ENHANCED DISCOVERY OF WEB SERVICES
Using Semantic Context Descriptions

Simone A. Ludwig
School of Computer Science, Cardiff University, Cardiff, UK
Simone.Ludwig@cs.cardiff.ac.uk

S.M.S. Reyhani
Department of Information Systems and Computing, Brunel University, Uxbridge, UK
smsreyhani@ieee.org

Keywords: Context information, Semantics, Ontology, Web Service Discovery

Abstract: Automatic discovery of services is a crucial task for the e-Science and e-Business communities. Finding a suitable way to address this issue has become one of the key points to convert the Web into a distributed source of computation, as they enable the location of distributed services to perform a required functionality. To provide such an automatic location, the discovery process should be based on a semantic match between a declarative description of a service being sought and a description being offered. This problem requires not only an algorithm to match these descriptions, but also a language to declaratively express the capabilities of services. This paper presents a context-aware ontology selection framework which allows an increase in precision of the retrieved results by taking contextual information into account.

1 INTRODUCTION

Recently, more and more organisations are implementing IT systems across different departments. The challenge is to find a solution that is extensible, flexible and fits well with existing legacy systems. Replacing legacy systems to cope with the new architecture is not only costly but also introduces a risk to fail. In this context, the traditional software architectures prove ineffective in providing the right level of cost effective and extensible Information systems across the organisation boundaries. Service Oriented Architecture (SOA) (McGovern, 2003) provides a relatively cheap and more cost-effective solution addressing these problems and challenges.

One important factor in defining a new model of Software Architecture is the ever-changing business model. Modern day business constantly needs to adapt to new customer bases. The ability to quickly adapt to the new customer base and new business partners is the key to success. Sharing IT systems with other organisations is a new trend in the business. For example, businesses like online auctions are opening their systems to third party organisation in an effort to better reach their customer base. In this context, SOA offers benefit and cost-effectiveness to the business. The process of adapting to the changing business model is not an easy task. There are many legacy systems, which are difficult to make available to the new business partners. These legacy systems might need to change to support the new business functions and integrate to the newly developed IT systems or integrate to the IT systems of its partners'. The complexity of this on the whole is what makes it a constant challenge to organisations.

Dynamic discovery is an important component of SOA. At a high level, SOA is composed of three core components: service providers, service consumers and the directory service. The directory service is an intermediary between providers and consumers. Providers register with the directory service and consumers query the directory service to find service providers. Most directory services typically organise services based on criteria and categorise them. Consumers can then use the directory services' search capabilities to find providers. Embedding a directory service within SOA accomplishes the following:

- Scalability of services
- Decoupling consumers from providers
- Allowing updates of services
- Providing a look-up service for consumers
- Allowing consumers to choose between providers at runtime rather than hard-coding a single provider.

Although the concepts behind SOA were established long before web services came along, web services play a major role in SOA. This is because web services are built on top of well-known and platform-independent protocols (HTTP (Hypertext Transfer Protocol) (HTTP, 2004), XML (Extensible Markup Language) (XML, 2004), UDDI (Universal Description, Discovery and Integration) (UDDI, 2000), WSDL (Web Service Description Language) (WSDL, 2004) and SOAP (Simple Object Access Protocol) (SOAP, 2004)). It is the combination of these protocols that make web services so attractive. Moreover, it is these protocols that fulfil the key requirements of a SOA. That is, a SOA requires that a service be dynamically discoverable and invokeable. This requirement is fulfilled by UDDI, WSDL and SOAP.

However, SOA in its current form only performs service discovery based on particular keyword queries from the user. This, in majority of the cases leads to low recall and low precision of the retrieved services. The reason might be that the query keywords are semantically similar but syntactically different from the terms in service descriptions. Another reason is that the query keywords might be syntactically equivalent but semantically different from the terms in the service description. Another problem with keyword-based service discovery approaches is that they cannot completely capture the semantics of a user's query because they do not consider the relations between the keywords. One possible solution for this problem is to use ontology-based retrieval.

In this paper, ontologies are used for classification of services based on their properties. This enables retrieval based on service types rather than keywords. This approach uses context information to discover services using context and services descriptions defined in ontologies.

This paper has the following structure: Section 2 gives an account of related research. In section 3 the framework is introduced showing the architecture and the matching algorithm. Section 4 describes the implementation of the prototype outlining the tools used. In section 5, an application example demonstrates the usability of the approach following an evaluation in section 6. Section 7 concludes this paper by summarising the findings.

2 RELATED RESEARCH

The Web Services Description Language (WSDL) is an XML-based language used to describe a Web service. This description allows an application to dynamically determine a Web service's capabilities, which are for example, the operations it provides, their parameters, return values, etc. A UDDI repository is a searchable directory of Web services that Web service requestors can use to search for Web services and obtain their WSDL documents. WSDL documents, however, do not need to be published in a repository for consumers to take advantage of them. They are also obtainable through a Web page or an email message.

The Universal, Description, Discovery and Integration Extension (UDDIe) (Shaikhali, 2003), takes an approach that relies upon a distributed registry of businesses and their service descriptions implemented in a common XML format. UDDIe specifications consist of an XML schema for SOAP messages, and a description of the UDDIe API specification. Together, these form a base information model and interaction framework that provides the ability to publish information about a broad array of Web services. It follows the same specification and standards for the registry data structure and API specification for inquiring and publishing services from the registry. However, there are slight changes and extensions in the data structure and the API to improve and maximise the usage of the registry. UDDIe defines four core types of information that provide the kinds of information that a technical person would need to know in order to use partner's Web services. These are: business information; service information, binding information; and information about specifications for services. This information can be discovered by discovery calls based on the later data types.

The Web Service Modeling Ontology (WSMO) (Keller, 2004) provides the conceptual framework for semantically describing web services and their specific properties. The Web Modeling Language (WSDL) is a formal language for annotating web services with semantic information, which is based on the WSMO conceptual framework. WSMO aims to create an ontology for describing various aspects related to Semantic Web Services, with the defined focus of solving the integration problem. WSMO also takes into account specific application domains (e-Commerce and e-Work) to ensure the applicability of the ontology for these areas.

Mandel and Sheila (Mandel, 2003) automated web service discovery by using a semantic translation within a semantic discovery service. The

approach uses a recursive back-chaining algorithm to determine a sequence of service invocations, or service chain, which takes the input supplied by BPWS4J and produces the output desired by BPWS4J. The translation axioms are encoded into translation programs exposed as web services. The algorithm invokes the DQL (DAML Query Language) (Fikes, 2002) service to discover services that produce the desired outputs. If the semantic discovery service does not have a required input, the algorithm searches for a translator service that outputs the required input and adds it to the service chain. As the process is recursive it terminates when it successfully constructs a service chain, or the profiles in the knowledge base are exhausted.

3 FRAMEWORK

As seen from the existing approaches the need for more expressiveness of service descriptions was stated revealing the limitation of a syntactic approach to service discovery. To follow these movements proposed by the related work towards a semantic based approach for service discovery the context-aware ontology selection framework is proposed. This approach supplements the current approaches by taking context attributes for the service discovery process into account. Additional requirements have driven this framework towards a context-aware ontology selection framework described in the following section.

3.1 Requirements

An advertisement matches a request, when the advertisement describes a service that is sufficiently similar to the service requested (Paolucci, 2002). The problem of this definition is to specify what "sufficiently similar" means. Basically, it means that an advertisement and a request are "sufficiently similar" when they describe exactly the same service. This definition is too restrictive, because providers and requesters have no prior agreement on how a service is represented and additionally, they have very different objectives. A restrictive criterion on matching is therefore bound to fail to recognise similarities between advertisements and requests.

Specific requirements for the context-aware ontology selection framework are as follows:

1. *High Degree of Flexibility and Expressiveness*
   The advertiser must have total freedom to describe their services. Different advertisers want to describe their services with different degrees of complexity and completeness. The description tool or language must be adaptable to these needs. An advertisement may be very descriptive in some points, but leave others less specified. Therefore, the ability to express semi-structured data is required.
2. *Support for Subsumption*
   Matching should not be restricted to simple service name comparison. A type system with subsumption relationships is required, so more complex matches can be provided based on these relationships.
3. *Support for Data Types*
   Attributes such as quantities and dates will be part of the service descriptions. The best way to express and compare this information is by means of data types.
4. *Matching Process should be Efficient*
   The matching process should be efficient which means that it should not burden the requester with excessive delays that would prevent its effectiveness.
5. *Flexible and Modular Structure*
   The framework should be flexible enough to Web applications to describe their context semantics in a modular manner.
6. *Lookup of Matched Services*
   The framework should provide a mechanism to allow the lookup and invocation of matched services.

3.2 Architecture

The architecture shown in Figure 1 comprises of clients, matchmaker, context and service ontologies, registries, and web servers hosting the web services.

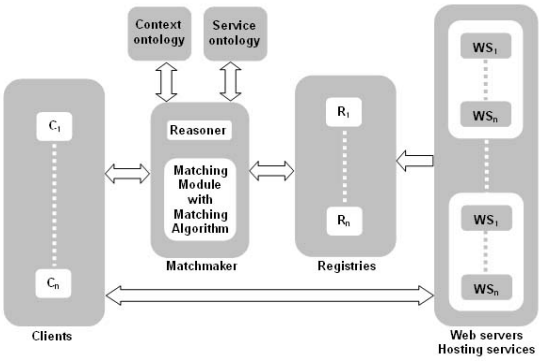

Figure 1. Matching Architecture.

The components are now explained in more detail:

- *Clients* provide an interface for the users to describe their service requests. The client also

lists the matches and provides the facility to call the web services retrieved.

- *Registries* contain the service information. Service descriptions are in the form of service name, service attributes (inputs and outputs) and service description.
- *Web Servers* host the web services.
- *Matchmaker* consists of the matching module including the matching algorithm and a reasoner for the ontology matching process. The matching algorithm is explained in further detail in the following section.
- *Ontologies* (context and services) describe the domain knowledge such as book shop services and provide a shared understanding of the concepts used to describe services. Contextual information is crucial to ensure a high quality service discovery process (Gruber, 1992).

Figure 2 shows the matchmaking steps as processed by the matching algorithm.

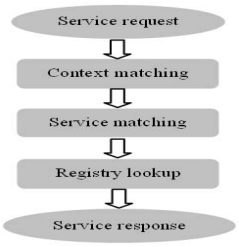

Figure 2. Matchmaking Steps.

The interactions of a service request are the following: The user contacts the matchmaker where the matching algorithm is stored. The matchmaker contacts the context ontology and reasons depending on a set of rules defined. The same is carried out for the services ontology. Having additional match values the registry is then queried to retrieve services descriptions which match the request and returns the service details to the user via the matchmaker. The parameters stored in the registry are service name, service attributes, service description and contact details. Having the URL of the service the user can then call the web service and interact with it.

3.3 Matching Algorithm

The main component of the context-aware ontology selection framework is the matching algorithm. The matching algorithm categorises the matches into different classes. The different matching degrees are as follows. Consider a user request R and a service description S. In order to rank the relevance of the match we classify the matches into the following 5 categories:

- *Exact match* $R = S$: The request matches the service exactly, i.e. all properties are a match.
- *Plug-in match* $R \subset S$: The service allows more than the requester wants.
- *Subsume match* $S \subset R$: A subset of the request is fulfilled.
- *Intersection match* $R \cap S \neq \phi$: The request is partially fulfilled.
- *Disjoint match* $R \cap S = \phi$: The request and the service do not share any properties.

The following three categories can be derived from classifying the types of matches that are useful for the user:

1. *Precise match*: Exact and Plug-in match - The service provides the requested functionality or more.
2. *Partial match*: Subsume and intersection match - The service is capable of providing part of the requested functionality.
3. *Mismatch*: Disjoint match - The service is not capable of providing the requested functionality and therefore will not be returned to the user.

```
CP:    Context parameters
CA:    Context attribute
SP:    Service parameters
RS:    Returned services descriptions
MS:    Matched services

CP, SP ← read_Service_Request()

Context Matching:
load_Context_Ontology()
parse_Context_Ontology_and_Load_Rule_Set()
CA ← query_Ontology_for_Context(CP)

Service Matching:
load_Services_Ontology()
parse_Services_Ontology_and_Load_Rule_Set()
RS ← query_Ontology_for_Context(CA,SP)

Registry Lookup:
MS ← lookup_Registry(RS)

return MS
```

Figure 3. Pseudo Code of Matching Algorithm.

The algorithm shown in Figure 3 reads the service request parameters (context attributes and service attributes) from the client first. Then the context ontology is parsed and rules are applied to match the context keyword by providing the context attributes. Having the context keyword and the service attributes allows to query the services ontology which in turn returns the service matches. This list is then forwarded to the registry module where the lookup is performed retrieving the necessary contact details for each service.

3.4 Requirement Fulfilment

This framework is based on semantic service descriptions and it fulfils the six requirements specified in section 3.1 as follows.

Requirement 1 to 4 is fulfilled by the use of a shared ontology and a reasoning engine to achieve semantic matchmaking. Shared ontologies are needed to ensure that terms have clear and consistent semantics. Otherwise, a match may be found or missed based on an incorrect interpretation of the request. The matchmaking engine should encourage providers and requesters to be precise with their descriptions. To achieve this, the service provider follows an XML-based description, which is the ontology language OWL. To advertise and register its services the service requester generates a description in the specified OWL format. Defining the ontologies precisely allows the matchmaking process to be efficient. The advertisements and requests refer to OWL concepts and the associated semantics. By using OWL, the matchmaking process can perform implications on the subsumption hierarchy leading to the recognition of semantic matches despite their syntactical differences between advertisements and requests. The use of OWL also supports accuracy, which means that no matching is recognised when the relation between the advertisement and the request does not derive from the OWL ontologies. Complex reasoning needs to be restricted in order to allow the matching process to be efficient.

Requirement 5 is fulfilled as the framework supports flexible semantic matchmaking between advertisements and requests based on the ontologies defined. Minimising false positives and false negatives is achieved with the selection process, where the request is matched within the appropriate application context. The design of having context and services ontologies separately allows a modular design as it encapsulates the context knowledge from the services knowledge. This allows other applications to specify their service semantics separate from the context semantics.

Requirement 6 is fulfilled by the usage of a registry service. The registry service allows the lookup of service details providing the user with the service URL.

4 IMPLEMENTATION

The prototype implementation is shown in Figure 4. The implementation is centred around the context and services ontologies that structure knowledge about the domain for the purposes of presentation and searching of services. The matchmaking engine performs the semantic match of the requested service with the provided services. This allows close and flexible matches of the matchmaking process. This prototype is based on Web services technology standards. The user interface is developed with JSPs (Java Server Pages). The communication from the JSPs with the underlying process is done with JavaBeans. The implementation of the Web services was done in Java using WSDL, XML and SOAP. The UDDI registry is used for the final selection stage which is the registry selection. The actual service is matched with the service request depending on the ontologies loaded.

The heart of the portal implementation is the semantic matchmaking. The OWL parser parses the context and services ontologies. With a defined set of rules the inference engine reasons about the ontologies and with the matched results a lookup in the UDDI registry is performed. The services get then displayed in the user portal, where the user can select the appropriate service from the list.

For the context and services ontologies OWL was chosen as it provides a representative notion of semantics for describing services. OWL allows subsumption reasoning on concept taxonomies. Furthermore, OWL permits the definition of relations between concepts. For the inference engine rules were defined using the JESS (Java Expert Systems Shell) language (JESS, 2004). The JESS API (Application Programming Interface) is intended to facilitate interpretation of information of OWL files, and it allows users to query on that information. It leverages the existing RDF API to read in the OWL file as a collection of RDF triples.

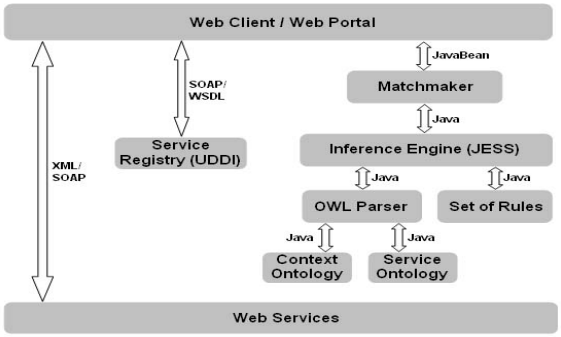

Figure 4. Prototype Implementation.

JESS was chosen as a rule-based language for the prototype as it provides the functionality for defining rules and queries in order to reason about the ontologies specified. It supports the development of rule-based expert systems which can be tightly coupled to code written in the portable Java

language. JESS is a forward chaining production system that uses the Rete algorithm (Forgy, 1982). The Rete algorithm is intended to improve the speed of forward-chained rule systems by limiting the effort required to recompute the conflict set after a rule is fired. Its drawback is that it has high memory space requirements.

In the prototype implementation, queries depending on the specified ontology and service definition structure are specified. These get called whenever a search request is performed by the user. The search request is given by search parameters the user specifies. If datatypes, in JESS syntax `PropertyValue`, of a defined class should be found then the `defquery` in Figure 5 is invoked.

```
(defquery query-for-class-of-a-given-property
"Find the class to a given property."
   (declare (variables ?class))
   (triple
      (predicate "http://www.w3.org/2000/01/rdf-
                  schema#domain")
      (subject ?class)
      (object ?x)
   )
)
```

Figure 5. JESS Query.

With such queries, reasoning about classes of the ontology is achieved by the matching module. The context ontology is parsed by a OWL parser. The attributes and classes of OWL describe the concept of the ontology. The service request is being matched semantically by parsing the context and services ontology and the application of the rules defined. The OWL code facilitates effective parsing of service capabilities through its use of generic RDF(S) symbols compared with OWL specific symbols. With a defined set of rules an inference engine reasons about the value parameters parsed from the ontology. Other queries implemented include sub-classing, datatype, object and functional properties.

5 APPLICATION EXAMPLE

An application scenario was chosen to demonstrate the usability of the approach. It is assumed that many e-shopping web services are available on the Web. These can be any kind of services e.g. Amazon, eBay, etc., wrapped as web services offering different goods to buy such as *Books*, *Bikes* and *CDs*. It is furthermore assumed that in most cases a client searches for a service not knowing the service name. The user only specifies a service request with a few keywords describing the service needs. For this scenario a context ontology was created supplying the categories of services for e-shopping. The context ontology contains categories representing *Food*, *Clothes*, *Bikes*, *Cars*, *Shoes*, *Books* and *CDs*. The underlying classes contain many associative relations to each of the categories. Each of the classes belonging to one of the categories contains attributes describing the class further. E.g. class *Business* (belonging to context *Books*) contains the attributes *computer*, *reading*, etc. For a special application domain two identical attributes in more than one class could be eliminated. However, if context ontologies would be reused from other sources this ambiguity can not be disqualified. The prototype implementation solves this problem by taking the additional context parameters into account to eliminate the "wrong" context. If the user only specifies one context parameter which matches two categories then the prototype returns a mismatch statement.

The context ontology is written in OWL (OWL, 2004) description containing class (`<owl:Class rdf:ID="Services"/>`) and subclass (`<rdfs:subClassOf rdf:resource="#Services" />`) relationships. An OWL ontology is made up of several components, some of which are optional, and some of which may be repeated. OWL constructs are presented in a structured format including RDF triples as shown below.

The structure of the e-shopping services ontology is the following: The first level contains the corresponding categories of the context ontology. The second level represents the actual service implementation with the attributes below. For example, one service specification outlines the *Books* web service. Different service implementations are *BookBuy*, *Bookshop*, *BuyBooks*, *Books* and *BookSale*.

In the services ontology not only class (`<owl:Class rdf:ID="Services"/>`) and subclass (`<rdfs:subClassOf rdf:resource="#Services" />`) relationships are declared but also data type property relationships (`<owl:DatatypeProperty rdf:ID="Price">`) describing the attributes of the service.

In order to demonstrate how the process from service request to service response works is shown next. The user issues a service request consisting of context and service attributes. The context attributes (e.g. *computer* and *reading*) are taken first and the context ontology is queried using these search attributes resulting in the context keyword *Books* which is used for the service search part. The services ontology is then reasoned by using the context keyword and the service attributes specified in the service request query. The retrieved services are *BookBuy*, *Bookshop*, *BuyBooks*, *Books* and *BookSale*. After these services are matched the

service details are retrieved from the registry and returned to the user.

6 EVALUATION

The evaluation is done by calculating precision and recall rates. Precision is the fraction of advertised services which is relevant, i.e. the highest number is returned when only relevant services are retrieved. Recall is the fraction of relevant services which has been retrieved, i.e. the highest number is returned when all relevant services are retrieved.

For the evaluation of precision and recall values a comparison of a keyword-based approach with the prototype approach was conducted. The focus for this evaluation was on book services.

Table 1: Relevant Services.

	service1	service2	service3	service4	service5
context attributes	**computer reading**				
service attributes	**title**	**heading**	**name**	**writing**	**title**
	author	**writer**	**authors**	**maker**	**composer**
	number	**issue**	**no**	**product**	**id**
	category	**class**	**family**	**concept**	**category**
	price	**cost**	**amount**	**worth**	**value**
	publisher	**owner**	**proprietor**	**publisher**	**owner**
	pages	**page number**	**page**	**pages**	**pages**

Table 1 shows the relevant services. All attributes shown in the table are the service attribute parameters used for this evaluation. Matches are indicated in bold.

Table 2: Irrelevant Services.

	service6	service7	service8	service9	service10
context attributes	graph picture				
service attributes	**title**	**issue**	**name**	**product**	**composer**
	number	**owner**	**proprietor**	**pages**	**id**
	price	isbn	issn	book	**value**
	pages	drink	shop	meal	**pages**
	book	pixel	colour	point	book
	shop	font	paragraph	space	food
	colour	space	bold	font	colour

Table 2 shows the irrelevant services. The attributes indicated in bold match with the extended context ontology taken for this experiment, however the context parameters do not match the *Book* category. The number of service attributes is the same for relevant and irrelevant services.

The context parameters define the category of the service which results in the two tables (Table 1 and 2) being relevant services and irrelevant services. The user wants to find *Book* shop services and specifies a service request 1 (context parameters: *computer*, *reading*; service parameters: *title*, *author*, *number*, *category*, *price*, *publisher*, *pages*) with the parameters specified for service 1 in Table 1. Service request 2 is specified with the parameters of service 2 (Table 1) and so on. The context parameters of the service request are always *computer* and *reading*.

Table 3: Matches of Service Requests.

	Number of relevant services	Keyword-based approach	Prototype implementation
Request 1	5	3 relevant 3 irrelevant	5 relevant
Request 2	5	2 relevant 1 irrelevant	5 relevant
Request 3	5	1 relevant 1 irrelevant	5 relevant
Request 4	5	3 relevant 3 irrelevant	5 relevant
Request 5	5	3 relevant 4 irrelevant	5 relevant

Table 3 shows the request and the matches comparing the keyword-based approach with the prototype approach. It shows that only the keyword-based approach returns irrelevant matches as the prototype was customised.

Figure 6 shows the results of the precision and recall values. The precision and recall results of the keyword-based approach range between 20% and 70%, whereby the prototype approach achieved a precision and retrieval rate of 100% in this experimental setup. As the recall and precision rates from the prototype show higher values than the rates from the keyword-based approach, it shows that the user receives a better subset of services that are relevant and in addition, the user receives no services that are irrelevant.

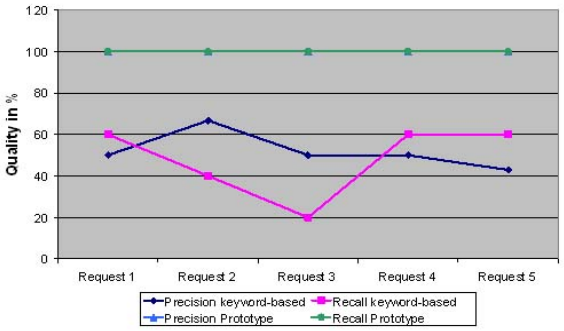

Figure 6: Evaluation of Precision and Recall Values.

Due to the fact that this research is conducted in a limited application domain, the set of advertised services, query and ontology are highly adapted and therefore a result of 100% is retrieved. In a real-world application scenario this correlation might not always be that high, especially if a context ontology from third-parties is used.

The accomplished result of service matches does not state that in every application scenario always values of 100% are achieved but it indicates the improvement in quality of service discovery results by using this semantic approach. Precision and recall measures showed the increase of quality of service matches, which was achieved by the customisation of the context and services ontologies.

7 CONCLUSION

The use of contextual information results in a better service discovery process due to an increased precision of the matched services. The contextual information enhances the expressiveness of the matching process, i.e. by adding semantic information to services, and also serves as an implicit input to a service that is not explicitly provided by the user. The prototype approach facilitates interoperability as the context and service properties are defined and specified in associated ontologies. Re-writing of code or interface wrapping does not need to be done in order to make systems interoperable. The development and maintenance is much easier due to the modular structure and encapsulation of context matching, service matching and registry selection. Whenever a service is added only an entry in the services ontology needs to be included and the service details need to be registered in the registry. The rules defined in the reasoning engine do not need to be modified and the service discovery process is not affected at all when adding services. This is a very important feature for modern information systems, and especially in the area of Web services, where interoperability is a major issue.

A drawback of this approach is that users registering services need to know the category their services belong to. Cases where a service falls into more than one category need to be registricted in order to allow an automatic and precise discovery and selection of service matches.

REFERENCES

McGovern, J., Tyagi, S., Stevens, M., Mathew, S., 2003. *The Java Series Books - Java Web Services Architecture*. Chapter 2, Service Oriented Architecture.

HTTP - Hypertext Transfer Protocol, 2004. W3C, http://www.w3.org/Protocols/.

Extensible Markup Language (XML), 2004. W3C. http://www.w3.org/XML/.

UDDI Technical White Paper, 2000. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf.

Web Services Description Language (WSDL), 2004. Version 1.1, W3C. http://www.w3.org/TR/wsdl.

SOAP Version 1.2, 2004. W3C. http://www.w3.org/TR/soap/.

ShaikhAli, A., Rana, O., Al-Ali, R., Walker, DW., 2003. UDDIe: An Extended Registry for Web Services. *Proceedings of the Service Oriented Computing: Models, Architectures and Applications*, SAINT-2003, Orlando, USA.

Keller, U., Lara, R., Polleres, A., Toma, I., Kifer, M., Fensel, D., 2004. WSML Deliverable – WSMO Web Service Discovery, WSML Working Draft.

Mandell, D.J., McIlraith, S.A., 2003. A Bottom-Up Approach to Automating Web Service Discovery, Customization, and Semantic Translation. *Proceedings of the 12th International World Wide Web Conference*, Workshop on E-Services and the Semantic Web(ESSW'03), Budapest.

Fikes, R., Hayes, P., Horrocks, I., 2002. DAML Query Language, Abstract Specification.

Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K., 2002. Semantic Matching of Web Services Capabilities. *Proceedings International Semantic Web Conference (ISWC 02)*.

Gruber, T.R., 1992. ONTOLINGUA: A Mechanism to Support Portable Ontologies, Version 3.0, Technical Report KSL 91-66, Knowledge Systems Laboratory, Department of Computer Science, Stanford University.

JESS, Java Expert Systems Shell, 2004. http://herzberg.ca.sandia.gov/jess/.

Forgy, C.L., 1982. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Journal of Artificial Intelligence*; 19-17-37.

W3C Working Draft, 2004. "Requirements for a Web Ontology Language". http://www.w3.org/TR/webont-req/.