A DAML-Based Repository for QoS-Aware Semantic Web Service Selection

A. Soydan Bilgin * †
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535
asbilgin@unity.ncsu.edu

Munindar P. Singh
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535
singh@ncsu.edu

Abstract

The Web is moving toward a collection of interoperating Web services. Achieving this interoperability requires dynamic discovery of Web services on the basis of their capabilities. The capability of a service can be properly determined by using not only its functional description (or service interface), but also its quality attributes as judged by previous users of the service. We develop a service repository that extends UDDI registries. This repository combines an ontology of attributes with evaluation data.

We base our repository on a new query and manipulation language based on DAML. Our language includes support for a rich set of operations, which are needed to maintain an attribute ontology, publish services, rate services, and select services based on their functional attributes as well as evaluations by others. We have implemented our approach and evaluated its practical completeness via a number of key query and manipulation templates.

1. Introduction

The Web services architecture supports the discovery and binding of services based on interfaces of services published by the providers. However, service interfaces are necessary but not sufficient for effective service selection. For example, a *DocToPdf Converter* Web service may take a *.doc* file as input and produce a *.pdf* file as output. However, this information about its interface may not be enough to assume a user who wants all hyperlinks or characters written in *Tahoma* format in his document to be converted properly.

The missing component is the expected behavior of a service, i.e., its *quality of service (QoS)*. QoS can possibly be predicted to some extent by knowing the implementation of a service, but examining implementations would violate the essential property of the services architecture. Thus, practically, a more reasonable way to judge QoS is to let service consumers evaluate the behavior of a service and to use such evaluations during service selection. The evaluation would involve a set of attributes, which would largely depend on the application domain [10]. Examples of attributes are availability, throughput, latency, security, broadness of feature set, price, location, and so on. Some well-known sites such as `bindingpoint.com` [1] and `salcentral.com` [14] already provide an interface to rank Web services.

Standards such as UDDI [16] and ebXML [11] address service discovery, but do not represent quality attributes of Web services. Another relevant effort is Web Ontology Language for Web services (OWL-S), which is an ontology description language for Web services. OWL-S provides a mechanism to describe the capabilities and properties of a Web service in a machine interpretable form and represents the subset of the quality attributes. In this respect, OWL-S is a valuable effort to represent Web service capabilities. However, what we need is a query language that Web service providers and Web service consumers can use to query resources in quality attributes ontology and service quality data, or insert resources into the same ontology and service repository.

Contribution. The main contribution of this work is the representation of services based on their quality attributes, and their selection based on a query language that respects these attributes. To this end, we develop a new query language based on DAML that accommodates several essential query and manipulation templates. This is implemented as a semantics and quality-sensitive enhancement to UDDI. Our query language uses the existing ontology language primitives, and can be used to query and manipulate the quality attributes ontology and the Web service quality data based on this ontology.

---

* Doctoral student.

† This research was partially supported by the National Science Foundation under grant ITR-0081742.

Organization. The rest of this paper is organized as follows. Section 2 provides the technical motivation and additional background for our approach, including a description of our ontology and the operation templates that must be supported by our language. Section 3 introduces our proposed query and manipulation language. Section 4 describes the implementation of our repository. Section 5 discusses the relevant literature. Section 6 summarizes our main contributions.

2. Motivation and Background

Our specific purpose is to express a Semantic Web services Query and Manipulation Language (SWSQL) that can be used to advertise and query quality attributes of Web services and evaluate it via a number of query and manipulation patterns. This purpose provides a sample usage of the proposed DAML query language.

Our work extends a query language for DAML, which was proposed by deVos [3]: hence we abbreviate it as dvQL. dvQL has a simple basis but can handle general queries by incorporating DAML class expressions. We first explain some of the insufficiencies of dvQL and then propose the necessary extensions and modifications to be able to query our sample query attributes ontology and the data stored in a relational database. We define a mapping between the relational database table structure and DAML classes, so agents use this mapping information to query the database as if the data were converted to DAML instances.

DAML provides representation for semi-structured knowledge objects with machine-processable data. It provides modeling primitives commonly found in frame-based languages while its formal semantics is defined in description logic. It is also the starting point for the Web Ontology Language [17]. Because of the greater availability of DAML-specific tools when we began this project, we used DAML instead of OWL for describing queries and ontologies. Our work can be easily migrated to OWL as its popularity increases in terms of tool support.

This section gives brief information about dvQL and lists a number of query and manipulation templates that we use to evaluate dvQL. We also present a sample quality attributes ontology we use to test the system built upon our language, SWSQL.

2.1. Quality Attributes Ontology

We incorporate a simple ontology for service categories and service quality attributes. In this ontology each service category corresponds to a daml:Class and each quality attribute corresponds to a daml:ObjectProperty. A hierarchical representation of a sample ontology is shown in Figure 1.

In this representation, ServiceCategories is the abstract service category with a subclass GenericServiceAttributes. Each service has one serviceKey (a unique identifier of the service provider in a public registry like Microsoft business UDDI Registry), one publishedRegistryUrl (the URL of the UDDI Registry), and one serviceProvider (the provider of the service, identified by the partyIdentifier). In real life, a service can be published in more than one registry, but for the sake of simplicity we use just one service key of the service.

Each quality attribute belongs to at least one service category. The range of each quality attribute is AttributeValues. The properties associated with AttributeValues are value, unit, predicate, numberOfSubmission, lastSubmissionTime, and submittedBy. According to our ontology, values of attributes can be submitted by either service providers or service consumers. Each service provider or service consumer party has a name and an identifier property.

2.2. Query and Manipulation Language Requirements

In order to realize a repository of the kind explained above, we would like to develop a query and manipulation language that supports the following main operation templates:

- T_1. Find the direct attributes (properties) of a specific service category.
- T_2. Find the transitive attributes of a specific service category.
- T_3. Find the direct subclasses of a service category (class).
- T_4. Find the transitive subclasses of a service category (class).
- T_5. Find the property values (e.g., domain) of an attribute (property).
- T_6. Insert a new service category (class).
- T_7. Insert a new attribute (property).

Each service corresponds to an instance of a service category class in our ontology. Our new language should also allow the following query and manipulation templates for the Web service repository:

- T_8. Find the attributes where a specific service has values.
- T_9. Find the attributes and their values for a specific service.
- T_{10}. Find the services which have a specific value for a specific attribute.

Proceedings of the IEEE International Conference on Web Services (ICWS'04)
0-7695-2167-3/04 $ 20.00 IEEE

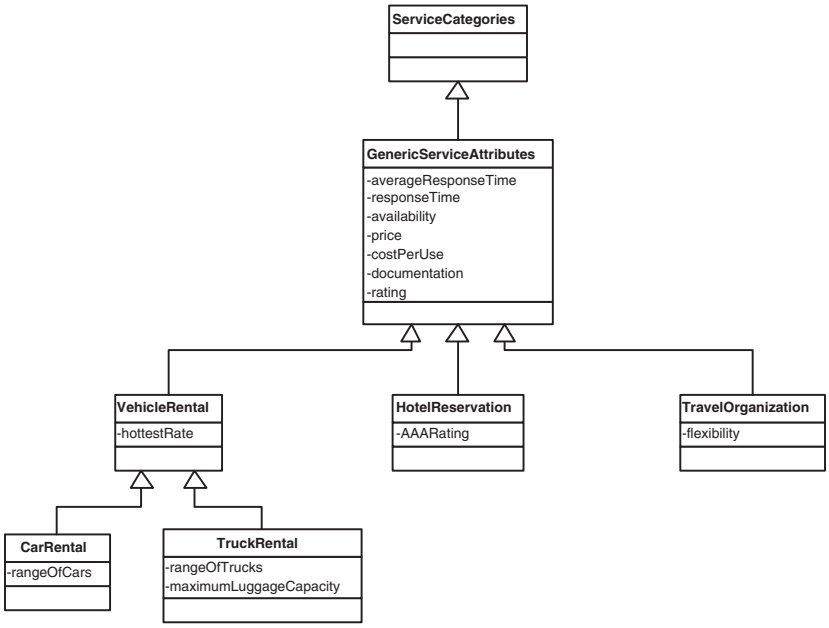

Figure 1. Representation of service categories and quality attributes ontology

T_{11}. Find the services having a given attribute.

T_{12}. Find the type of a service.

T_{13}. Find the services which are of a specific service category.

T_{14}. Find the services which have values for the attributes whose range/domain is a specific category.

T_{15}. Insert a new service.

T_{16}. Insert or modify values for an attribute of a specific service.

2.3. Basics of dvQL

As mentioned above, dvQL enables querying DAML instances. A query in dvQL is formulated with an expression of the form select <property expression> from <class expression>. The query results are triples. The from clause, which is an expression describing a DAML class, can be used for expressing complex DAML concepts. dvQL can be implemented over conventional (i.e., non-DAML) data sources, such as relational databases. Figure 2 shows the basis of the query language as expressed in DAML.

The result of the query in Figure 2 is the set of all statements with a subject of type Departments and a predicate of name or manager.

In general, the from clause in the query can accept any DAML class. dvQL uses daml:Restriction for narrowing the search space and daml:hasValue to reference a literal value or a resource. Multiple restrictions can be expressed by daml:subClassOf expressions. The boolean combination of all these restrictions yields the projected subject. It is also possible to enumerate the instances to be queried by using daml:oneOf property. Different classes can be joined by using the daml:hasClass primitive as in Figure 3. Further, dvQL has constructs to introduce new class expressions and to nest queries.

```
<Query>
        <select rdf:resource="#manager"/>
        <select rdf:resource="#name"/>
        <from rdf:resource="#Departments">
</Query>
```

Figure 2. Find the name **and the** manager **of all subjects of type** Departments

2.4. Limitations of dvQL

dvQL can handle the following two query templates well:

1. Select properties from the instances of a class having a given property.

Proceedings of the IEEE International Conference on Web Services (ICWS'04)
0-7695-2167-3/04 $ 20.00 IEEE

```
<Query>
  <select rdf:resource="#name"/>
    <from>
      <daml:Restriction>
        <daml:onProperty rdf:resource="#address"/>
        <daml:hasClass>
          <daml:onProperty rdf:resource="#street"/>
          <daml:hasValue> Avent Ferry
          </daml:hasValue>
        </daml:hasClass>
      </daml:Restriction>
    </from>
</Query>
```

Figure 3. Find the name **of all subjects, which has** address **property whose objects has** street **= "Avent Ferry"**

- Find the attributes where a specific service has values.
- Find the services having a given attribute.

2. Select properties from the instances of a class having a property with a specific value(s).
   - Find the attributes and their values for a specific service.
   - Find the services which have a specific value for a specific attribute.

The above query templates are important for our system. Obtaining attribute values from the services or checking whether a service has a value for the specified attributes is crucial. However dvQL does not meet our requirements, especially for some of the query templates listed above. Specifically, the following extensions are needed:

1. Queries should be at the semantic level, not only at the structure level. At the structure level, the (meta)data only consists of a set of triples. However, we require a query language that is sensitive to the semantics of the DAML and RDF Schema primitives. SWSQL needs to elaborate subsumption and equivalence relationships between classes and properties, e.g., SWSQL has to be aware of the transitivity of the subclass relation.
2. We need to answer the following queries, which use our ontology:
   - Find the subclass of VehicleRental service.
   - Find the properties whose range is AttributeValues.

   Because dvQL cannot query named DAML classes, it cannot evaluate the above queries. This situation is ideal for service instances, because named classes are a DAML language detail and will be transparent to the user while joining different classes by using daml:hasClass primitive. If we had stored the name of the classes in a relational database, it would have been infeasible to determine while inserting new instances whether the name for a DAML class instance has been used before. However, we should be able to query the ontology via the name of the class or property. For example, in Figure 4, we have a daml:Class with name VehicleRental and a DAML instance of type VehicleRental with name vehicleRental1.

```
<ont:VehicleRental rdf:ID="vehicleRental1">

    *****

<daml:Class rdf:ID="VehicleRental">
  <rdfs:label>VehicleRental</rdfs:label>
  <rdfs:subClassOf rdf:resource="#GenericAttributes"
</daml:Class>
```

Figure 4. The vehicleRental **class and an instance of the** vehicleRental **class**

   We may not build a query by using vehicleRental1 if we store service instances in a non-DAML data source. We should be able to build queries by using VehicleRental.

3. A mechanism to insert (meta)data expressed by DAML. This is one of the reasons we decided to use and extend dvQL. Because of its dependency on DAML primitives, it is very straightforward to insert metadata.

3. Semantic Web Services Query and Manipulation Language

Our system incorporates the Semantic Web services Query and Manipulation Language (SWSQL), which the service providers and consumers use to query, insert, or modify both the quality attributes ontology and the service description data based on this ontology. The ontology is stored as triples in memory and the service description data is stored in a relational database. We use a relational database because of the SQL-like functionality of SWSQL. A major advantage of using a relational database is that it provides a scalable off-the-shelf solution. For the rest of the chapter, the metadata part of our system corresponds to our ontology and the data part corresponds to our database.

3.1. Identification of Services

Service providers can advertise millions of services. If we abstract the system as a huge table built according to

Proceedings of the IEEE International Conference on Web Services (ICWS'04)
0-7695-2167-3/04 $ 20.00 IEEE

our ontology (metadata part), each service rating will correspond to a tuple containing attribute values in a relational model. In this respect, each service (class instance) should be uniquely identified and can be obtained from a real public registry (i.e., UDDI). We add an identifier term to our language. This term resembles the where term of SQL except that it can only be used for unique identifier properties.

3.2. Evaluating Class Instances

In dvQL, ⟨from RDF:resource = "class"⟩ is evaluated as *with a subject of type class*. This kind of structure always evaluates instances of a class (data part). However, dvQL has no constructs to evaluate the actual class itself (metadata part), which we need. This motivates the following terms:

- **RestrictedTo:** This is subclass of daml:Restriction and can have three properties: select, objectType, and subjectType. RestrictedTo is used for the type declarations of instances.
- **objectType:** This is a property with range of daml:Class. It is used to restrict the search space to triples that have object type = *class*.
- **subjectType:** This is a property with range of daml:Class. It is used to restrict the search space to triples that have subject type = *class*.

RestrictedTo inherits all the properties of daml:Restriction (i.e., daml:onProperty). The query in Figure 5 finds the serviceKey of all car rental services that have rangeOfCars property with value equal to "Average." If we do not need to query according to the type of the class, then RestrictedTo should not be used.

3.3. Querying Properties

A limitation of dvQL is its inability to query metadata. To rectify this problem, we first added RestrictedTo to get rid of the ambiguity between class and class instance projection in the from part. However, we still would not be able to answer two simple query templates listed below:

1. Find the range (domain) of a specified property.
2. Find resources whose range is a specified class.

The from term in dvQL can only include daml:Class. We first changed the range of this term to daml:Thing, so we are able to put properties in the from part of the query. For example, to find the domain of the properties whose ranges are AttributeValues in our sample ontology, we have the following query:

```
<Query>
  <select rdf:resource="http://vegas.csc.ncsu.edu:8080/wsap/
                ServiceTypes.daml#serviceKey"/>
  <from>
    <RestrictedTo>
      <subjectType
        rdf:resource="http://vegas.csc.ncsu.edu:8080/wsap/
                ServiceTypes.daml#CarRental" />
      <daml:onProperty
        rdf:resource="http://vegas.csc.ncsu.edu:8080/wsap/
                ServiceTypes.daml#rangeOfCars"/>
      <daml:hasClass>
        <daml:Restriction>
          <daml:onProperty
            rdf:resource="http://vegas.csc.ncsu.edu:8080/
              wsap/ServiceTypes.daml#value"/>
          <daml:hasValue
            rdf:resource="http://vegas.csc.ncsu.edu:8080/
              wsap/QualityRating.daml#Average"/>
        </daml:Restriction>
      </daml:hasClass>
    </RestrictedTo>
  </from>
</Query>
```

Figure 5. Find the property of services with the given type and the value for the given property

```
<Query>
   <select rdf:resource=" http://www.w3.org/2000/01/
                        rdf-schema#domain "/>
   <from>
     <daml:ObjectProperty>
       <daml:range
         rdf:resource="http://vegas.csc.ncsu.edu:8080
           /wsap/ServiceTypes.daml#AttributeValues" />
     </daml:ObjectProperty>
   </from>
</Query>
```

Figure 6. Find the domain of properties whose range is the given class

We use daml:ObjectProperty for the query in Figure 6, to narrow the search space of the subject resources to object properties.

3.4. Returning Name of Classes and Properties

dvQL is based on *triples* and returns all statements that include properties in the select part over the resources specified in the from part of the query. Instead of returning the result as triples, we return only values of the properties specified in the select part of the query as key-value pairs. This approach, which resembles resource-centric query languages, reduces the useless information that would be re-

Proceedings of the IEEE International Conference on Web Services (ICWS'04)
0-7695-2167-3/04 $ 20.00 IEEE

turned if we query services (data portion of the system) where we do not need the name of the subject or the object. As shown in Figure 4, we are not interested in the name of the resource vehicleRental1.

Querying the service data is just one side of the coin. On the other side of the coin, we need to query service metadata. While querying the service metadata, we need the obtain the name of classes and properties. For example, we add subjectRdfID as a property to return the name of all subject resources of triples as in Figure 7. Similar to subjectRdfID, we also define objectRdfID and predicateRdfID properties as part of our language syntax.

3.5. Transitive Properties

One of the most important features of our system are subsumption and equivalence relationships between concepts (classes or properties). DAML has no built-in primitives to query such relations. If we want to find the domain of a property by using daml:domain or RDF-Schema:domain properties in the select statement, we can only find classes that are declared to be the domain of a specific property. We cannot find the transitive domain of a property by using the current primitives. For example, if the domain of the availability property is GenericAttributes (as specified in our ontology) and VehicleRental is the subclass of GenericAttributes, we cannot figure out that VehicleRental is also the domain of the availability property by using the current language primitives. For this reason we added some primitives that can be used to query transitivity of properties. Each of these primitives is defined as a daml:Property. All of the properties below can either be used in the select or the from parts of a query:

- **transitiveDomain:** The domain is daml:Property and their range is daml:Class.
- **transitiveRange:** The domain is daml:Property and their range is daml:Class.
- **transitiveSubPropertyOf:** The domain is daml:Property and the range is daml:Property.
- **transitiveSubClassOf:** The domain is RDF-Schema:Resource and the range is daml:Class. The domain of this property is declared as RDF-Schema:Resource; because the select statement can only accept daml:Property. On the other hand, like daml:subClassOf, transitiveSubClassOf can also be used as a property whose domain is daml:Class. Both daml:Class and daml:Property are subclasses of RDF-Schema:Resource.

For example, the query in Figure 7 finds the name of classes, that are subclasses of VehicleRental:

```
<Query>
   <select rdf:resource="http://vegas.csc.ncsu.edu:8080/
                  wsap/dql.daml#SUBJECT_RDF_ID"/>
   <from>
    <daml:Class>
      <transitiveSubClassOf
           rdf:resource="http://vegas.csc.ncsu.edu:8080/
                  wsap/ ServiceTypes.daml#VehicleRental"/>
    </daml:Class></from>
</Query>
```

Figure 7. Find the names of resources that are subclasses the given class

3.6. Inserting New Data

We also need to insert meta(data). We added the insert property whose domain is Query and range is daml:Thing. The insert property resembles to the functionality of SQL insert function. If the meta(data) with the specified identity already exists, then it does not insert it; otherwise it inserts the new meta(data). Insertion is done at the class or the property levels. In the metadata, consistency checking is done through the name of the class or the property. In the database in which service data is stored, consistency is checked through the primary keys of tables, which can be mapped to the identifiers of the subjects, e.g., the serviceKey and the publishedRegistryUrl attributes of a service. We either insert a tuple to the database or we insert a new class and new property to our ontology. Updates are not allowed. We cannot merely change the specific column value, or the properties of the daml:Class and daml:Property resources. This restriction is due to the hard consistency requirement between the ontology and the data. For the insertion request, class or property definitions are given using the declare property. The query in Figure 8 inserts maximumBaggageCapacity into our sample ontology.

3.7. Intersection, Union and Except

dvQL uses daml:subClassOf in the from part of the query to achieve multiple restrictions. In effect, dvQL uses an implicit intersection. By contrast, SWSQL uses explicit operators such as unionOf and disjointWith to support other algebraic operations on result sets.

4. Implementation

We implemented the service repository as an RPC-based Web service. Just as UDDI is used to discover and publish Web services catalog information, SWSQL can be used to query quality attributes of a Web service as well as ontol-

Proceedings of the IEEE International Conference on Web Services (ICWS'04)
0-7695-2167-3/04 $ 20.00 IEEE

```
<Request>
   <declare>
      <daml:ObjectProperty
            rdf:ID="maximumBaggageCapacity">
         <rdfs:label>maximumBaggageCapacity</rdfs:label>
         <rdfs:domain
            rdf:resource="http://vegas.csc.ncsu.edu:8080/
                  wsap/ServiceTypes.daml#CarRental"/>
         <rdfs:range
            rdf:resource="http://vegas.csc.ncsu.edu:8080/
                  wsap/ServiceTypes.daml#AttributeValues"/>
      </daml:ObjectProperty>
   </declare>
   <evaluate>
      <Query>
        <insert rdf:resource="#maximumBaggageCapacity"/>
      </Query>
   </evaluate>
</Request>
```

Figure 8. Insert maximumBaggageCapacity **property to the ontology**

ogy of these quality attributes. Our repository also provides limited publishing functionality.

We store and process our ontology in memory. On the other hand, we store Web service data in a relational database as a non-DAML data source. There are two main reasons to use a relational database for storing our data portion of the system. First, relational databases scale well for large amounts of data. Second, mapping from DAML classes to database tables is trivial. The users of our repository can use the mapping information to query the database as if the data were converted to DAML instances.

The input to our service is the URL for the query file and the output is the URL for the answer file generated by the service. The input query file is expressed in DAML, so we first validate it by using the DAML Validator, which is available on `daml.org`. This validator uses Jena's ARP Parser to parse input files. After validation, we read the input query file by using DAMLJessKB, which converts a DAML file into a set of equivalent subject-predicate-object triples. We have used DAMLJessKB instead of the Jena DamlModel to read the query file because of its more structured representation of anonymous classes, which are heavily used in our queries. We also used DAMLJessKB to figure out subsumption and equivalence relations.

After parsing the input query file, if we have to query the metadata, we use the Ontology Processor. The Ontology Processor is initialized by reading our quality attributes ontology into main memory by using DAMLJessKB package, so that we obtain the triple-based representation of our ontology. These triples are then asserted into Jess [5] knowledge base and the query is applied on this knowledge base. The usage of the Jess and the DAMLJessKB reduced our development time and facilitated reasoning about subsumption and equivalence relations between concepts. We also used the Jena DAML API to insert new classes and properties into the ontology.

If we have to query the data, which corresponds to a MySQL database, then we use the SWSQLtoSQL Mapper that converts a SWSQL query into an SQL query. SWSQLtoSQLMapper relies on the relational schema of the underlying database to generate SQL statements. This module also uses the methods of the Ontology Processor to make inferences.

5. Related Work

RQL (RDF Query Language) is a language for retrieving information represented in RDF and RDF Schema from the Web [7]. It adopts the functionality of XML query languages to RDF and RDF Schema description bases. RQL doesn't provide inference support for DAML description bases and update support to modify the contents of existing ontology and the data. Our query language mainly differs from RQL by its syntax, which depends on DAML primitives.

RDQL (RDF Data Query Language) uses SQL-like constructs for the query description [6]. RDQL regards the RDF model as a set of triples. It can only query at the structure level. Sirin et al. present a database agent that translates RDQL queries into SQL queries by defining a mapping function from DAML classes to database table structure [15]. The functionality of the database agent resembles our SWSQLtoSQLMapper module, but we provide the mapping from a query described by DAML primitives to SQL and we consider the results of inference being done via the constructs such as sameClassAs, samePropertyAs, subClassOf, and subPropertyOf while mapping.

RDF Query Specification [9] is the first proposed SQL style approach, viewing RDF description bases as a relational database. It resembles RDQL in terms of querying capabilities except that RDF Query Specification doesn't cover RDF Schema description bases. The specification allows the usage of RDF primitives while constructing queries analogous to how SWSQL uses DAML primitives.

DARPA Dql [4] provides a description model, which allows hypothesizing an object by using a query premise. It allows <if... else...> kind of queries. DARPA Dql is a kind of query description language for rule engines like Jess. DARPA Dql is a language for querying and reasoning metadata instead of data. Every subject and object is referenced by its name.

SWSQL resembles to RDQL in terms of SQL-like approach, but we have a different query description syntax, which uses DAML primitives and doesn't assume an underlying DAML-based repository. Like RQL, we also provide

Proceedings of the IEEE International Conference on Web Services (ICWS'04)
0-7695-2167-3/04 $ 20.00 IEEE

basic inference support and subquerying where the output of the subquery can be the input for the outer query.

In terms of service discovery based on Web service quality, several extended UDDI architectures are proposed [2] [12] [13]. However, these proposals overlook an important essential requirement for QoS-aware service discovery and selection architecture, which is an enhanced query mechanism for service quality data that also respects the semantics of the quality attributes [8] [13]. We also respect the semantics of the quality attributes during the representation of the query, not during the representation of the Web service data so that we can query the service quality data independently of the way this data is stored.

6. Main Contributions and Directions

This paper developed an approach for selecting services based on their semantics as well as their quality as judged by users. It proposed a service repository and a new query and manipulation language as a basis for this repository. This query language incorporates several query templates that are geared for querying ontologies of services as well as repository of ratings of services by users. Existing query languages are not geared toward service quality scenarios and do not adequately cover the desired query templates.

SWSQL uses DAML primitives for describing queries, but goes beyond previous DAML query languages to facilitate querying quality attributes ontology and the Web service descriptions expressed by the concepts from this ontology. Our repository uses a hybrid architecture consisting of a relational database for services and main memory for an ontology. We also provided a simple methodology for representing Web services with their quality attributes and identified the required query and manipulation templates to accommodate use cases for service selection and insertion. SWSQL can be the starting point for a query description standard that can be used among distributed agents on multiple data sources.

Porting this approach to OWL would be straightforward but practically valuable. An important enhancement would be to adapt our service repository to accommodate OWL-S. OWL-S is an OWL ontology for services that includes abstractions for service selection and composition. It provides an approach for modeling attributes with which services may be evaluated.

References

[1] bindingpoint.com, 2001. www.bindingpoint.com.

[2] Z. Chen, L.-T. Chia, B. Silverajan, and B.-S. Lee. UX: An architecture providing QoS-aware and federated support for UDDI. In *Proceedings of the 1st International Conference on Web Services*, pages 171–176, 2003.

[3] A. deVos. An RDF query language based on DAML, 2002. http://www.langdale.com.au/RDF/DAML-Query.html.

[4] R. Fikes, P. Hayes, and I. Horrocks. DAML query language (DQL). Abstract specification, DARPA, 2003.

[5] E. J. Friedman-Hill. *Jess in Action: Rule-Based Systems in Java*. Manning, Greenwich, CT, 2003.

[6] HPL. RDQL: RDF data query language, 2002. http://www.hpl.hp.com/semweb/rdql.htm.

[7] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A declarative query language for RDF. In *Proceedings of the Eleventh International World Wide Web Conference*, pages 592–603, Honolulu, Hawaii, 2002. ACM Press.

[8] K. Lee, J. Jeon, W. Lee, S.-H. Jeong, and S.-W. Park. Qos for Web services: Requirements and possible approaches. Working draft, World Wide Web Consortium, 2003.

[9] A. Malhotra and N. Sundaresan. RDF query specification. Technical report, IBM, 1998.

[10] E. M. Maximilien and M. P. Singh. Conceptual model of Web service reputation. *ACM SIGMOD Record*, 31(4):36–41, Dec. 2002.

[11] OASIS. ebXML, 2001. http://www.ebxml.org/.

[12] S. Pokraev, J. Koolwaaij, and M. Wibbelse. Extending UDDI with context-aware features based on semantic service descriptions. In *Proceedings of the 1st International Conference on Web Services*, pages 184–190, 2003.

[13] S. Ran. A framework for discovering Web services with desired quality of services attributes. In *Proceedings of the 1st International Conference on Web Services*, pages 208–213, 2003.

[14] salcentral.com, 2001. www.salcentral.com.

[15] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of Web services using Semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure Workshop in Conjunction with ICEIS*, 2003.

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

[17] F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web ontology language reference. Working draft, World Wide Web Consortium, 2003.

Proceedings of the IEEE International Conference on Web Services (ICWS'04)
0-7695-2167-3/04 $ 20.00 IEEE