steps in order to extract the senses of the sample keywords that corresponding to the inputs of the operation "getSoftwareOffer" introduced in the Section II. For reasons of simplicity we will restrict our focus on Web service input parameters, however this process is applied to other elements too (e.g. service name, outputs, preconditions, or effects). Our prototype finds 10 URIs for the keyword "CAD software" and 18 for "deliveryaddress", but due to space limitations we only show three.

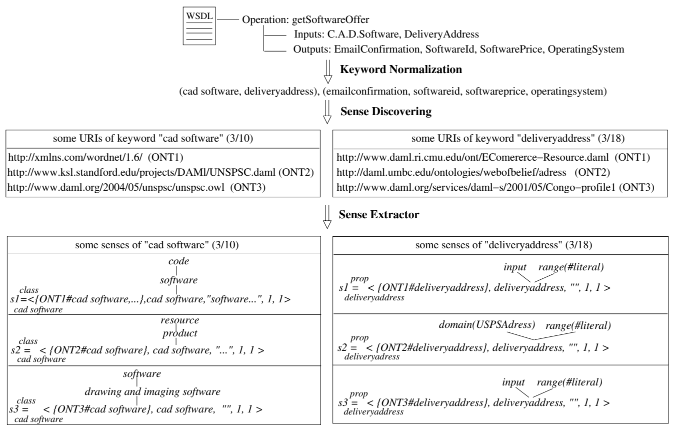

WSDL
Operation: getSoftwareOffer
Inputs: C.A.D.Software, DeliveryAddress
Outputs: EmailConfirmation, SoftwareId, SoftwarePrice, OperatingSystem
Keyword Normalization
(cad software, deliveryaddress), (emailconfirmation, softwareid, softwareprice, operatingsystem)
Sense Discovering
some URIs of keyword "cad software" (3/10)
http://xmlns.com/wordnet/1.6/ (ONT1)
http://www.ksl.standford.edu/projects/DAMl/UNSPSC.daml (ONT2)
http://www.daml.org/2004/05/unspsc/unspsc.owl (ONT3)
some URIs of keyword "deliveryaddress" (3/18)
http://www.daml.ri.cmu.edu/ont/EComererce−Resource.daml (ONT1)
http://daml.umbc.edu/ontologies/webofbelief/adress (ONT2)
http://www.daml.org/services/daml−s/2001/05/Congo−profile1 (ONT3)
Sense Extractor
some senses of "cad software" (3/10)
code
software
s1=<{ONT1#cad software,...},cad software,"software...", 1, 1>
class
cad software
resource
product
s2 = < {ONT2#cad software}, cad software, "...", 1, 1 >
software
drawing and imaging software
s3 = < {ONT3#cad software}, cad software, "", 1, 1 >
some senses of "deliveryaddress" (3/18)
input
range(#literal)
s1 = < {ONT1#deliveryaddress}, deliveryaddress, "", 1, 1 >
prop
deliveryaddress
domain(USPSAdress)
s2 = < {ONT2#deliveryaddress}, deliveryaddress, "", 1, 1 >
deliveryadress
s3 = < {ONT3#deliveryaddress}, deliveryaddress, "", 1, 1 >

Fig. 3. Extraction of senses of "CAD software" and "deliveryaddress".

B. Alignment of Senses.

This process uses an incremental algorithm for the alignment of the different keyword senses in order to remove the possible semantics redundancy among them. Senses are merged when the estimated synonymy probability between them is above a certain threshold. The synonymy measure combines a standard string distance metric with a structural similarity measure that is based on vector space techniques. Thus the result is a set of different possible senses for each user keyword.

C. Disambiguation of Keyword Senses

The previous steps provide a set of possible (non-redundant) senses for each user keyword, but what is the intended meaning of each keyword? we guess that the user intention is implicitly contained in the set of keywords that he or she entered. In [4] we propose a method to disambiguate[2] iteratively each user keyword taking into account the possible senses of the other keywords. Its output is a single ontological sense for each keyword. More details about disambiguation, which is not the goal of this paper, can be found in [4].

In the Figure 4, we show the steps in order to align and disambiguate the candidate senses of keywords. Notice that the final sense alignment process has integrated the senses $s1$ and $s3$ of keyword "cad software" into single sense $s1'$ and $s3'$ respectively. The same happens with the sense $s1$ of keyword "deliveryaddress", the result is the sense $s1'$. In the next step, disambiguation, our system selects the most probable intended meaning of each keyword: for the keyword "cad software" the selected sense $s1'$ is "software used in art and architecture and

[2]Disambiguation is the process of picking up the most suitable sense of a polysemous word according to a given context.

engineering and manufacturing to assist in precision drawing", and the sense $s1'$ ("an input") is proposed for the keyword "deliveryaddress".

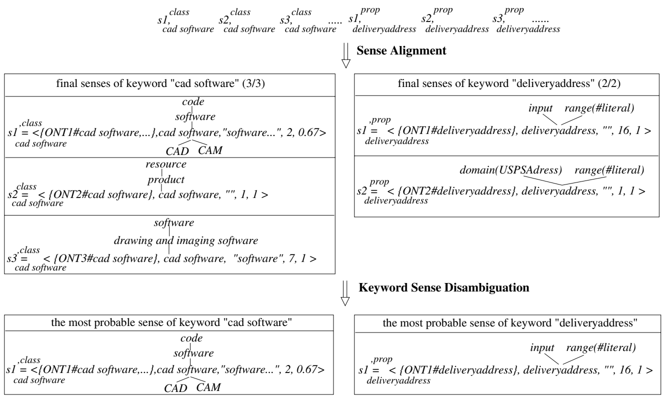

s1, class cad software
s2, class cad software
s3, class cad software
.....
s1, prop deliveryaddress
s2, prop deliveryaddress
s3, prop deliveryaddress
......
Sense Alignment
final senses of keyword "cad software" (3/3)
code
software
s1 class cad software = <{ONT1#cad software,...},cad software,"software...", 2, 0.67>
CAD
CAM
resource
product
s2 class cad software = < {ONT2#cad software}, cad software, "", 1, 1 >
software
drawing and imaging software
s3 class cad software = < {ONT3#cad software}, cad software, "software", 7, 1 >
final senses of keyword "deliveryaddress" (2/2)
input
range(#literal)
s1 prop deliveryaddress = < {ONT1#deliveryaddress}, deliveryaddress, "", 16, 1 >
domain(USPSAdress)
range(#literal)
s2 prop deliveryaddress = < {ONT2#deliveryaddress}, deliveryaddress, "", 1, 1 >
Keyword Sense Disambiguation
the most probable sense of keyword "cad software"
code
software
s1 class cad software = <{ONT1#cad software,...},cad software,"software...", 2, 0.67>
CAD
CAM
the most probable sense of keyword "deliveryaddress"
input
range(#literal)
s1 prop deliveryaddress = < {ONT1#deliveryaddress}, deliveryaddress, "", 16, 1 >

Fig. 4. Sense alignment and disambiguation of candidate senses.

IV. Discovering heterogeneous Web services

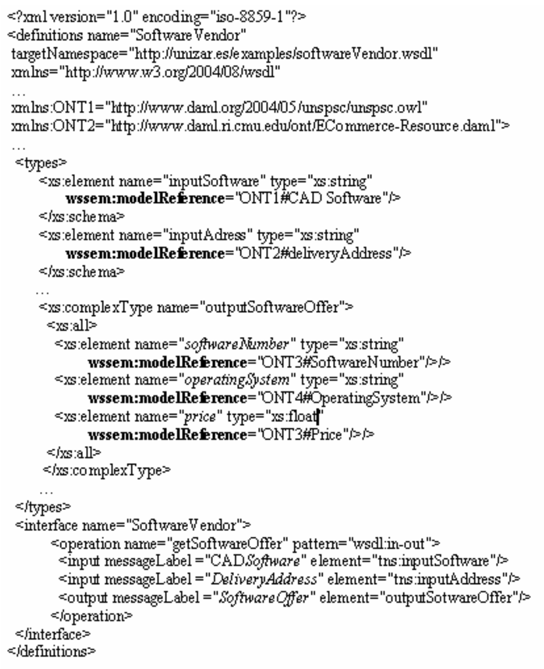
In this section we show the contribution of this paper in the tasks of description and discovery of services using semantic keywords.

A. Semantic Description and Publication of Web Services

The previous step provides a set of semantic keywords for each element that describes the wanted Web service. In order to add the semantics found to the description of the service we relies on the WSDL-S [19]. The WSDL-S specification defines WSDL based elements (using the extensible elements of WSDL) that can be used to semantically annotate a Web service, allowing publishers to unambiguously describe its characteristics. The WSDL-S specification is agnostic to the ontology specification language, unlike OWL-S and WSMO which require the use of specific conceptual models.

Although our system proposes automatically ontological concepts to describe a Web service, we consider that a certain intervention of the user is necessary still. Thus, the service provider can just confirm the result provided by the system (a WSDL semantically annotated) and proceed directly to service publishing. Alternatively he can modify the semantic annotations that he considers convenient, either choosing another sense for a keyword or executing a new search for obtaining other semantic keywords that define the elements of the service.

The sketched WSDL file shown in Figure 5 represents the software vendor service introduced as motivating example. Remember that the service has operations for both offering and buying software products. In the Figure, the WSDL construct inputs *CAD software* and *deliveryadresss* are mapped to ontological concepts discovered automatically by our system. The same process is performed for other elements of the service. Notice also that in this example, the inputs are simple types while the output is a complex type composed of "SoftwareNumber", "OperatingSystem" and "Price". To annotate simple types we use the extensibility element modelReference



```
<?xml version="1.0" encoding="iso-8859-1"?>
<definitions name="SoftwareVendor"
 targetNamespace="http://unizar.es/examples/softwareVendor.wsdl"
 xmlns="http://www.w3.org/2004/08/wsdl"
 ...
 xmlns:ONT1="http://www.daml.org/2004/05/unspsc/unspsc.owl"
 xmlns:ONT2="http://www.daml.ri.cmu.edu/ont/ECommerce-Resource.daml">
 ...
  <types>
      <xs:element name="inputSoftware" type="xs:string"
          wssem:modelReference="ONT1#CAD Software"/>
      </xs:schema>
      <xs:element name="inputAdress" type="xs:string"
          wssem:modelReference="ONT2#deliveryAddress"/>
      </xs:schema>
     ...
      <xs:complexType name="outputSoftwareOffer">
       <xs:all>
        <xs:element name="softwareNumber" type="xs:string"
             wssem:modelReference="ONT3#SoftwareNumber"/>/>
        <xs:element name="operatingSystem" type="xs:string"
             wssem:modelReference="ONT4#OperatingSystem"/>/>
        <xs:element name="price" type="xs:float"
             wssem:modelReference="ONT3#Price"/>/>
       </xs:all>
      </xs:complexType>
      ...
  </types>
  <interface name="SoftwareVendor">
        <operation name="getSoftwareOffer" pattern="wsdl:in-out">
         <input messageLabel ="CADSoftware" element="tns:inputSoftware"/>
         <input messageLabel ="DeliveryAddress" element="tns:inputAddress"/>
         <output messageLabel ="SoftwareOffer" element="outputSotwareOffer"/>
        </operation>
  </interface>
</definitions>
```

Fig. 5. WSDL semantically describe with ontology concepts.

provided by WSDL-S to associate annotations to element. On the other hand, the complex types are annotated using a bottom level annotation: annotating at leaf element level (e.g. "SoftwareNumber" concept). In the following we describe the steps in order to discover the most suitable service(s) that satisfy the request of a service consumer.

B. Semantic Web Service Discovery

In order to obtain more accurate and relevant results, both the description of the Web service and the user request needs to be semantically described through ontology concepts. Coming back to our example, let us suppose that the service consumer provides the following information to obtain offers of software products (see top of the Figure 6). Thus, to achieve to goal this keywords need add semantics a their descriptions. In the Figure 6 we show the three first steps performed by the *semantic keyword discovering* process (described in the Section III) in order to obtain the semantics of the service request input parameter (due to space limitations). Remember that this same process is used to discover the semantics of the elements that describe a Web service. In the next steps of this process, our system selects, as the most probable intended meaning of the keyword "software system", the sense $s1$ ("written programs or procedures or rules and associated documentation pertaining to the operation of a computer system...").

With the requirements of the user captured and semantically described, in a first phase the system matches Web services considering the characteristics that they provide. In the example, the user specifies the keywords "system", "application", and "software program" for describing the service. This information is sent to the *service ontology* which queries the OWLJessKB reasoner in order to match services categories with the user request. A catalog of services that fulfil the

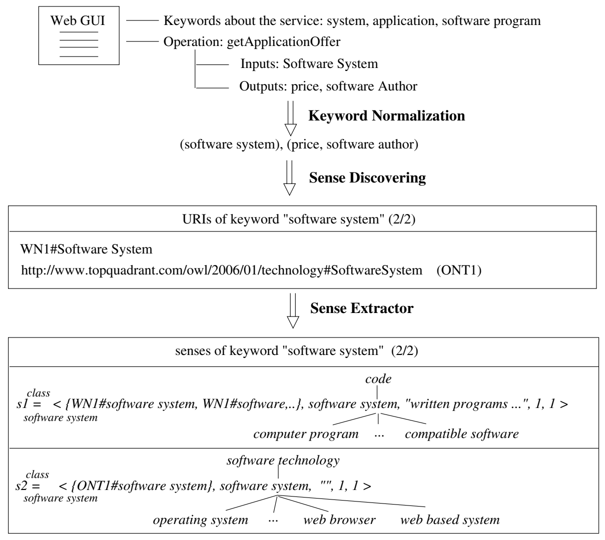

Web GUI
Keywords about the service: system, application, software program
Operation: getApplicationOffer
Inputs: Software System
Outputs: price, software Author
Keyword Normalization
(software system), (price, software author)
Sense Discovering
URIs of keyword "software system" (2/2)
WN1#Software System
http://www.topquadrant.com/owl/2006/01/technology#SoftwareSystem (ONT1)
Sense Extractor
senses of keyword "software system" (2/2)
code
class
s1 = < {WN1#software system, WN1#software,..}, software system, "written programs ...", 1, 1 >
software system
computer program ... compatible software
software technology
class
s2 = < {ONT1#software system}, software system, "", 1, 1 >
software system
operating system ... web browser web based system

Fig. 6. Extraction of senses of the keyword "software system".

general requirements expressed by the user are stored in the *semantic discovering engine* as result. We relies in JESS and OWLJessKB to reason with OWL ontologies, because JESS is a one of the smallest and fastest rule engines available. Furthermore, it is tightly integrated with Java and incorporates the latest advances in rule-based reasoning algorithms. On the other hand, OWLJessKB is an OWL reasoner based on Jena and JESS APIs. It support TBOX reasoning (the intentional knowledge in terms of concepts and properties) also has explicit support for subsumption and classification reasoning.

In order to obtains a list of functional requirements that the user is looking for, the *selection service by functionality* step is in charge of processing the user request. This information is translated by a dedicated parser into a semantic template process (as suggested by [8]) which captures the abstract functionality of the service request. The semantic template is expressed in the same language used to describe a Web service (see Figure 5). All information captured in the template is then passed to the semantic discovering engine, which performs a semantic matching between the user requirements and service capabilities.

The semantic discovering engine works on one service at a time. For each user requirement this process verifies if a service contains one or more compatible capabilities. This process is covered by 1) a ontology matching process and, 2) a subsumption reasoning algorithm. The first process is use to make the concepts of the description and discovering of services interoperable. In a previous work [2] we proposed a synonymy measure based on statistics techniques and ontological similarity used to integrate senses that are similar enough. This measure can be used also to discover the equivalence relationships between the concepts used to describe and discover a service. Our approach to ontology matching relies on both linguistic and structural characteristics of ontologies. In the Figure 7, we show the hierarchical graph of hypernyms and hyponyms of the ontological terms used for describe the input of a Web service. The graph of the left correspond to concept "cad software" used by the service provider, while the graph of the right corresponds to concept "software system" provided by the service consumer. For instance, our measure discovers that the concepts "software" and "software system" are similar enough, since both their names and ontological contexts (hypernyms and hyponyms) share common characteristics. The same happens for the concept "code" of both ontologies. By asserting the "software" class equivalent to "software system" class using owl:equivalentClass, the concepts become equivalent for any reasoner and deductions are made accordingly. These found correspondences expand the knowledge assertions used by the OWLJessKB reasoner, which may increase the matching accuracy.

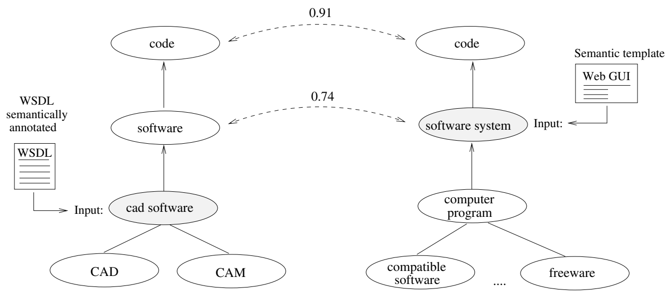


Fig. 7. Matchings between the concepts used as input of a Web service.

In order to discover matchings between the service description(s) and the service request, we are planning to use the algorithm based on [9]. This algorithm focuses primarily on comparing inputs and outputs of a service as semantic concepts represented in OWL. By extracting subsumption relationships between input requirements and outputs they propose a way of ranking semantic matching results. We are working to extend this algorithm to another elements of the Web service as *preconditions* and *effects*. Furthermore to distinguish the different types of match we will adopt a scoring system where exact matches are assigned the highest score. Coming back to our example, consider some facts obtained after performing the ontology matching process described above (see Table I). Now, the reasoner infers that the input "cad software" of the Web service description is defined as a subclass of the input "software system" of the query, so it is considered a match according to subsumption.

TABLE I
N-TRIPLE FACTS CONSIDERED IN THE SEMANTIC MATCHING PROCESS.

subject	*predicate*	*object*
cad software	*subClassOf*	software
software	*subClassOf*	code
computer program	*subClassOf*	software system
software	*equivalentClass*	software system
...	...	...

V. Related Work

The semantics-guided discovery of Web services has gained tremendous importance and new approaches appear frequently. Our work is very close to the work presented in [10], where they propose a flexible discovery of Web services over the

Semantic Web. The framework allow to the users to specify context-specific semantic correspondences between multiple ontologies to solve semantic differences between them. These correspondences are used for selecting services based on the user functional and non-functional requirements, which are then ranked based on a user-specified criteria. However, this work supposes that exist various concepts in different domain ontologies in order to describe a service and the service request. Unlike this work, we propose a way to discover these ontology concepts. In [7] they describe one approach to associate semantic metadata with web services and a registry to enable users to easily find relevant web services and integrate them into web processes. The semantic metadata used are concepts defined in an ontology used for reference, therefore the discovering of web services is limited to the domain of that specific ontology. Our work has inherent advantages as queries third-party ontology pools; we so advocate a shared knowledge approach. In [14] they translate a query into series of queries to Google taking into account the semantics of keywords. In particular the semantic search uses ontologically (WordNet) defined knowledge about keywords and embedded support of advanced Google-search query features in order to construct more efficient queries from formal textual description of searched information. The main drawback of this work is the lack of a highly specialized terminology for narrow domains in WordNet. On the contrary, we advocate using a pool of ontologies instead of just a single one like WordNet, because many technical or subject-specific senses could not be found in just one ontology.

In [11] they describe an automatic extraction method that learns domain ontologies for web service descriptions from textual documentations attached to web services. Unlike this work, we annotate the elements of the WSDL file with the concepts extracted of different ontologies. In [1] they present an approach that combines the strength of the World Wide Web, with the strength of semantic Web services. Their objective is that a human user, e.g., a consultant or an administrator can seamlessly browse the existing WWW and the emerging Web services and that he can easily annotate and invoke Web services on the fly. In this approach the user manually establishes the mappings between concepts, relationships and attributes from the ontology used by the service provider to his ontology. Our prototype uses an ontology matching process in order to discover the matchings between the ontological terms of a service description and a request for service selection.

VI. Conclusions

In this paper we have presented an approach to discover semantic keywords, which can be used to describe and discover Web services respectively. This work enables users to find highly suitable and appropriate partner services in much less time than by manual discovery using syntactic based UDDI searching mechanism. The main features of our proposal are the following:

1) It discovers the possible senses for a set of keywords, by searching and extracting relevant knowledge from different ontology pools; ontology matching and synonymy estimation techniques are used to merge senses considered similar enough.
2) It enables users to specify a service query at a high level of abstraction, and it subsequently reasons about the query in order to verify its compatibility with available services.
3) It uses an ontology matching process in order to translate the concepts in the user request into terms of concepts in domain ontologies (which are used to describe the services).
4) It uses a reasoner to find similarity between service advertisements with the request based on the match between functional elements.

As future work we will extend our approach to deal with invocation and workflow composition. Also, we are planning to incorporate QoS attributes to the process of Web service discovery.

References

[1] S. Agarwal, S. Handschuh, and S. Staab. Annotation, composition and invocation of semantic web services. *Web Semantics*, jul 2004.

[2] M. Espinoza, J. Gracia, R. Trillo, and E. Mena. Discovering the semantics of keywords: An ontology-based approach. In *The 2006 International Conference on Semantic Web and Web Services (SWWS'06), Las Vegas, Nevada (USA)*. CSREA Press, June 2006.

[3] M. Espinoza, R. Trillo, J. Gracia, and E. Mena. Discovering and merging keyword senses using ontology matching. In *1st International Workshop on Ontology Matching (OM-2006) @ 5th International Semantic Web Conference ISWC-2006, Athens, Georgia (USA)*, volume 225, pages 1–5. CEUR-WS, ISSN 1613-0073, Nov. 2006.

[4] J. Gracia, R. Trillo, M. Espinoza, and E. Mena. Querying the web: A multiontology disambiguation method. In *Sixth International Conference on Web Engineering (ICWE'06), Palo Alto (California, USA)*. ACM, July 2006.

[5] HP Labs Semantic Web Programme, Jena : Semantic Web Framework for Java , 2003. http://jena.sourceforge.net/.

[6] Joe Kopena, OWLJessKB: OWL Reasoning for JESS, 2005. http://edge.cs.drexel.edu/assemblies/software/owljesskb/.

[7] K. Li, K. Verma, R. Mulye, R. Rabbani, J. A. Miller, and A. P. Sheth. *Chapter 13. SemBOWSER: Adding Semantics to Biological Web Services Registry*. Kluwer Academic Publishers, May 2006.

[8] K. Li, K. Verma, R. Mulye, R. Rabbani, J. A. Miller, and A. P. Sheth. *Chapter VII. Designing Semantic Web Processes: The WSDL-S Approach*. Springer, August 2006.

[9] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *ISWC 2002, International Semantic Web Conference*, 2002.

[10] J. Pathak, N. Koul, D. Caragea, and V. G. Honavar. A framework for semantic web services discovery. In *WIDM 2005, International Workshop on Web Information and Data Management*, 2005.

[11] M. Sabou, C. Wroe, C. Goble, and H. Stuckenschmidt. Learning domain ontologies for semantic Web service descriptions. *Web Semantics*, sept 2005.

[12] Sandia National Laboratories, JESS - The Rule Engine for Java, 2005. http://herzberg.ca.sandia.gov/jess/.

[13] B. Srivastava and J. Koehler. Web service composition -current solutions and open problems. In *ICAPS 2003 Workshop on Planning for Web Services*, 2003.

[14] V. Terziyan, H. Kaykova, O. Klochko, A. Taranov, O. Khriyenko, O. Kononenko, and A. Zharko. Semantic search facilitator: Concept and current state of development. In *InBCT Tekes Project Report, Chapter 3.1.3: "Industrial Ontologies and Semantic Web"*, 2004.

[15] Universal Description, Discovery, and Integration, UDDI, 2002. http://www.uddi.org.

[16] Web Ontology Language for Web Services, OWL-S, 2003. http://www.daml.org/services/owl-s.

[17] Web Service Description Language, WSDL, 2003. http://www.w3.org/TR/wsdl.

[18] Web Service Modeling Ontology, WSMO, 2005. http://www.wsmo.org.

[19] WSDL-S, 2005. http://www.w3.org/TR/wsdl.