Chapter #

CONTEXT-AWARE ONTOLOGY SELECTION FRAMEWORK

Simone A. Ludwig[1] and S.M.S. Reyhani[2]
[1]School of Computer Science, Cardiff University, Cardiff CF24 3AA, UK; [2]Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex UB8 3PH, UK

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 in a distributed source of computation, as it enables the location of distributed services to perform a required functionality. To provide such an automatic location, the discovery process should be based on the semantic match between a declarative description of the 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 section presents a context-aware ontology selection framework, which allows an increase in precision of the retrieved results by taking the contextual information into account.

Key words: Context information; semantic service discovery; web services.

1. INTRODUCTION

Recently, more and more organizations are implementing IT systems across different departments. The challenge is to find a solution that is extensible, flexible and fits well with the 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 organization boundaries. Service Oriented

Architecture (SOA) [1] 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 organizations is a new trend in the business. For example, businesses like online auctions are opening their systems to third party organization 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 organizations.

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 organize services based on criteria and categorize 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) [2], XML (Extensible Markup Language) [3], UDDI (Universal Description, Discovery and Integration) [4], WSDL (Web Service Description Language) [5] and SOAP (Simple Object Access Protocol) [6]). 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 approach, ontologies are used for classification of the 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 service descriptions defined in ontologies.

2. BACKGROUND TO ONTOLOGIES

When two or more parties seek a common understanding of something, they must work together to ensure that there is a high degree of correlation and similarity between the details of their respective descriptions and definitions of what they are trying to agree on [7]. This implies that shared understanding requires shared definitions. For example, day-to-day human interactions are made possible by the fact that our society's members share common knowledge and common values. This sharing of common understanding is categorized as the science of ontology, which involves the study of the general concepts and abstractions that make up the fundamental aspects of our world.

Until the 20th century, ontology was considered a sub-field of philosophy. Since the early 1990s, an ontology is also a way to model things in computer science and artificial intelligence. The meaning of the term *ontology* has evolved over the years, and its definition has been slightly blurred when applied to different areas of computing and cybernetics.

Despite certain claims, the term *ontology* is used in a radically different sense in artificial intelligence. This term first appeared in the artificial intelligence literature in 1992 in a paper by Gruber, who stated that "an ontology is a set of definitions of content-specific knowledge representation primitives: classes, relations, functions, and object constants" [8]. With this definition, an ontology is both human and machine readable. An ontology, together with a syntax and semantics, provides the language by which knowledge-based systems can interoperate at the knowledge-level by exchanging assertions, queries and answers.

For Gruber, ontology is the term used to the shared understanding of some domain of interest. It necessarily entails or embodies some sort of world view with respect to a given domain. The world view is often conceived as a set of concepts (e.g., entities, attributes, processes), their definitions and their inter-relationships; this is referred to as a *conceptualization*. Such a conceptualization may be implicit, e.g. existing only in someone's head, or embodied in a piece of software. For example, an accounting package presumes some world view encompassing such concepts as invoice, and a department in an organization. The word ontology is sometimes used to refer to this implicit conceptualization. However, the more standard use is that the ontology is an explicit account or representation of a conceptualization [9].

Depending on who you talk to, the purpose of an ontology can range from a mere vocabulary of terms to a strict formal logic. To understand the terminology used, let us consider the example of an auction. An auction ontology would have to define sellers, buyers, bids, etc. In particular, the following aspects could be found [10]:

- *A taxonomy of concepts:*
  Both buyers and sellers could be considered agents; as a result, agent is the super-concept of the concepts buyer and seller.
- *Relationships between the concepts:*
  Sellers offer goods or buyers make bids.
- *Facts:*
  A fact could be that eBay is a marketplace.
- *Rules:*
  If a buyer makes a bid, then include him in the marketing category "parent".
- *Constraints:*
  A later bid for the same offer must be higher.

In this example, the ontology would contain the taxonomy of the concepts in a domain and would define the relationships between these concepts. The facts, rules and constraints defined could then be applied to the ontology in order to reason about the knowledge.

3. 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) [11], 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 service from the registry. However, there are slight changes and extensions in the data structure and the API to improve and maximize the use 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 a partner’s Web services. These are: business information; service information; binding information; and information about specifications for services. Further, this information can be discovered by discovery calls based on the later data types.

The Web Service Modeling Ontology (WSMO) [12] 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 [13] 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 axiom are encoded into translation programs exposed as web services. The algorithm invokes the DQL (DAML Query Language) [14] 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.

Semantically enhanced service discovery has also been introduced in the area of Mobile Computing. DReggie [15] is a dynamic service discovery infrastructure targeted at mobile commerce applications that besides performing syntactical matching exploits semantic matching using DAML (DARPA Agent Markup Language) to describe services and uses a Prolog reasoning engine for inference. A DReggie Lookup Server to which DReggie Clients submit their services performs the matching process and returns information about matches back to the clients.

The UUID-based description and matching of services mechanism of Bluetooth was enhanced using semantic information associated with services rather than simple UUIDs in hotspot environments [16]. This includes priorities, expected values or service attributes and some index of a match's closeness. To support this matching mechanism and allow more efficient service discovery, a service ontology described in a semantic language and a Prolog-based reasoning engine that uses the ontology was introduced.

Similar research in the Grid area was addressed by Deelman et al. [17] with their workflow generator and Tangmurarunkit et al. [18] with their resource selector. The workflow generator addresses the problem of automatically generating job workflows for the Grid. They have developed two workflow generators. The first one maps an abstract workflow defined in terms of application-level components to the set of available Grid resources. The second generator takes a wider perspective and not only performs the abstract to concrete mapping but also enables the construction of the abstract workflow based on the available components. The system operates in the application domain and chooses application components based on the application metadata attributes.

The ontology-based resource selector exploits ontologies, background knowledge, and rules for solving resource matching in the Grid to overcome the restrictions and constraints of resource descriptions in the Grid. In order to make the matchmaking more flexible and also to consider the structure of VOs the framework consists of ontology-based matchmakers, resource providers and resource consumers or requesters. Resource providers periodically advertise their resources and capabilities to one or more matchmakers using advertisement messages. The user can then activate the matchmaker by submitting a query asking for resources that satisfy the request specification. The query is then processed by the TRIPLE/XSB deductive database system [19] using matchmaking rules, in combination with background knowledge and ontologies to find the best match for the request.

4. CONTEXT-AWARE ONTOLOGY SELECTION 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.

4.1 Need for “Context-Awareness”

Definition of context and “context-awareness” is challenging and is done in many research areas such as artificial intelligence, human-computer interaction, ubiquitous computing etc. In the past, context has been left out in computer science [20]. The computer science field has strived for context-independence for simplicity reasons. By improving the computer’s access to context, thereby introducing context-independence, the richness of communications in human-computer interactions can be improved and more useful computational services can be created [21]. For instance, consider the following example of an e-shopping service, where the e-services can be selected depending on the shopping context. This incorporation of contextual information for the matchmaking process should provide a higher precision and recall of service matches.

4.2 Framework Requirements

An advertisement matches a request, when the advertisement describes a service that is sufficiently similar to the service requested [22]. 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 recognize similarities between advertisements and requests.

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

1. Specification of context descriptions
   Enabling the selection of services via the context descriptions.

2. 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.
3. 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.
4. 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.
5. 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.
6. Flexible and modular structure
   The framework should be flexible enough to Web applications to describe their context semantics in a modular manner.
7. Lookup of matched services
   The framework should provide a mechanism to allow the lookup and invocation of matched services.

4.3 Architecture

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

The components are now explained in more detail:

1. Clients provide an interface for the users to describe their service requests. The interface also lists the matches and provides a facility to call the web services retrieved.
2. Registries contain the service information storing all service data. Service descriptions are in the form of service name, service attributes (inputs and outputs), service description and metadata information.
3. Web Servers host the web services.
4. Matchmaker consists of the matching module including the matching algorithm and a reasoner for the ontology matching part. The matching algorithm is explained in further detail in the following section.

5. 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 [8].

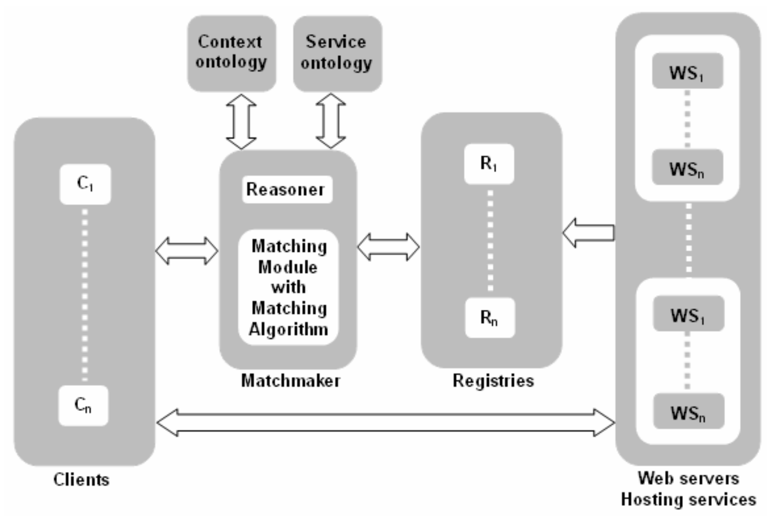


Figure 1. Matching Architecture.

The sequence diagram in Figure 2 shows the interactions of a service request. The user contacts the matchmaker where the matching algorithm is stored (1). The matchmaker contacts the context ontology (2 and 3) and reasons depending on a set of rules defined. The same is done for the services ontology (4 and 5). Having additional match values the registry is then queried (6) to retrieve service descriptions which match the request and returns the service details to the user via the matchmaker (7). The parameters stored in the registry are service name, service attributes, service description, metadata information and contact details. Having the URL of the service the user can then call the web service (8) and interact (9) with it.

Three steps are necessary to perform the request. First the service request is matched semantically within the context specified which provides further attributes for the service matching where services are matched semantically within their service domain and finally a lookup with the registry is done to return the matched service details.

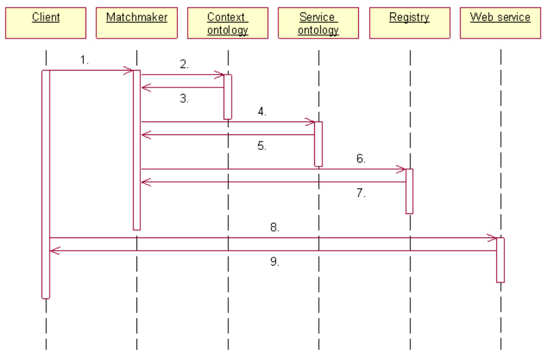


Figure 2. Interaction Diagram.

4.4 Matching Algorithm

The main component of the context-aware ontology selection framework is the matching algorithm. The matching algorithm categorizes 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 5 categories (Figure 3).

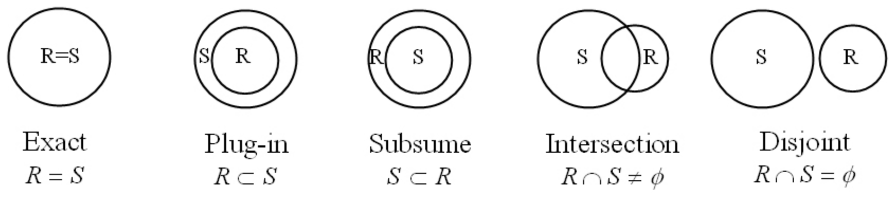


Figure 3. Matching Categories.

These are:

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

Three categories can be derived from classifying the types of matches that are useful for the user. These are:

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.

Furthermore, to break down the matching categories, the matching algorithm implemented for the prototype calculates match scores taking into consideration the number of parameters for each category type (service attributes, service description and metadata information). To relate the match scores with the matching categories the classification is as follows. If the match score is equal to 1 then the match was a precise match which means that all service parameters matched. If the match score is smaller than 1 then the match was a partial match and if the match score returns 0 then it was a mismatch.

```
SNR: Service name from request
SAR: Service attributes from request
SDR: Service description from request
SMR: Service metadata information from request
SN:  Service name from registry
SA:  Service attributes from registry
SD:  Service descriptions from registry
SM:  Service metadata information from registry
MV:  Match value
MS:  Match score
MD:  Match details of service

SNR, SAR, SDR, SMR ← read_service_request_from_GUI()
SN, SA, SD, SM ← load_service_descriptions()
for all service_descriptions_in_registry do
     if SNR equals SN
          MS = 1
     else
          ontology_search_of_context_and_services()
          check_SAR_with_SA()
          MV ← calculate_match_value()
          check_SDR_with_SD()
          MV ← calculate_match_value()
          check_SMR_with_SM()
          MV ← calculate_match_value()
          MS ← calculate_match_score()
     end if
     MD ← store_service_match_details()
end for
```

Figure 4. Matching Algorithm.

The matching algorithm (Figure 4) is defined taking the classification and match categories into consideration. The algorithm reads the service request parameters from the GUI first. Then a connection to the registry is made in order to search and read the service parameters. In a “for loop” considering all services stored in the registry, first the service name of the service is compared with the service name of the request. If they are “equal” (assuming that the user knows the name of the service) the match score is set to 1 and no further steps are necessary. If “not” then the following steps need to be performed. The context and service ontology parameters are read, then the registry is queried using the service request and ontology parameters. If matches are found, then the match values are calculated for all three categories (service attributes, service description and service metadata). Afterwards the overall match score for a particular service is calculated and the service details are retrieved which are then stored and returned.

The overall consideration within the matchmaking approach for the calculation of the match score is to get a match score returned which should be between 0 and 1, where 0 represents a “mismatch”, 1 represents a “precise match” and a value in-between represents a “partial match”. The overall match score consists of the match score for service attributes, service description and service metadata respectively:

$$M_O = \frac{M_A + M_D + M_M}{3} \qquad (1)$$

whereby M_O, M_A, M_D, M_M are the overall, attribute, description and metadata match scores respectively.

Looking at the service attributes first, it is necessary to determine the ratio of the number of service attributes given in the query in relation to the number given by the actual service. To make sure that this ratio does not exceed 1, a normalisation is performed with the inverse of the sum of both values. This is multiplied by the sum of the number of service attributes matches divided by the number of actual service attributes (2a). Similar equations (2b) and (2c) were derived for service descriptions and service metadata respectively. The importance of service attributes, description and metadata in relation to each other is reflected in the weight values.

$$M_A = \frac{w_A}{(n_{AQ} + n_{AS})} \cdot \frac{n_{AQ}}{n_{AS}} \cdot \frac{n_{MA}}{n_{AS}} \qquad (2a)$$

$$M_D = \frac{w_D}{(n_{DQ} + n_{DS})} \cdot \frac{n_{DQ}}{n_{DS}} \cdot \frac{n_{MD}}{n_{DS}} \qquad (2b)$$

$$M_M = \frac{w_M}{(n_{MQ} + n_{MS})} \cdot \frac{n_{MQ}}{n_{MS}} \cdot \frac{n_{MM}}{n_{MS}} \tag{2c}$$

whereby w_A, w_D and w_M are the weights for attributes, description and metadata respectively; n_{AQ}, n_{AS} and n_{MA} are the number of query attributes, service attributes and service attribute matches respectively; n_{DQ}, n_{DS} and n_{MD} are the number of query descriptions, service descriptions and service description matches respectively; n_{MQ}, n_{MS} and n_{MM} are the number of query metadata, service metadata and service metadata matches respectively.

4.5 How does the Architecture fulfill the Requirements?

This framework is based on semantic service descriptions and it fulfils the seven requirements specified in section 4.2 as follows. Requirement 1 is satisfied with the context selection stage. Requirement 2 to 5 are 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 6 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 context and semantic selection stages could have been integrated into one, however 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 and furthermore allows the context selection e.g. been inplemented and searched for via database queries. Requirement 7 is fulfilled by the use of a registry service. The registry service allows the lookup of service details to provide the user with the service URL.

5. IMPLEMENTATION OF PROTOTYPE

The prototype implementation is shown in Figure 5. 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 Java Server Pages (JSPs). 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.

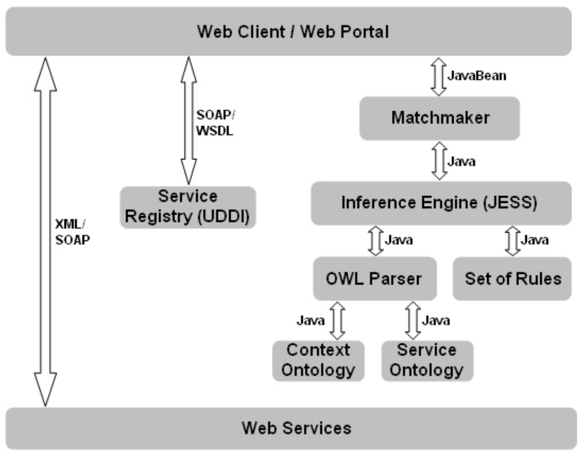


Figure 5. Prototype Implementation.

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 the context and 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 [23]. 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.

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. JESS is an expert system shell and scripting language written entirely in the Java language. JESS 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 [24]. 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 6 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 6. JESS Rule.

With such queries, reasoning about classes of the ontology is achieved with the matching modules and works as follows. 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 rules implemented include sub-classing, datatype, object and functional properties.

6. 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.

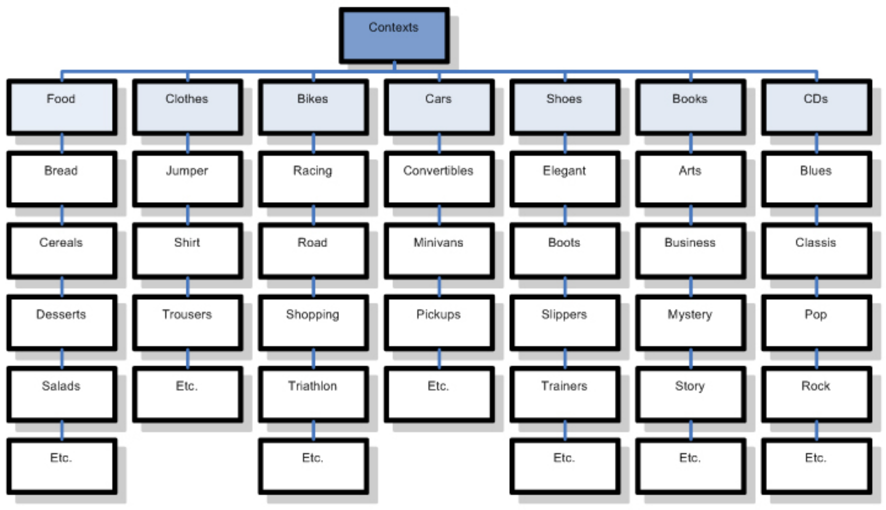


Figure 7. Context Hierarchy.

This ontology, shown in Figure 7, lists the contexts chosen which represent *Food*, *Clothes*, *Bikes*, *Cars*, *Shoes*, *Books* and *CDs*. The underlying classes show many associative links to the different categories. These are normally linked directly with the category class, however for the ease of the reader this single-structured hierarchy was used. In addition, it only shows the classes of the context hierarchy but not the attributes.

Each of the classes belonging to one of the categories contains attributes describing the class further. E.g. class *Business* contains the attributes *computing*, *reading*, etc. For a special application domain the two identical attributes in more than one class could be eliminated. However, if context ontologies would be reused from other sources this can not be disqualified. The prototype implementation solves the 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.

Figure 8 shows the fragment of the OWL description of the context ontology showing class (`<owl:Class rdf:ID="Services"/>`) and subclass (`<owl:Class rdf:ID="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.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.cs.cardiff.ac.uk/ontologies/context.owl#"
  xml:base="http://www.cs.cardiff.ac.uk/ontologies/context.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Context"/>
  <owl:Class rdf:ID="Road"><rdfs:subClassOf><owl:Class rdf:ID="Bikes"/>
     </rdfs:subClassOf></owl:Class>
  <owl:Class rdf:ID="Books"><rdfs:subClassOf
     rdf:resource="#Context"/></owl:Class>
  <owl:Class rdf:ID="Triathlon"><rdfs:subClassOf><owl:Class
     rdf:about="#Bikes"/></rdfs:subClassOf></owl:Class>
  <owl:Class rdf:about="#Bikes"><rdfs:subClassOf
     rdf:resource="#Context"/></owl:Class>
  <owl:Class rdf:ID="Racing"><rdfs:subClassOf
     rdf:resource="#Bikes"/></owl:Class>
  <owl:Class rdf:ID="Food"><rdfs:subClassOf
     rdf:resource="#Context"/></owl:Class>
  <owl:Class rdf:ID="Shopping"><rdfs:subClassOf
     rdf:resource="#Bikes"/></owl:Class>
...
</rdf:RDF>
```

Figure 8. Fragment of OWL Context Ontology.

In Figure 9, the structure of the e-shopping services ontology is shown. The first level contains the corresponding categories of the context ontology. The second level represent the actual service implementation with the attributes below. Given in this hierarchy is only one service specification outlining the *Books* web service. Different service implementations are *BookBuy*, *Bookshop*, *BuyBooks*, *Books* and *BookSale*.

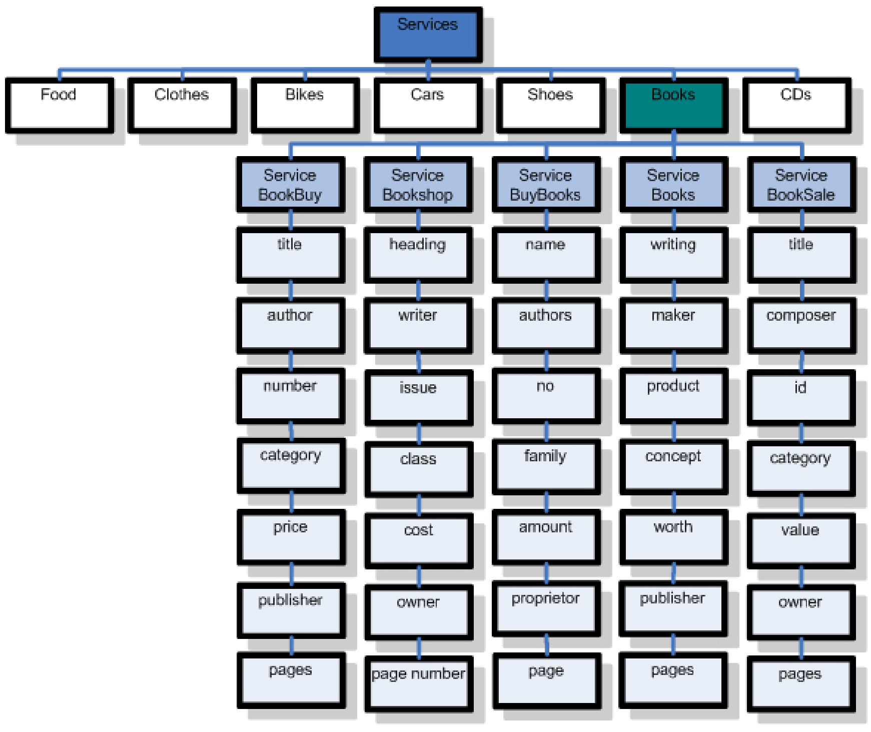


Figure 9. Services Hierarchy.

Figure 10 shows part of the OWL file of the services ontology. In this service ontology not only class (`<owl:Class rdf:ID="Services"/>`) and subclass (`<owl:Class rdf:ID="Services"/>`) relationships are declared but also data type property relationships (`<owl:DatatypeProperty rdf:ID="Price">`) describing the attributes of the service.

How the process from service request to service response works is shown next. The user issues the following service request shown in Figure 11 consisting of context attributes and service attributes.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.cs.cardiff.ac.uk/ontologies/services.owl#"
  xml:base="http://www.cs.cardiff.ac.uk/ontologies/services.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Services"/>
  <owl:Class rdf:ID="Cars"><rdfs:subClassOf
     rdf:resource="#Services"/></owl:Class>
  <owl:Class rdf:ID="ServiceRacer"><rdfs:subClassOf><owl:Class
     rdf:ID="Bikes"/></rdfs:subClassOf></owl:Class>
  <owl:Class rdf:ID="Books"><rdfs:subClassOf
     rdf:resource="#Services"/></owl:Class>
  <owl:Class rdf:about="#Bikes<owl:Class rdf:ID="Services"/></owl:Class>
  <owl:Class rdf:ID="Food"><rdfs:subClassOf rdf:resource="#Services"/>
  </owl:Class>
  <owl:Class rdf:ID="CDs"><rdfs:subClassOf rdf:resource="#Services"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:ID="Price">
    <rdfs:domain rdf:resource="#ServiceRacer"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="CatalogueNumber">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#ServiceRacer"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="NumberOfGears">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#ServiceRacer"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="WheelSize">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#ServiceRacer"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="FrameSize">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#ServiceRacer"/>
  </owl:DatatypeProperty>
...
</rdf:RDF>
```

Figure 10. Fragment of OWL Services Ontology.

The user has the choice of either specifying the name of the service or the rest of the service request file (attributes, description, metadata and weights). In most cases the user will not know the name of the service, therefore the name tag will remain unspecified. The user will specify service request parameters for either only the attribute section or all of the categories attributes, description and metadata. The weight values can be defined according to the user’s preference. The context attributes are first taken 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. All the retrieved services (*BookBuy*, *Bookshop*, *BuyBooks*, *Books* and *BookSale*) are then calculated using the match score metric, in order to identify the ranking of the service result set. After these services are matched the service details are retrieved from the registry and returned to the user.

```
<ServiceRequest>

   <Name>
      <parameter name=”” value=””>                                  EITHER
   </Name>

   <Attributes>                                                         OR
      <ContextAttributes>
       <parameter name=”computer” value=””>
       <parameter name=”reading” value=””>
      </ContextAttributes>
      <ServiceAttributes>
       <parameter name=”title” value=””>
       <parameter name=”author” value=””>
       <parameter name=”isbn” value=””>
       <parameter name=”category” value=””>
       <parameter name=”price” value=””>
       <parameter name=”pages” value=””>
      </ServiceAttributes>
   </Attributes>

   <Description>
      <parameter name=”description” value=”This guide discusses Information
      Retrieval data structures and algorithms”>
   </Description>

   <Metadata>
      <parameter name=”attribute1”
       value=”http://www.amazon.com/exec/obidos/tg/detail/-
       /0134638379/102-2173778-1724124?v=glance”>
      <parameter name=”attribute2”
       value=”http://www.amazon.com/gp/reader/0134638379/
                  ref=sib_dp_pt/102-2173778-1724124#reader-link”>
   </Metadata>

   <Weights>
       <parameter name=”attributes” value=”0.5”>
       <parameter name=”description” value=”0.3”>
       <parameter name=”metadata” value=”0.2”>
   </Weights>

</ServiceRequest>
```

Figure 11. Example of Service Request.

7. EVALUATION OF PROTOTYPE

The purpose of this section is to show that the prototype implementation satisfies the performance requirements as applied in real-world applications and most importantly to show the quality improvement of the matches. Three different set of evaluation measures were carried out. These are performance measurements, precision and recall measurements and match score measurements.

7.1 Performance Measurements

Measurements were carried out to investigate the performance of the prototype. Ten measurements were taken to quantify the time needed to fulfill a service request. Only the service discovery process was measured (without an actual service call) as the primary focus was the matchmaking.

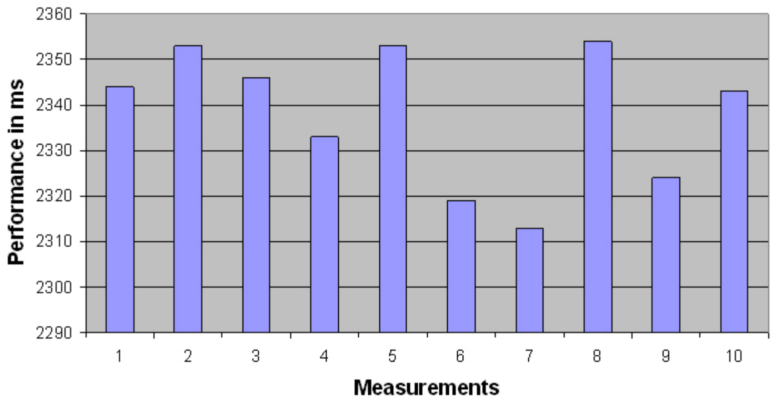


Figure 12. Performance Measurements.

Figure 12 shows that the average time of a search request to be matched is 2338ms. The distribution shows a variation of ± 21ms of the average result. This is a quite acceptable performance outcome in comparison to real-world applications. However, the ontology size might be much bigger for real-world applications which needs to be investigated next.

Another set of performance measurements of the prototype were conducted in order to see how the behaviour of the performance over the complexity of an ontology varies. Only the services ontology was enabled having the context ontology part disabled providing the search request with the context attribute. The measurement setting had the following conditions. All nine ontologies of different complexity levels were placed on the Internet and retrievable by a URL, so that real world measurements could be conducted. The complexity of 1 of the ontology is defined as having 112 elements, thereof 47 classes and 65 data type properties. Complexity 2 is the double amount of elements. Having complexity 16 results in 1792 elements, where 752 are classes and 1040 are data type properties.

Figure 13 shows the performance measurements of the prototype with respect to ontology complexity. The graph shows a linear distribution. The regression line shows an average increase of about 700ms per increase of complexity of the ontology. There is an offset of about 2000ms which is due to the instantiation and resetting of the reasoning engine and the rules and queries applied. As expected, the flexible and powerful matchmaker has a disadvantage which is a reduced performance, in particular for large ontologies. The results highlight the linear degradation in performance exhibited by the search. If only a keyword based approach was desired the performance would be better. However, for small and medium size ontologies (up to 1000 elements) the evaluation shows an acceptable performance.

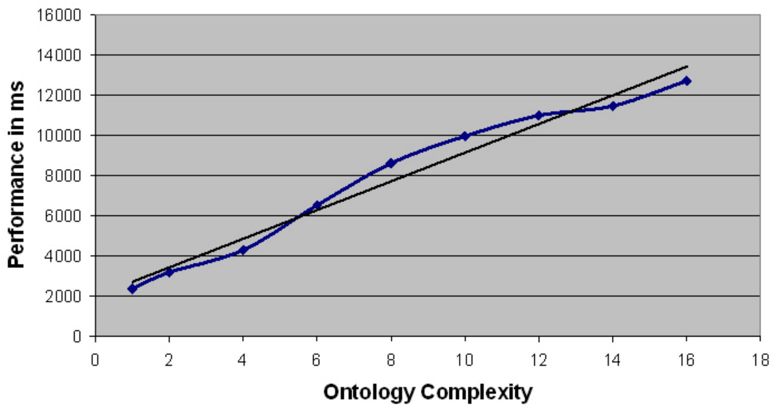


Figure 13. Performance versus Ontology Complexity.

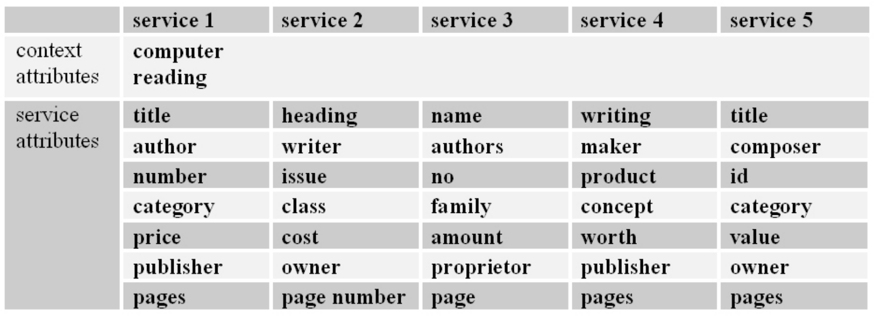
As expected, the additional functionality results in a lower performance shown in the linear performance decrease by an increase in complexity of the ontology. However, the prototype approach achieves an increase in the quality of service matches. Therefore, precision and recall measurements were taken to show the quality improvement for this semantic approach.

7.2 Precision and Recall Measurements

The evaluation is done by calculating precision and recall rates. Consider a set of relevant services (R) within a set of advertised services (A). Ra is the number of services in the intersection of the sets R and A.
Precision is the fraction of advertised services which are relevant i.e., $PRECISION = \frac{|Ra|}{|A|}$. The highest number is returned when only relevant services are retrieved.

Recall is the fraction of relevant services which have been retrieved i.e., $RECALL = \frac{|Ra|}{|R|}$. The highest number is returned when all relevant services are retrieved.

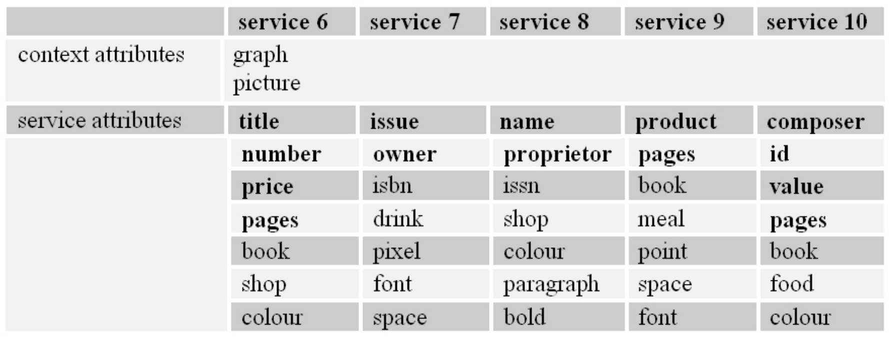
For the evaluation of precision and recall values a comparison of a keyword-based approach with the prototype approach was conducted. Even though the matching algorithm considers all service categories (service name, service attributes, service descriptions, metadata) for this evaluation only the service attributes were taken into consideration focusing on book services.



	service 1	**service 2**	**service 3**	**service 4**	**service 5**
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. Relevant services.

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.



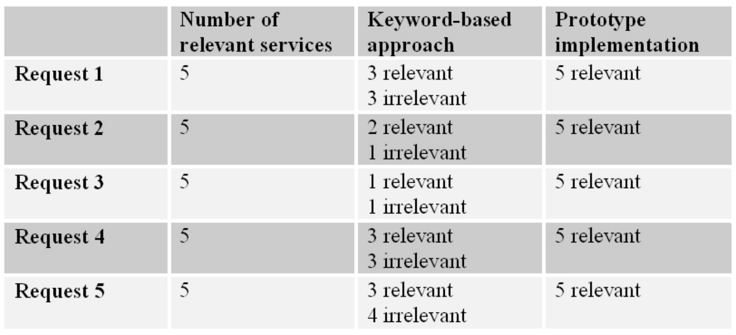
	service 6	**service 7**	**service 8**	**service 9**	**service 10**
context attributes	graph picture				
service attributes	**title**	**issue**	**name**	**product**	**composer**
	number	**owner**	**proprietor**	**pages**	**id**
	price	isbn	issn	book	**value**
	pages	drink	shop	meal	**pages**
	book	pixel	colour	point	book
	shop	font	paragraph	space	food
	colour	space	bold	font	colour

Table 2. Irrelevant Services.

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 service requests are similar to the one shown in Figure 11. The context parameters define the category of the service request which results in the split of 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 with the parameters as stated in Figure 12 which are exactly the parameters specified for service 1 in Table 1. Service request 2 is specified with the parameter of service 2 (Table 1) and so on. The context parameters are always the same as defined in Figure 11.

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 customized. Figure 14 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 fewer services that are irrelevant.



	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. Request and matches.

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 the context ontology is used from third-parties.

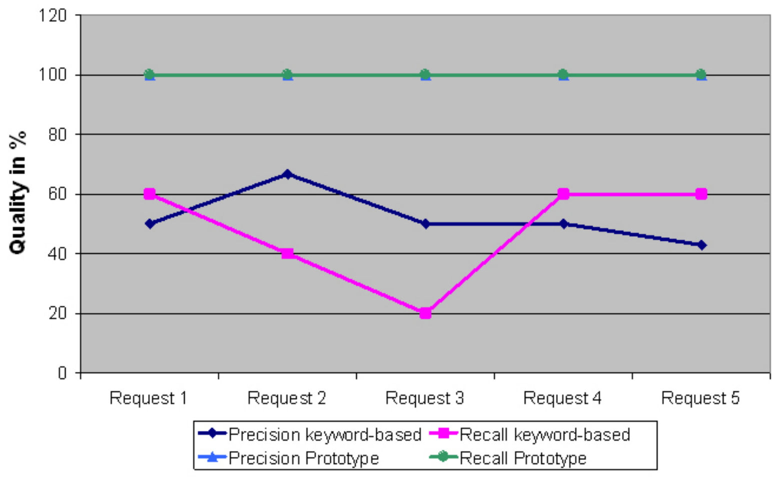


Figure 14. Evaluation of Precision and Recall Values.

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 using this semantic approach.

7.3 Match Score Measurements

The precision and recall measurements showed that the quality of the return of service matches is increased. However, to ensure a ranking process to indicate which returned services match best, the match score values were introduced. The service requests were the same as the ones from the precision and recall measurements (Figure 11) with the additional parameters for description and metadata. The weight values were set to 1/3 for service attributes, service descriptions and metadata information respectively. Table 4 shows the match scores for the five service requests.

Request	M_A	M_D	M_M	M_O
1	0.543	0.621	0.234	0.466
2	0.285	0.477	1	0.587
3	0.482	0.286	0.425	0.397
4	0.318	0.419	0.527	0.421
5	0.424	0.539	0.611	0.524

Table 4. Match Scores.

The match scores are vital in a matching system where due to the semantic matching process the quality of service matches is increased but whereby the user needs to be given a ranked service result set in order to indicate the best matches. Best matches are those with the highest similarity in comparison to the service request.

Figure 15 shows the distribution of match scores for the five service requests. The average match score is 0.479. This shows that providing the user with three categories (service attributes, service description and metadata information) allows specifying the service request more flexibly. The weight values can be specified by the user as a confidence value indicating which of the three categories are more important than the others.

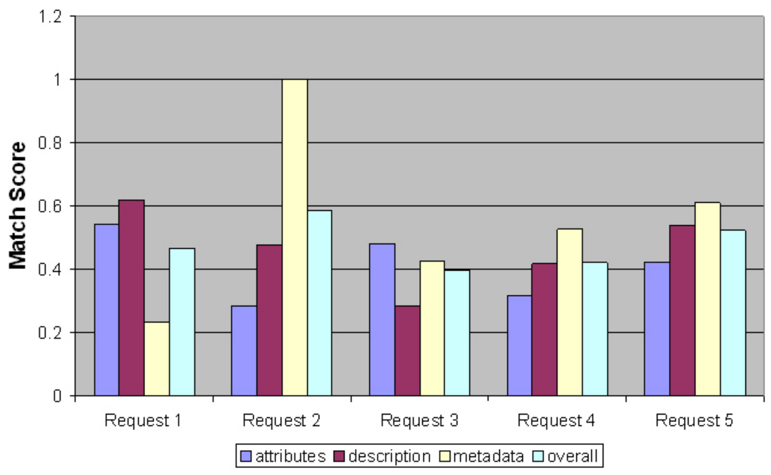


Figure 15. Match Scores Diagram.

7.4 Summary

As seen by the performance measurements the additional semantic feature results in a lower performance, however a higher precision for service matches is achieved. In particular, an acceptable performance is achieved for small and medium sized ontologies. The fact of the linear decrease of performance for growing ontologies needs to be considered carefully at design time. The choice of a “faster” reasoning engine might improve the matchmaking speed. Precision and recall measures showed the increase of quality of service matches, which was achieved by the customization and use of the context and services ontologies. The only problem is that the user might be overwhelmed by the number of service responses, therefore the match score values are vital. The match score values are a good measure to firstly rank the service responses and secondly restrict matches where the match score is smaller than a certain threshold value. The evaluation of the prototype showed a significantly improved precision of service matches.

8. CONCLUDING REMARKS

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 introduction of match scores serves as a selection criterion for the user to choose the best match. 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 for 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.

9. REFERENCES

1. J. McGovern, S. Tyagi, M. Stevens, S. Mathew, The Java Series Books - Java Web Services Architecture, Chapter 2, Service Oriented Architecture, 2003.
2. HTTP - Hypertext Transfer Protocol, W3C, 2004. http://www.w3.org/Protocols/.
3. Extensible Markup Language (XML), W3C, 2004. http://www.w3.org/XML/.
4. UDDI Technical White Paper. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf.

5. Web Services Description Language (WSDL) 1.1, W3C, 2004. http://www.w3.org/TR/wsdl.
6. SOAP Version 1.2, W3C, 2004. http://www.w3.org/TR/soap/.
7. W3C Working Draft, “Requirements for a Web Ontology Language”. http://www.w3.org/TR/webont-req/.
8. T.R. Gruber, ONTOLINGUA: A Mechanism to Support Portable Ontologies, Version 3.0, Technical Report KSL 91-66, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 1992.
9. M. Uschold, M. Gruninger, Ontologies: Principles, Methods and Applications, Knowledge Engineering Review, 1996, 11-2.
10. S.A. Ludwig, A Semantic Approach To Service Discovery In A Grid Environment, Ph.D. Thesis, Brunel University, UK, 2004.
11. A. ShaikhAli, O. Rana, R. Al-Ali, D.W. Walker, UDDIe: An Extended Registry for Web Services, *Proceedings of the Service Oriented Computing: Models, Architectures and Applications*, SAINT-2003, Orlando, USA, 2003.
12. U. Keller, R. Lara, A. Polleres, I. Toma, M. Kifer, D. Fensel, WSML Deliverable – WSMO Web Service Discovery, WSML Working Draft, 2004.
13. D.J. Mandell, S.A. McIlraith, 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, 2003.
14. R. Fikes, P. Hayes, I. Horrocks, DAML Query Language, Abstract Specification, 2002.
15. D. Chakraborty, F. Perich, S. Avancha, and A. Joshi, Dreggie: Semantic service discovery for m-commerce applications, *Workshop on Reliable and Secure Applications in Mobile Environment*, 20th Symposium on Reliable Distributed Systems, New Orleans, LA, October 2001.
16. S. Avancha, A. Joshi, T.W. Finin, Enhanced Service Discovery in Bluetooth, IEEE Computer 35(6): 96-99, 2002.
17. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda, Mapping Abstract Complex Workflows onto Grid Environments, *Journal of Grid Computing*, Vol. 1, No. 1, pp 9--23, 2003.
18. H. Tangmunarunkit, S. Decker, C. Kesselman, Ontology-based Resource Matching in the Grid - The Grid meets the Semantic Web, *Proceedings of the First Workshop on Semantics in Peer-to-Peer and Grid Computing*

(SemPG03), In conjunction with the Twelfth International World Wide Web Conference 2003, Budapest, Hungary, May 2003.
19. The XSB Research Group. http://xsb.sourceforge.net.
20. H. Lieberman et al. Out of context: Computer systems that adapt to, and learn from, context, *IBM system journal*, Volume 39, Numbers 3 & 4, MIT Media Laboratory, 2000.
21. A. Dey, Providing Architectural Support for Context-Aware applications, Thesis, Georgia Institute of Technology, November 2000.
22. M. Paolucci, T. Kawamura, T.R. Payne, K. Sycara, Semantic Matching of Web Services Capabilities, *Proceedings International Semantic Web Conference (ISWC 02)*, 2002.
23. JESS, Java Expert Systems Shell. http://herzberg.ca.sandia.gov/jess/docs/61/index.html.
24. C.L. Forgy, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Journal of Artificial Intelligence* 1982, 19-17-37.