A QoS-based Search Engine for Semantic Web Services

Gustavo Fortes Tondello and Frank Siqueira

Federal University of Santa Catarina
Department of Informatics and Statistics
Florianópolis, SC – Brazil – 88040-900
{fortes,frank}@inf.ufsc.br

Abstract. This paper presents a Semantic Web Services search engine that focus on the discovery of Web Services that fulfill a defined set of QoS constraints. Our approach relies on the use of the QoS-MO ontology, which provides means for specifying the QoS characteristics of Web Services, and the SPARQL language, which allows the specification of queries for discovering Web Services based on a set of QoS constraints. A prototype of this search engine has been developed, with both a programming and a Web-based interface, resulting in a simple and efficient implementation which validates the proposed mechanism.

Key words: Semantic Web Services, QoS, OWL-S, Search, Discovery.

1 Introduction

Semantic Web Services are one of the core technologies employed for building the Semantic Web, which is "an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [1]. The well-defined semantics of these services makes them suitable for automatic publication, discovery, composition and execution.

A few standards for Semantic Web Services description have been defined. However, there is still the need for a well-defined standard to specify QoS (Quality of Service) characteristics of Web Services and standards that allow the construction of search engines which are capable of finding Web Services that, besides providing the desired functionality, fulfill some specific QoS constraints that are specified by the Service's clients as sets of QoS requisites.

In this paper we introduce a Web Services search mechanism which is built upon the QoS-MO ontology for QoS-enabled Web Services specification [11]. Our mechanism relies on the combined use of a reasoner to identify class hierarchies and compositions that are specified on the ontology, and SPARQL [13] queries to find the Web Services that match specific criteria. This choice results in an approach that makes possible the semantic discovery of Web Services while keeping a simple implementation, since there is no need for a complex semantic matchmaking algorithm: we only need to convert the search constraints to a SPARQL query.

The remainder of this paper is structured as follows. Section 2 presents the

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 589-598, 2008.

technologies that were used for the definition of QoS-MO. Section 3 presents a summarized description of the QoS-MO Ontology, which is the basis for the construction of the Search mechanism. Section 4 describes the Semantic Web Service search mechanism that uses the QoS characteristics expressed with QoS-MO to find the Web Services. Section 5 presents some related works in this research field. Finally, section 6 presents some conclusions and perspectives for future work.

2 Background

This section presents the technologies, languages and ontologies that were employed to build the semantic framework for the QoS description of Web Services which was used to construct the QoS-enabled Semantic Web Services search mechanism.

2.1 Ontologies

The term *ontology* was created in the field of Philosophy, where it refers to a branch of metaphysics that studies existence. In Software Engineering, an ontology may be described as a specification of a conceptualization or a description of the concepts and relationships that may exist for an agent or a community of agents [4].

Ontologies are being used in the field of Software Engineering to allow the sharing and reuse of knowledge. When there is an ontology available for a particular domain, software agents may be written according to something that Gruber has called an "ontological commitment" [4], i.e., the commitment of using the vocabulary defined by such ontology. That way, different agents that share the same ontology can communicate between themselves since their vocabulary is understood by everyone.

2.2 OWL-S

OWL (Web Ontology Language) [14] is an XML/RDF-based language for ontology description that was designed to be compatible with the World Wide Web and the Semantic Web. The OWL language is used to define Classes, Properties, relationships between classes, cardinalities, equality relations, type and characteristics of properties and enumerated classes. A class is defined in OWL as a set of requisites that an individual must fulfill to be considered an instance of the class.

OWL-S [9] is an OWL ontology that is used for the description of properties and capabilities of Web Services. In OWL-S, a service is described as an instance of the *Service* class. The key part of the description of a service is its profile, which is described by an instance of a subclass of the *ServiceProfile* class. The standard profile description, using the *Profile* class, includes the description of the organization that provides the service, the functionalities that it provides and its characteristics. The definition of a service may contain a model (*ServiceModel* class) which specifies the processes of a service, i.e., how should a client interact with it, and the grounding (*ServiceGrounding* class) that is the mapping between the abstract specification of the service and its concrete specification.

2.3 QoS Specification Languages

Several existing QoS Specification languages have been analyzed, trying to take the best characteristics of each one and adapt them to be employed in the ontology that is used as the description framework of QoS characteristics and Constraints.

The analyzed specification languages were: Quality of Service Modeling Language (QML) [3], Web Service Level Agreement (WSLA) Language [7], QoS Specification Language (QSL) [10] and QoS Description Language (QDL) [16]. Despite providing mechanisms for QoS specification, these languages lack the flexibility provided by an ontology for dealing with the semantic meaning of QoS constraints.

2.4 OMG QoS Metamodel

The OMG QoS framework [8] metamodel defines the abstract language for a modeling language that supports modeling general QoS concepts. It was designed as a metamodel to the definition of the QoS UML Profile. This metamodel is widely accepted in its area of research due to its adequacy for modeling QoS concepts, and is also adopted as the basic reference for the definition of the QoS-MO ontology.

The OMG QoS metamodel defines three packages. The QoS Characteristics package contains the model elements for the description of QoS Characteristics, QoS Dimensions (different ways to quantify a characteristic) and QoS Contexts (quality constraints that combine several QoS Characteristics). The QoS Constraints package includes the modeling elements for the description of QoS contracts and constraints. The QoS Levels package includes the modeling elements for the specification of QoS modes and transitions.

2.5 QoS Modeling Ontologies

Before deciding to define the QoS-MO ontology, the authors have analyzed other existing ontologies for QoS specification, and have identified a series of limitations. The analyzed ontologies were:

- The QoSOnt ontology [2], which defines a model for the specification of QoS for Semantic Web Services. It allows the association of a QoS specification with an OWL-S service profile.
- The DAML-QoS ontology [15], which defines a model for the specification of QoS parameters and profiles. DAML-QoS allows the specification of a QoS profile as a set of QoS characteristics.
- The OWL-Q ontology [6], which is an upper level ontology that extends OWL-S to describe QoS metrics and constraints. Using OWL-Q it is possible to define complete QoS specifications, and it also provides means to specify unit conversion and composite metrics.

In our analysis, these modeling ontologies lack the ability to specify complex QoS requisites like dimension mapping or complex QoS profiles, or depend on complex matchmaking algorithms to find Web Services that fulfill specific QoS requirements.

3 The QoS-MO Ontology

The Quality of Service Modeling Ontology (QoS-MO) [11] is an upper-level ontology for the description of QoS Characteristics and Constraints of Web Services. To instantiate a QoS description of a particular Web Service, one must create a new ontology that will import the QoS-MO ontology, extend its classes as needed and create the required individuals. A Web Service's functional and non-functional (QoS) specifications may be described on a single ontology or on two separated ones. Figure 1 depicts the hierarchy of ontologies employed for defining a Web Service.

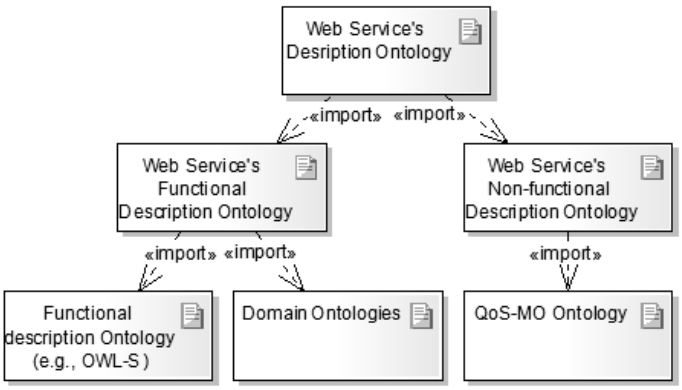

Fig. 1. The Ontologies for the description of a Web Service.

The QoS-MO ontology is composed of three different models that address all requirements for a complete QoS specification. We present here only a summarized description of each of these models. The complete specification can be found in [11].

3.1 QoS Characteristics

This model contains the classes to specify non-functional characteristics of Web Services. The class *QoSCharacteristic* is the main class in the QoS-MO ontology for the definition of quantifiable characteristics of services. Characteristics may be general like latency, throughput, availability, reliability, security and accuracy, or domain specific ones. QoS characteristics may be grouped in categories using the class *QoSCategory*. The class *QoSCharacteristic* is a subclass of *QoSContext*. A QoS Context is a definition of a set of one or more QoS Characteristics.

Classes *QoSCharacteristic* and *QoSCategory* may be extended or specialized, creating hierarchical taxonomies of characteristics and categories.

The dimensions for the quantification of characteristics are modeled using class *QoSDimension*. One QoS Characteristic may have more than one way of measuring it, or it may require more than one dimension for its quantification. Examples of dimensions for latency are (from [8]): minimum latency, maximum latency and jitter. The measured values of dimensions are modeled using class *QoSValue*.

The *QoSDimensionMapping* class is employed to define mapping scripts between characteristics, allowing for search mechanisms or automatic QoS negotiation middleware to identify a dimension even if its value is not given for a particular characteristic, provided that there is a mapping script between it and any other dimensions of the same characteristic or even of other characteristics.

3.2 QoS Constraints

This model contains the classes to specify the QoS constraints that represent the characteristics of a particular Web Service. There are three types of QoS Constraints: *QoS Offered, QoS Required* and *QoS Contract*. The QoS Offered and QoS Required constraints can be used by provider or client services to specify their QoS constraints. A QoS Contract constraint represents the final quality agreed between two services on an assembly. A QoS Constraint must reference the QoS Context or Contexts that provide the reference expressions and values associated to the constraint.

3.3 QoS Levels

This model is used to represent the different modes of QoS that a service can support. The QoS Constraints associated with a *QoS Level* represent the constraints that the service must satisfy to operate on such level. When a service is no longer capable of operating at a given level, for example, when it is receiving more requests and its maximum response time constraint will have to change, a *QoS Transition* occurs. When this happens, it is likely that the services involved will have to renegotiate their execution parameters and QoS Contracts. A QoS Transition definition may specify all the adaptation actions required for the transition.

3.4 OWL-S Extension Example

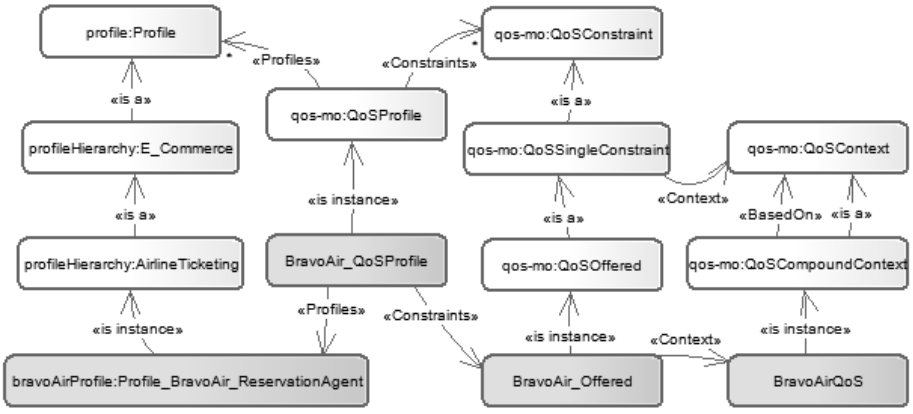

Fig. 2. Extension of the BravoAir Web Service OWL-S profile with QoS constraints.

The QoS-MO Ontology extends OWL-S with QoS related constraints which may refer to a Service as a whole or to a specific Process of a Service.

Figure 2 depicts a complete example of an extension to the BravoAir Web Service profile [9] represented by the *Profile_BravoAir_ReservationAgent* individual. The QoS Profile *BravoAir_QoSProfile* links the service's profile with the QoS Offered constraint *BravoAir_Offered* which is defined with one QoS Context *BravoAirQoS*.

4 The QoS-based Search Engine for Web Services

We have developed a search mechanism for locating Web Services modeled with OWL-S that fulfill some specific QoS requirements. This mechanism writes SPARQL [13] queries to search for QoS models that are within the specified constraints and return the selected Web Services. It consists of an API that allows its client to discover the QoS Characteristics that are available on the QoS specification ontology and execute queries for Web Services with a particular set of QoS constraints.

We have also developed a web-based search tool where the user specifies the QoS constraints he/she expects to be fulfilled by Web Services. These constraints are sent to the Query API, which executes a SPARQL query in order to locate Web Services that satisfy them. The returned results are displayed on the Web interface.

Figure 3 presents an overview of the Web Services publication and discovery processes using the QoS-MO ontology to specify Web Services' QoS constraints.

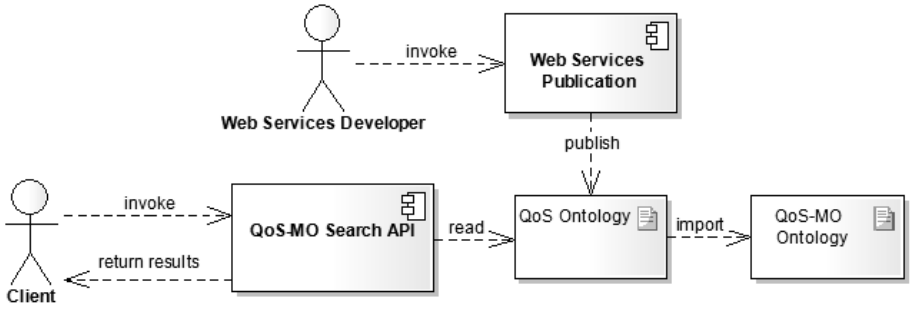

Fig. 3. Overview of Web Services publication and discovery processes.

4.1 The Query API

The Query API of the proposed search mechanism was developed in Java using the Jena and ARQ APIs [5]. It can be used directly by another Java application or by any other application using its Web Service interface. It is composed of three parts:

- The first part is able to read the tree of QoS Characteristics that is available on the ontology and return it to the caller. This allows the construction of automatic QoS negotiation middleware which is able to dynamically discover the characteristics

that are available on a system or a GUI that allows the user to choose which QoS characteristics he/she is searching for. This was done in our Web search tool.

- The second part allows the caller to write and execute a Query. Query execution is accomplished by a component that converts the requested QoS constraints into a SPARQL query and executes this query on the ontology that contains the QoS description of the available Web Services.
- The third part is a component that retrieves all the Web Services matched by the query and returns them to the caller.

Before executing SPARQL queries, the search mechanism loads the ontology and runs the reasoner. Reasoning is needed to identify the class hierarchies of QoS characteristics on the ontology, as well as the compositions of QoS contexts, constraints and levels. After the reasoning, the queries are able to find QoS profiles even if there are a few levels of specialization or composition in their definitions.

The reasoner uses a subset of OWL inference rules that is available on the Jena API, which is called *Micro OWL*. This subset is sufficient for the implementation of our search mechanism, since it supports the two inference rules that are needed: the *rdfs:subClassOf* rule, to identify class hierarchies, and the *owl:TransitiveProperty* rule, to identify compositions, which are defined by transitive properties.

4.2 SPARQL queries

A QoS requirement is specified as a set of a QoS characteristic, a QoS dimension, an operator and a value to restrict the value of the dimension. From the specified requirements, the query execution component of the API generates and executes the SPARQL query. As an example, the following is the SPARQL query to find a Web Service's QoS constraint offering a *ResponseTime* characteristic with a *MaxResponseTime* dimension whose value is lower than 10 seconds.

```
SELECT ?constraint ?context ?charact ?dimension ?value
WHERE {
  ?constraint rdf:type qos-mo:QoSOffered .
  ?constraint qos-mo:Context ?context .
  ?context rdf:type qos-mo:ResponseTime .
  ?context ?predicate ?dimension .
  ?predicate rdfs:range qos-mo:MaxResponseTime .
  ?dimension qos-mo:value ?value .
  FILTER (?value < 10)
}
```

4.3 The Web Search Tool

The web search tool we have built is able to present the characteristics available on the ontology for the user, using the Characteristics discovery feature of the Query API. Then, the user can construct its search query using the web-based interface. After he/she completes the search query, the Search button calls the query execution

and Web Service listing features of the API to show another page with information about the Web Services that matched the search criteria.

4.4 Query Performance

We have tested the performance of our search mechanism using SPARQL queries to verify the efficiency of our approach. Tests were conducted on a computer with an Intel Core 2 Duo 2.4 GHz processor and 2 GB of RAM running Fedora 7.

The first test evaluated the ontology loading time. We found that loading time with the *Micro OWL* inference rule set is acceptable, since the Jena API was able to load an ontology with 10,000 Web Services in a reasonable time.

The second test was the performance of the search execution. We have found that, with the increase in the number of Web Services in the ontology, the search time grows at a lower rate. As shown by Figure 4, the search over an ontology with 1,000 Web Services took less than two seconds, and if the number of services grows tenfold, the search time is less than four times greater.

The test results, presented in Figure 4, also show the memory that was allocated to load the ontology and the search results.

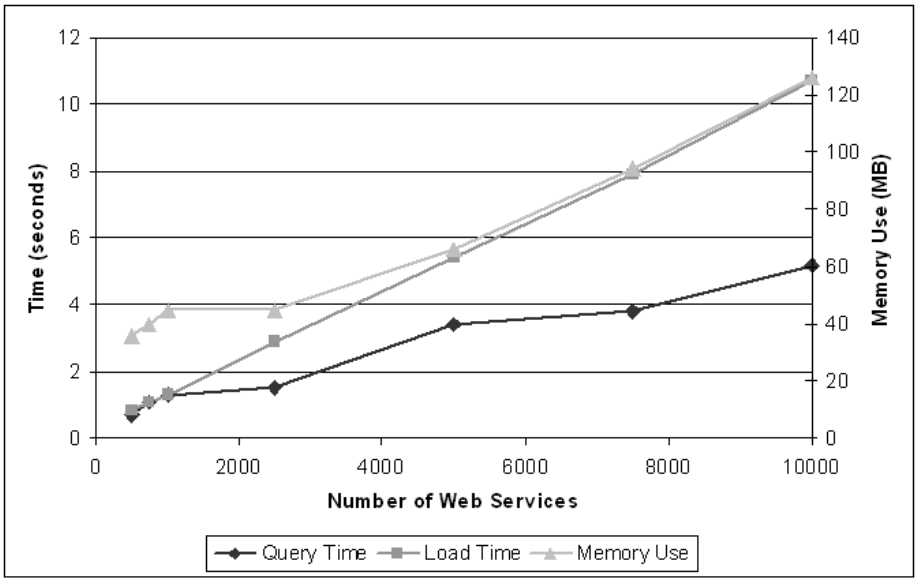

Fig. 4. Results of performance and memory allocation tests for the execution of SPARQL queries over a QoS-MO ontology of Web Services.

Additional tests were conducted to verify the efficiency of the search mechanism with concurrent queries. Test results, which will not be presented here due to space constraints, show that query efficiency drops with the increase of the ontology size or the number of concurrent queries, but at a lower rate. Our test machine was not really

prepared to support a highly concurrent environment, however it was able to hold a good throughput level. To run 1,000 concurrent queries, the maximum reached throughput was 77 queries per second with an ontology of 2,500 Web Services, and the minimum was 7 queries per second with an ontology of 10,000 Web Services.

5 Related Work

Zhou et al. have defined a model for specification of QoS parameters and profiles and an algorithm that uses a DL reasoner to Web Services discovery [15]. Nevertheless, there is no possibility to extend an existing profile or create a profile composition. The main flaw of DAML-QoS is that it uses the cardinality restrictions between a QoS profile and its metrics to set the values of QoS characteristics due to restrictions of its DL reasoner, thus limiting their values to non-negative integer numbers.

Dobson et al. have defined a model for the specification of QoS for Semantic Web Services and an UDDI+XML solution for service discovery [2]. However, this model does not provide any means to specify a QoS profile from a set of QoS characteristics, nor to reuse or extend a previous specification.

The OWL-Q ontology extends OWL-S to describe the possible parts of QoS metrics and constraints and Kritikos and Plexousakis have proposed a semantic matchmaking algorithm for Web Service discovery [6]. Using OWL-Q it is possible to define complete QoS specifications, and it also provides means to specify unit conversion and composite metrics. Unlike OWL-Q, our approach relies on the simple implementation of a SPARQL query instead of a complex semantic algorithm.

Vu et al. describe a QoS-enabled Semantic Web Service discovery framework [12]. They define a complete model that comprises QoS specification and verification framework, a classification mechanism to index QoS characteristics, an algorithm to discover Web Services and a distributed framework to parallelize query execution. Their framework is more complete than ours, but is also more complex. Our approach is a suitable and interesting solution for small scale environments since it is extremely simple to implement and maintain, while their framework will probably be more efficient in large scale environments.

6 Conclusion

In this paper we have presented a QoS-Enabled Search Engine for Semantic Web Services. The search mechanism consists on an API that is able to discover the QoS characteristics of an ontology and run queries that search for Web Services that meet specific QoS requisites. The ontology model used to describe QoS characteristics is the QoS-MO ontology.

Our approach is simple and easy to implement and maintain. Queries are conducted through the conversion of the user query to a SPARQL query. Before the execution of the queries, the QoS specification ontology goes through a reasoner to identify specialization and composition hierarchies of specifications.

We have built a Web search tool to demonstrate our approach and have conducted

some tests to measure the performance of the search engine. Despite the simplicity of the query mechanism, our tests showed that this approach is viable, since we were able to get manageable response times even for ontologies with 10,000 Web Services running 1,000 concurrent queries.

On the future, the search mechanism will be submitted to other tests to better evaluate its efficiency to answer a great number of concurrent requisitions. Another interesting study would be to test the performance of the search mechanism if the Ontologies were stored on a relational database instead of an in-memory OWL file. This is supported by the Jena API to improve scalability, allowing the utilization of relational databases features like indexing and replication to improve performance. Finally, it would be important to conduct experiments to compare the performance of the proposed search mechanism with the related proposals to evaluate the efficiency of each one of the ideas presented in this research area.

References

1. Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web. Scientific American, May. (2001)
2. Dobson, G., Lock, R., Sommerville, I. QoSOnt: a QoS Ontology for Service-Centric Systems. 31st Euromicro Conference on Software Engineering and Advanced Applications (Euromicro SEAA '05). Porto, Portugal. (2005)
3. Frølund, S., Koistinen, J. QML: A Language for Quality of Service Specification. http://www.hpl.hp.com/techreports/98/HPL-98-10.html (1998)
4. Gruber, T. R. A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220. http://tomgruber.org/writing/ontolingua-kaj-1993.htm (1993)
5. Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net/ (2008)
6. Kritikos, K., Plexousakis, D. Semantic QoS Metric Matching. 4th European Conference on Web Services (ECOWS '06), December 2006, pp.265-274. (2006)
7. Ludwig, H., Keller, A., Dan, A., King, R.-P., Franck, R. Web Service Level Agreement (WSLA) Language Specification. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf (2003)
8. Object Management Group. UML Profile for Modeling QoS and FT Characteristics and Mechanisms Specification, v1.0. (2006)
9. OWL-S Coalition. OWL-S 1.1 Release. http://www.daml.org/services/owl-s/1.1/ (2004)
10. Siqueira, F. Especificação de Requisitos de Qualidade de Serviço em Sistemas Abertos: A Linguagem QSL. 20th Brazilian Symposium on Computer Networks (SBRC'2002). Búzios - RJ, Brazil. (2002)
11. Tondello, G., Siqueira, F. The QoS-MO Ontology for Semantic QoS Modeling. Proceedings of the 23rd Annual ACM Symposium on Applied Computing, Fortaleza, Brazil. (2008)
12. Vu, L.-H., Hauswirth, M., Porto, F., Aberer, K. A Search Engine for QoS-enabled Discovery of Semantic Web Services. Special Issue of the International Journal on Business Process Integration and Management (IJBPIM), Vol. 1, No. 4, pp.244–255. (2006)
13. W3C. SPARQL Query Language for RDF. Candidate Recommendation. http://www.w3.org/TR/rdf-sparql-query/ (2007)
14. W3C. Web Ontology Language (OWL). http://www.w3.org/2004/OWL/ (2004)
15. Zhou, C., Chia, L., Lee, B. DAML-QoS ontology for Web services. IEEE International Conference on Web Services (ICWS'04). San Diego, California, USA, pp.472-479. (2004)
16. Zinky, J., Bakken, D., Schantz, R. Architectural Support for Quality of Service for CORBA Objects. Theory and Practice of Object Systems, Vol. 3(1). (1997)