A Search Engine for QoS-enabled Discovery of Semantic Web Services

Le-Hung Vu, Manfred Hauswirth, Fabio Porto, and Karl Aberer
School of Computer and Communication Sciences,
Ecole Polytechnique Fédérale de Lausanne (EPFL),
CH-1015 Lausanne, Switzerland

Abstract: Directory services are a genuine constituent of any distributed architecture which facilitate binding attributes to names and then querying this information, i.e., announcing and discovering resources. In the web services domain this functionality is provided by UDDI or some of its semantic extensions/counterparts for semantic-based resource description and discovery. All of these approaches deal with functional aspects, i.e., interfaces, of web services but do not address non-functional properties such as quality-of-service (QoS). However, in a business environment, usually the non-functional properties are actually the first ones to be applied in deciding whether a specific resource will be used and only if the QoS is considered satisfactory, the functional requirements are being looked at. In this paper we address this problem by providing an approach to semantic description and discovery of web services which specifically takes into account QoS. Our approach allows web service providers to describe the functional and QoS properties of semantic web services and users may discover services based on both types of information with results ranked according to their preferences. As QoS specifications by the provider are basically claims, our approach includes a user feedback mechanism which allows the users of a web service to provide feedback on the actually perceived QoS. This information together with information from trusted third parties (rating agencies) is fed in to a robust statistical trust and reputation model, which takes into account attacks such as bad-mouthing, collusion, etc., and then allows us to provide an accurate picture of the actual QoS to user. Our search engine is based on an algebraic discovery model and uses adaptive query-processing techniques to parallelize expensive operators. Architecturally, the search engine can be run as a centralized service for small-scale environments or can be distributed among any number of cooperating registry providers. To enable scalable discovery we devise a peer-to-peer-inspired distribution strategy which enables fault-tolerance and allows registries to maintain full control and confidentiality on the information they provide.

Keywords: Semantic web services; service discovery; QoS; trust; reputation

1 INTRODUCTION

High-quality means for resource discovery are an essential prerequisite for service-oriented architectures. Developers need to be able to discover web services and other resources for building their distributed applications. In the upcoming Semantic Web, discovery becomes even more important: Assuming that web services are being semantically annotated, the tasks of choreographing and orchestrating *semantic* web services is supposed to be delegated to the computer. This requires expressive means to describe web services semantically which is a major ongoing research effort, for example, WSMO (1), but also puts more complex requirements onto discovery.

Typically, most of the ongoing efforts address functional aspects. However, in a business environment, usually the non-functional properties are actually the first ones to be applied in deciding whether a specific resource will be used and only if the QoS is considered satisfactory, the functional requirements are being looked at. In this paper we address this problem by providing an approach to semantic description and discovery of web services which specifically takes into account QoS. Our service discovery framework includes a semantic QoS description model which is suitable for many application domains, is simple yet expressive and could be used for semantically describing the QoS requirements of the users and the advertised Service Level Agreement (SLAs) of the providers, taking into account

the semantic modeling of environmental factors on QoS parameter values and the relationships among them.

In contrast to functional specifications which can easily be verified, the assessment of non-functional properties such as performance, throughput, reliability, availability, trust, etc. is much more complicated. Basically the QoS described in a specification has to be seen as a claim. For assessing the actually provided QoS further evaluation infrastructures need to be in place which enable online monitoring or feedback mechanisms or a combination of these. Both have to address specific technical intricacies: Online monitoring requires modifications of the participating software components to provide the required data along with proofs. User feedback on the other hand must take into account that users provide false feedback, collude to raise or lower QoS properties, or try other attacks to get wrong data into the feedback system. This requires a carefully devised model to filter out wrong data, but requires only minimal changes to the existing infrastructures. In the approach presented in this paper we include a feedback mechanism which is based on user feedback and feedback from (typically a few) trusted third parties, e.g., rating agencies, which exist in many application and business domains. Based on this data and a robust statistical trust and reputation model, which takes into account attacks such as bad-mouthing, collusion, etc., to filter out dishonest behavior, we can provide an accurate picture of the actual QoS.

The suggested framework is general and enables to apply different matchmaking algorithms to the discovery process in the form of discovery algebra operators and parallelization of the semantic discovery for minimizing the total discovery time. More precisely, we perform the categorization of services based on the indexing of their characteristic vectors as originally proposed in (10) and then use this information to filter irrelevant services during the matchmaking steps. Our framework also sketches the solution to deal with the heterogeneous and distributed ontologies using mediation services during the semantic reasoning. The search engine can be run as a centralized service for small-scale environments or can be distributed among any number of cooperating providers. To enable scalable discovery we devise a peer-to-peer-inspired distribution strategy which enables fault-tolerance and allows registries to maintain full control and confidentiality on the information they provide.

This paper is organized as follows: In Section 2 we will first outline briefly the system environment to allow the reader to assess how our system is embedded in a real system. In Section 3 we present our semantic QoS description model followed by a description how discovery works in a centralized setting in Section 4 which we subsequently extend to a distributed environment in Section 5. Section 6 presents the details of the architecture of our QoS-enabled service discovery framework. An analysis of our new solution approach is provided in Section 7, Section 8 discusses it in the context of related research efforts, and Section 9 rounds out the paper with our conclusions.

2 SYSTEM ENVIRONMENT

In the following descriptions we use the conceptual model of the Web Service Modeling Ontology (WSMO) (1). In fact, the use of WSMO in our solution is only a proof-of-concept of the proposed discovery approach, which is generally applicable to other models, such as OWL-S (3). The reason for choosing WSMO is that it has been proposed to the W3C as a standard and is used in our DIP EU project (4). To be able to concentrate on discovery, our search engine exploits several components available in the WSMO Execution environment (WSMX), which is a reference implementation of the WSMO model (9). However, as the dependencies among WSMX components are minimal and the WSMX interfaces are open and published, the few WSMX components we use, e.g., the registry, can easily be replaced by any other component providing the same API.

In the following, we assume that web services are described semantically, including QoS properties, using the WSMO conceptual model. We focus on the discovery component and assume that other functionalities such as publishing of services, managing of service advertisements and ontologies, invocation of services, etc., are handled by existing (WSMX) components, and the the interaction between our discovery component and other components is done via a small set of pre-defined APIs. In the distributed case, these requirements apply to any node hosting part of the overall discovery infrastructure. In reality, a (WSMX) node could be a web service portal where different providers can publish descriptions of their services and where potential customers can submit their search requests.

A query, which is referred to as a *goal* in WSMO, is formulated in a way that it describes the functional and QoS properties of the required services and can be submitted to the system via the interface of any node. For the centralized version of discovery, our search engine compares the user request with the available web service descriptions in its local repository. This semantic matchmaking step chooses only those services satisfying all user functional and QoS requirements, ranks the result list according to the relevancy to the query, and then returns it to the user. In the distributed scenario, the query would also be forwarded to other nodes in the discovery network to retrieve other candidate services elsewhere in the network.

3 QUALITY OF SERVICE MODEL

3.1 Modeling QoS Parameters

Our approach for describing web services builds on the fact that a consumer analyzes and selects a service according to certain criteria, the most common one being the set of objects a service delivers. As an example, consider a ticket reservation service, whose delivered objects are tickets. The criterion specified by the web services describes

the type of tickets it can deliver, i.e., it is part of the functional service description. In contrast, the criterion we are mainly interested in is QoS, i.e., quality-related parameters included in the semantic description of the service advertisements and the user requests. These parameters could vary widely, ranging from the application-level QoS properties to various network or software-related performance features that a user expects from the service execution. Examples of application-level QoS parameters, are the quality of the food delivered by a online-order pizza service or the timely delivery of a book, etc. Typical network or software-related QoS parameters are the availability, response-time, execution-time, etc., of web services.

In this context, selecting a service includes agreeing on the function it delivers and on at least one of QoS parameters (criterion) offered by the provider. Thus, we model a web service description as $WSD = F \wedge Q$, where F is a functional criterion and Q is a QoS criterion. Based on real-world requirements and the most relevant QoS specification standards such as WSLA (6), QML (7), and WSOL (8), we need to provide a semantic description for the following important information:

- The basic knowledge about the QoS parameters, e.g., the parameter names and textual descriptions, the respective measurement units, the possible values of the parameters, the associated evaluation methods, etc.
- The conditions to achieve a certain level of QoS, for instance, the price of the service, the minimum network connection and system requirements, the maximal number of requests per time unit, etc.

A QoS criterion provides consumers with a different type of engagement from a service provider, in which the latter specifies aspects of the real-world which are supposed to hold for the service. The same service may be offered according to different real-world scenarios which entail a different quality engagements. Formally, a QoS criterion is specified as a tuple $t = \langle C, V, O_i \rangle$, where C is a QoS concept defined in O_i, a QoS ontology, and V is a set of concepts specifying a real-world scenario. In addition, for all $v \in V$ there is a relation instance $R \in O_i$, such that $R(C, v)$, modeling the fact that the real-world concept v is associated with the QoS concept C in ontology O_i, composing its real-world scenario. In a web service description a QoS parameter sets the admitted values for its corresponding QoS concept properties and a condition on the associated real-world scenario in which the provider engages to offer the QoS values.

Indeed, a QoS parameter is a tuple $q = \langle C', cnd \rangle$, in which C' is a concept expression that constrains the instances of C, a QoS concept, and cnd is an axiom over elements in $U = \{instances \in V\}$, with V as above. The latter captures the necessary and sufficient conditions in a real-world scenario instance for offering the QoS concept instances specified by C'. For brevity, we sometime refer to cnd as the *context* to achieve the *QoS level* C' henceforth.

Consider, for example, a QoS parameter $qs = \langle C', cnd \rangle$ defined as: $C' = \{o_1, o_2, \ldots, o_n\}$, where C' determines a set of QoS instances I satisfying all constraints $o_1, o_2, \ldots, o_n$, and $cnd = \{p_1, p_2, \ldots, p_m\}$ where $p_k, 1 \leq k \leq m$, is a simple logical formula. The semantics in qs is such that $cnd \Leftrightarrow I$, or cnd establishes the necessary and sufficient condition for achieving a certain QoS instance $qi \in I$.

3.2 Modeling a User Query

Given the previous definitions, we are able to define a service discovery request (or user query or user request or user's goal in our context). Symmetrically to a web service description, a user query should provide functional and QoS criteria a web service should offer to fulfill user needs. In this context, a user query is specified as $G = F \wedge Q$, where F is a functional criterion and Q is a QoS criterion. A QoS criterion in user queries is similar to its counterpart in web service descriptions. Web service consumers indicate the QoS concepts they are interested in and provide information regarding the state of the real-world they are able to agree with. Note that, in other to match a web service description with a user query, the real-world scenario in the former must be subsumed by that of the user query. Thus, if qs_{ws} and qs_g are the QoS parameter instance criteria of the web service description WSD and the user query G , respectively, then we say that WSD matches QoS requirements specified by G iff $qs_{ws} \subseteq qs_g$.

3.3 Feedback on QoS of Web Services

To facilitate the discovery of services with QoS information, we must evaluate how well a service can fulfill a user's quality requirements from the service's past performance. Therefore, the discovery component provides an interface for the service users to submit their feedback on the perceived QoS of the consumed services. The following information should be provided in a user's QoS report:

- the QoS parameter and its observed value;
- the context in which the user obtained this specific value of QoS. More precisely, this would include: the times when the user observed this value and reported it to the discovery component, and, if possible, the detailed information of the user's environmental factors related to the QoS of the transaction between the client and the service provider.

Supposing that the corresponding QoS requirements of this user are $Q = \langle \{C'_1, \ldots, C'_n\}, \{cnd_1, \ldots, cnd_n\} \rangle$, where C'_j and cnd_j are the required constraints and the respective real-world condition of the QoS parameter C_j, $1 \leq j \leq n$. Formally, a corresponding QoS report is represented as a tuple $\langle u, S, t, \{qs_1, \ldots, qs_n\} \rangle$, where u, S, and t denote the identifier of the user, the identifier of the corresponding service, and the current time stamp. Each term $qs_j = \langle qi_j, cnd_j \rangle$, $1 \leq j \leq n$ includes qi_j as the instance of a QoS concept C_j and cnd_j as an axiom which describes the condition over the real-world scenario instance of the user submitting the report. The service is considered as fulfilling all QoS requirements of the user

if $KB \models C'_1(qi_1), \ldots, C'_n(qi_n)$, with KB is the knowledge base of the system constructed by composing ontologies referenced in the service and user query's descriptions.

3.4 Modeling the Semantic Matchmaking

The above definitions lead to a symmetric representation of the web service description and the user query, expressed as $S = F \wedge Q$, where S generalizes both description types. Nevertheless, F and Q have different meanings. The functional description specifies a set of classes that models objects delivered by a web service, whereas the quality criteria is modeled as constraints on instances of quality concepts and conditions on associated real-world concept instances. As a result of this, the semantic matchmaking follows different models when considering functional and quality of service properties.

Matching of functional descriptions as presented here has been discussed in other work (33; 34). A knowledge base KB is constructed by composing ontologies referenced in both descriptions. Given a web service's functional description specified by a concept C_a and its user query counterpart C_b, a match $\mu_F(C_a, C_b)$ occurs iff there exists an instance i, such that $i \in (C_a \sqcap C_b)$, in KB, or $KB \models \mu_F(C_a, C_b)$. Matchmaking of quality criteria is slightly different. Given a pair of QoS criteria, web service description Q_{ws} and user query Q_{uq}, we want similarly to define a match as $\mu_Q(Q_{ws}, Q_{uq})$, where $Q_{ws} = \langle C'_{ws}, cnd_{ws} \rangle$ and $Q_{uq} = \langle C'_{uq}, cnd_{uq} \rangle$. The QoS concept constraint C'_{ws} in Q_{ws} specifies a set of QoS instances, $I_{ws} = \{i_1, i_2, .., i_n\}$ and in addition, cnd_{ws} specifies a space of instances of the real-word scenario. Let us consider $cnd_{ws} = \{p_1, p_2, \ldots, p_m\}$, with $p_k, 1 \leq k \leq m$ being logical formulas constraining instances of V_{ws}, the set of real-world concepts associated with the QoS concept C, such that $C'_{ws} \sqsubseteq C$ and $p_k \sqcap p_l = \emptyset, \forall\ k \neq l$, modeling restrictions on disjoint real-world concepts. Symmetrically, the QoS concept constraint C'_{uq} in Q_{uq} specifies another set of QoS instances, $I_{uq} = \{ri_1, ri_2, .., ri_n\}$. In this context, $\mu(Q_{ws}, Q_{uq})$ represents a match iff, given the above knowledge base KB, such that, (1) $\forall p_j \in cnd_{ws}, \exists p_i \in cnd_{uq}$, such that $p_j \sqsubseteq p_i$; (2) $C'_{uq} \sqsubseteq C'_{ws}$, or equivalently, $I_{uq} \sqsubseteq I_{ws}$; and (3) $KB \models C'_{uq}(i_{ws})$. Note that the subsumption in (1) follows an inverse direction with respect to (2). This reflects the fact that the real-world scenario offered by the user must meet the requirements defined by the provider, whereas with respect to the QoS concept instance the requirements are stated inversely. The subsumption in (3) states that our QoS-enabled service discovery also exploits reputation-based trust management in the semantic matchmaking process, in which i_{ws} is the QoS instance representing the actual QoS capability of a service in the context cnd_{ws}. This actual QoS level is estimated by our reputation-based trust management model based on the set of past QoS instances collected from the feedback of users under the similar environmental conditions, i.e., those QoS reports of the form $\langle u, ws, t, qs \rangle$, where $\exists \langle qi, cnd_j \rangle \in qs$ and $cnd_{ws} \sqsubseteq cnd_j$.

Finally, we can formulate a complete matching process, $\mu(WS, UQ)$, between a web service description $WS = F_{ws} \wedge Q_{ws}$ and a user query $UQ = F_{uq} \wedge Q_{uq}$. Given a knowledge base $KB = O_D \cup O_Q$, where O_D is a set of domain ontologies and O_Q is a set of QoS ontologies, $KB \models \mu(WS, UQ)$ iff $KB \models \mu_F(F_{ws}, F_{uq}) \wedge \mu_Q(Q_{ws}, Q_{uq})$.

3.5 Example of QoS Modeling in WSMO

The usefulness of this representation model in practical applications is illustrated in the following scenario, which we adapt from a real-world case study of our DIP EU project (4). A financial-information service provider offers a web service to retrieve the most important index information of the European stock market. We suppose that this service is accessible via an interface named *subscribedServiceInterface*. The announced level of availability of this interface is around 0.999999, i.e., the probability that the service to be available given all conditions fulfilled is approximately 0.999999 for 100,000 requests. Moreover, the produced information is assured to be as fresh as the actual values in the stock market, i.e. 10 minutes old at most. The service is only accessible in the TechGate building in Vienna, and to use the service via this interface, a valid credit card number of the user is required to pay for using the service. Figure 1 shows how we encode QoS parameters and their associate QoS conditions for this service in the WSMO conceptual model.

Let's now assume a user who is currently at Vienna International Airport would like to obtain statistics on the European stock market which should be not older than 15 minutes. The WSMO representation of the corresponding user query for such a service (formulated with the help of GUI tools) is shown in Figure 2.

In this query the freshness of data is considered as the QoS requirement of the user. For a complete reference, we also include the QoS ontology for this example in Figures 3 and 4.

Due to space limitations and for the sake of clarity, this QoS ontology is quite simple which in reality can easily be extended for more sophisticated requirements with more QoS concepts and utility information such as the QoS evaluation methods, measurement metric units, etc. It is important to note that the information regarding the user's environment that maybe necessary for the service discovery process, i.e., here the user's location and credit card number, is also incorporated into the query (with the help of appropriate GUI tools to simplify query formulation and access). Regarding the above *EUStockMarketMainIndexes()* service description, this query does not match the web service's requirements as the user is not in the appropriate location to be able to use the service.

In another situation, a user working in the TechGate building uses the web interface of a discovery component, e.g., a well-known financial information portal on the Internet, to search for such a service and could obtain the link to the one shown in Figure 1. With the help of the portal, the user could execute this web service and may

```
WebService _"EUStockMarketMainIndexes"
importsOntology _"StockMarketOntology"
capability EUStockMarketMainIndexesCapability
...
Interface subscribedServiceInterface
   importsOntology {_"StockMarketQoSOntology",
                    _"UserInformationOntology"}
   nonFunctionalProperties
      dc#relation hasValue {providedAvailability,
                            availabilityContext,
                            providedDataFreshness,
                            dataFreshnessContext}
   endNonFunctionalProperties
   ...
axiom providedAvailability
 definedBy
     ?serviceAvailability[
           hasMeanValue hasValue 0.999999
           hasBaseNumberOfRequests
                hasValue 100,000
     ] memberOf StockMarketQoSOntology#Availability.

axiom availabilityContext
 definedBy
     ?userData[
           userLocation hasValue loc#Techgate,
     ]memberOf UserInformationOntology#userInformation.

axiom providedDataFreshness
 definedBy
     ?serviceDataFreshness[
        hasMeanValue hasValue ?mean
        hasStandardDeviation hasValue ?std
        hasMeasurementUnit
        hasValue StockMarketQoSOntology#minute
     ] memberOf StockMarketQoSOntology#DataFreshness
     and (?mean <= 10.0).

axiom dataFreshnessContext
 definedBy
     ?userData[
           userLocation hasValue loc#Techgate,
           hasCreditCardNumber hasValue ?credNo
     ]memberOf UserInformationOntology#userInformation
   and StockMarketOntology#validCreditCard(?credNo).
```

Figure 1: Representation of QoS information in a WSMO Web Service description

find out that it is unavailable at the moment. After trying 20 consecutive requests without any success, the portal automatically helps the user to report this problem to the provider via a QoS report. The semantic description part $qs = \{qs_1, \ldots, qs_n\}$ of the observed value of the QoS parameters and the associated information about the user would be represented as shown in Figure 5. Here, the estimated availability value from the user's viewpoint is 0.0, which is averaged on the 20 requests.

4 CENTRALIZED DISCOVERY APPROACH

As already mentioned, our search engine can be run both in a centralized and in a distributed configuration. We first describe the centralized setting and discuss decentralization in the following section. For the centralized discovery solution, a service query of a user (which can also be a program interacting with the discovery component via the API), which includes both functional and QoS properties of the required services, is submitted to the system. The

```
Goal   _"EUStockMarketMainIndexesGoal"
importsOntology _"StockMarketOntology"
capability EUStockMarketMainIndexesRequestedCapability
...
Interface   EUStockMarketMainIndexesRequestedInterface
   importsOntology {_"StockMarketQoSOntology",
                    _"UserInformationOntology"}
   nonFunctionalProperties
     dc#relation hasValue {requestedDataFreshness,
                           dataFreshnessContext,
   endNonFunctionalProperties
   ...

axiom requestedDataFreshness
 definedBy
     ?requiredDataFreshness[
        hasMeanValue hasValue ?mean
        hasStandardDeviation hasValue ?std
        hasMeasurementUnit
        hasValue StockMarketQoSOntology#minute
     ] memberOf StockMarketQoSOntology#DataFreshness
     and (?mean <= 15.0).

axiom dataFreshnessContext
 definedBy
     ?userData[
           userLocation hasValue loc#viennaIntAirport,
     ]memberOf UserInformationOntology#userInformation.
```

Figure 2: Representation of QoS information in a WSMO Goal description

basic local discovery operation is the evaluation of a user's goal against all available web service descriptions in the local repository. This semantic matchmaking chooses only those services satisfying all user requirements, in terms of both service functionality and QoS, ranks the preliminary list according to the relevance to the query, and then returns it to the user. To do this step efficiently, we apply some simple techniques: First, all services which obviously do not satisfy the query are filtered out. Then, the discovery process can be done in parallel by any number of available semantic query engines, which compare the functional and the QoS properties of each service description to the query, and rank them to produce the final result for the user. The matchmaking, selection, and ranking of services based on their QoS, etc., are the operators specified in a discovery algebra, which we will describe in Section 4.3. These operators can have different implementations for different service domains and can be plugged into the system flexibly. The QoS-based service matchmaking and service ranking operators use the actual QoS values as estimated by our reputation-based trust management system, which will be discussed in Section 4.2. This evaluation is based on the services' performance statistics obtained from user feedback reports.

4.1 Semantic Categorization of Web Services

The core of any semantic web service discovery solution is a logic-based matchmaking between a user request and the available web service descriptions, which is a very expensive task. Therefore, it should not be performed on those service descriptions which do not match with the query in any situation. To achieve this, we perform a categoriza-

```
ontology _"StockMarketQoSOntology"
importsOntology _"UserInformationOntology"
   nonFunctionalProperties
     dc#description hasValue "This ontology contains
     the most representative Quality of Service (QoS)
     concepts for Web Services."
   endNonFunctionalProperties

concept _"StockMarketQoS"
   nonFunctionalProperties
      dc#relation hasValue {_"validQoSValue",
                            _"lessthanQoSValues"}
   endNonFunctionalProperties
   hasMeanValue impliesType  (1 1) _double
   hasStandardDeviation impliesType (0 1)_double
   hasMeasurementUnit impliesType (0 1) _"MeasurementUnit"

concept _"ApplicationQoS" subConceptOf _"StockMarketQoS"
concept _"DataFreshness" subConceptOf _"ApplicationQoS"
    nonFunctionalProperties
      dc#relation hasValue {_"validDataFreshnessUnit",
                            _"greaterQoSDataFreshness"}
    endNonFunctionalProperties
    hasMeasurementUnit impliesType (1 1) _"TimeUnit"

concept _"NetworkRelatedQoS" subConceptOf _"StockMarketQoS"
concept _"Availability" subConceptOf _"NetworkRelatedQoS"
    nonFunctionalProperties
      dc#relation hasValue {_"validAvailabilityValue",
                            _"greaterQoSAvailability"}
    endNonFunctionalProperties
    hasBaseNumberOfRequests impliesType (1 1) _integer

concept _"MeasurementUnit" subConceptOf _string
concept _"TimeUnit" subConceptOf {_"MeasurementUnit", _string}

instance _"minute" memberOf _"TimeUnit"
```

Figure 3: QoS ontology for the stock market information example

```
axiom validQoSValue definedBy
    constraint ?X memberOf _"StockMarketQoS"
    and  (exists(?X.hasMeanValue) -> ?X.hasMeanValue > 0.0) .
    and  (exists(?X.hasStandardDeviation)
       -> ?X.hasStandardDeviation > 0.0) .

axiom validAvailabilityValue definedBy
   constraint ?X memberOf _"Availability"
   and  (?X.hasMeanValue<= 1.0)
   and  (exists(?X.hasStandardDeviation)
        -> ?X.hasStandardDeviation < 1.0)
   and  (?X.hasBaseNumberOfRequests > 0.0) .

axiom greaterQoSAvailability definedBy
    ?X >= ?Y <-
    ((?X memberOf _"Availability" and
     ?Y memberOf _"Availability")
    and
    (?X.hasMeanValue >= ?Y.hasMeanValue)
    and
    ((?X.hasStandardDeviation - ?Y.hasStandardDeviation)
    *(?X.hasStandardDeviation - ?Y.hasStandardDeviation)
    <= _double(0.000001))).

axiom validDataFreshnessUnit
   ...
axiom greaterQoSDataFreshness
   ...
axiom lessthanQoSValues
   ...
relation dependencyOfAvailability(ofType Availability,
    ofType UserInformationOntology#userInformation)
relation dependencyOfDataFreshness(ofType Availability,
    ofType UserInformationOntology#userInformation)
```

Figure 4: Associated constraints on the QoS ontology for the stock market information example

```
instance  observedAvailability
  memberOf StockMarketQoSOntology#Availability
  nonFunctionalProperties
     dc#relation hasValue QoSAvailabilityContext
  endNonFunctionalProperties
  hasMeanValue hasValue _double("0.0")
  hasBaseNumberOfRequests
         hasValue _integer("20")

axiom QoSAvailabilityContext
  definedBy
   ?userData[
          userLocation hasValue loc#Techgate,
          accessDay hasValue _gday("Feb 02 2006"),
    ] memberOf UserInformationOntology#userInformation.
```

Figure 5: Example of a QoS Report

tion of service descriptions and user queries into different classes based on the similarity (both in terms of functionality and QoS) among them. This semantic categorization scheme is an extension of our previous work (10). Given this categorization information, the semantic matchmaking is only applied to the user query and the set of service descriptions belonging to the same class. Note that the categorization itself is an expensive process but does not affect the performance of the system since it can be done off-line and incrementally.

For each service description or a user request, we generate a corresponding *characteristics vector* using the available information in its inputs, outputs, precondition, postcondition, and QoS specification (in the WSMO model, this is the service capability and service interface description). All service descriptions and queries are then classified into different categories, depending on their characteristics vectors, as presented in detail in (10). In the first step we partition all concepts in the knowledge-base into different concept groups based on their similarity. Algorithms for determining the semantic similarity between two concepts are widely available, which can be based on the affinity, structural, contextual similarity, etc., between two concepts, for example, (29). The similarity threshold used to decide whether two concepts are in the same group depends on the level of matching the system should provide (the usual trade-off between precision and recall). For example, we could simply set the threshold to 0.0 and use only parental relationships to compute the similarity between concepts. In this case each concept group would be a *partition* of its corresponding ontology. For instance, the *StockMarketOntology* of the Stock Market use case described in the previous sections, the concepts *Stock*, *StockVariations*, *StockMarket*, *Index*, *Dividend*, etc., could be put into one concept group of terminologies related to the Stock Market domain, whereas the other concepts like *creditCard, masterCard, visaCard, eCash, paymentModel* should be put into another concept group.

Given the classification of all ontological concepts in the

local repository, we perform another step of categorization of web services as shown in pseudo-code in Algorithm 1.

Algorithm 1 ClassifyServices()

```
1: for each web service description S do
2:    Create a Bloom key K_S;
3:    for each concept C_i in the capability and QoS description of
      S do
4:       Find the group CG_i of the concept group containing C_i;
5:       Add CG_i to the Bloom key K_S;
6:    end for
7:    Classify the service S based on the Bloom key K_S;
8: end for
```

As a result of this algorithm, all web services are categorized based on the concept groups they operate on. Here we utilize Bloom filters (35) to determine the membership of a certain concept group to a service class efficiently. More precisely, all services S belonging to the same service class SC are described by a Bloom key K_S which is generated using the identifiers of the concept groups that those services operate on. Therefore, the necessary condition for a service S to match with a query G is that its description at least must contains all concepts related to those specified in the query. Analogously, the Bloom filter representation K_S of a service class SC must contain the Bloom filter representation K_G of the query, i.e., $K_S = K_S \bigoplus K_G$, where $\bigoplus$ denotes for bit-wise OR. The pseudo-code for this step is shown in Algorithm 2.

Algorithm 2 FindMatchedServiceClasses(*UserQuery* G): *ListOfServiceClasses* L

```
1: Create a Bloom key K_G;
2: for each concept C_i in the capability and QoS description part
   of G do
3:    Find the group CG_i of the concept group containing C_i, using
      a hash table of concepts;
4:    Add CG_i to the Bloom key K_G;
5: end for
6: Initialize the list of service classes L as empty;
7: for each service class with Bloom key K_S do
8:    if K_S = K_S ⊕ K_G then
9:       Add service class K_S to L;
10:   end if
11: end for
```

For example, in the StockMarket use case, all services whose descriptions do not contain those concepts related to *Stocks*, *StockIndexes*, and do not have any claims for the QoS parameter *DataFreshness*, would be classified as irrelevant for the query and be filtered out.

Note that the implementation for the partial matchmaking is straightforward: Appropriate GUI tools can help the user to specify the set of only the most important concepts in the query from which to generate the key K_G. Since service providers as well as users can use different ontologies for their descriptions, our categorization scheme may use the available ontology mediators to homogenize different ontologies in the service repository if necessary. An ontology mediator is a set of rules to map between two ontologies, which has been already realized in many ontology mapping frameworks. The architecture of our discovery component is designed to be able to interact and exploit the available ontology mediation services in the system and thus addresses ontology heterogeneity appropriately.

Given a user request G and the list of its corresponding matching service classes L, we need to compare G against each web service S in L to choose only those services satisfying all user requirements, which include functionality and QoS requirements. Furthermore, we also need to perform another step of selection and ranking of results based on the actual QoS properties of the web services. This step can be parallelized to reduce the total search time, as presented in Section 4.3.

4.2 Reputation-based QoS Management Model

The discovery of services with QoS information requires an accurate evaluation of how well a service can fulfill a user's quality requirements from the service's past performance. For this estimation, we use a reputation-based model which exploits data from many information sources: (1) We use the QoS values promised by providers in their service advertisements. (2) We provide interface for the service users to submit their feedback on the perceived QoS of consumed services. (3) We also use similar reports produced by a few trustworthy QoS monitoring agents, e.g., rating agencies, to observe the QoS statistics of a number of web services. (4) In a distributed setting, the different discovery components in the network may periodically exchange the reputation information and performance statistics of the service users, services, providers and of the discovery nodes themselves.

For each QoS parameter C_j (a concept in a QoS ontology) provided by a service S, we evaluate the real capability of this service in providing this QoS parameter to the users as follows. With every context cnd_{jk}, i.e., a set of real-world conditions as in Section 3.1, in which the service S advertises C_j, the related QoS instances q_{jkt} are collected. Specifically, we gather all user reports which are on S and refer to C_j and of the corresponding context cnd_{jk}. Such a report by a user u at time t has the form $\langle u, S, t, qs\rangle$ where $\exists\langle q_{jkt}, ucnd_{jk}\rangle \in qs$ and $cnd_{jk} \sqsubseteq ucnd_{jk}$. The reputation-based estimation of the actual capability of S in providing the QoS parameter C_j in context cnd_{jk} is an instance $\widehat{q}_{jk}$ of C_j computed as follows: Since each QoS instance q_{jkt} consists of a list of property-value pairs $\langle p_l, v_{lt}\rangle$, each property p_l of the QoS instance $\widehat{q}_{jk}$ would then have the value $\widehat{v}_l$. The estimation of a $\widehat{v}_l$ based on its historical statistics $\langle t, v_{lt}\rangle$ is then done using the time-based regression methods which we proposed in (11).

So as to improve the accuracy of this QoS evaluation, the feedback mechanism firstly has to ensure that false ratings of the malicious autonomous agents, for example, badmouthing about a competitor's service or pushing the own rating level by fake reports or collusion with other malicious parties, can be detected and dealt with. Secondly, it is also necessary to create incentives for users to submit feedback on their consumed services and produce such reports truthfully.

For dealing with the first problem, we have developed a reputation-based trust management model under two realistic assumptions. First, we assume *probabilistic behavior of services and users*. This implies that the differences between the real quality conformance which users obtained and the QoS values they report follow certain probability distributions. These differences vary depending on whether users are honest or cheating as well as on the level of changes in their behaviors. Secondly, we presume that *there exist a few trusted third parties*. These well-known trusted agents always produce credible QoS reports and are used as trustworthy information sources to evaluate the behaviors of the other users. In reality, companies managing the discovery components at each peer can deploy special applications themselves to obtain their own experience on QoS of some specific web services. Alternatively, they can also hire third party companies to do these QoS monitoring tasks for them. In contrast to other models (24; 25; 26; 27; 28) we do not deploy these agents to collect performance data of all available services in the registry. Instead, we only use a small number of them to monitor QoS of some selected services because such special agents are usually costly to set up and maintain. In order to detect possible frauds in user feedbacks, we use reports of trusted agents as reference values to evaluate behaviors of other users by applying a trust-distrust propagation method and a clustering algorithm. Reports that are considered as incredible will not be used in the QoS evaluation process. Details of this step are presented in our previous paper (11). This proposed dishonest detection algorithm has shown to yield very good results under various cheating behaviors of users although we only deploy the trusted third parties to monitor QoS of a relatively small fraction of services.

The solution for the second problem is studied extensively by economists using game-theory as their major tool. In our environment, so as to give incentives for users to participate and produce reports truthfully, side-payment schemes with appropriate scoring rules could be well applicable (12). That is, the users who give feedback truthfully on their consumed web services should get certain benefits, from either the discovery component or from the service providers, e.g., reduced subscription prices when using or searching for web services.

4.3 An Execution Model for QoS-enabled Semantic Web Service Discovery

One may envisage a single discovery component managing a large number of web service descriptions and being targeted by numerous user queries with completely unpredictable arrival rates. In this context, the performance of a discovery process becomes of primordial importance as well as its ability to respond to variations on incoming query arrival rates, while keeping the discovery time of each query at an acceptable level.

In order to provide such guarantees, we model the discovery process as a cost-based adaptive parallel query processing system (13). Within the discovery process we distinguish independent operators with a clear semantic and to which one may associate an estimated evaluation cost. A discovery query is modeled as an operator tree, in which nodes represent discovery operators and edges denote the dataflow between each pair of them. Potentially, a single discovery query may be modeled by a number of different operator trees, albeit equivalent in terms of the results they produce. In this context, a query optimizer is a component which is responsible for deriving an operator tree producing the smallest estimated cost for a given discovery query and the load information as well as the throughput statistics of the different nodes in the system, based on a cost model.

We have identified a set of discovery operators that together form a discovery algebra. Each operator represents a particular function within the discovery process and may be implemented using different algorithms:

- *restriction* σ - reduces the set of web services descriptions candidates for matching with the user query. Our current implementation adopts the Bloom filtering strategy described in Section 4.1. Web services descriptions whose Bloom key do not match those of the user query are filtered-out.
- *match* μ - applies a semantic matchmaking algorithm to assess the similarity between a web service description and an user query. Matchmaking is implemented as described in Section 3.
- *rank* ρ - orders matched web service descriptions based on the results of the match operation and according to user's preferences, as we will discuss later in this section.
- *project* π - delivers results to the user.

In addition to ordering operators into an operator tree, our execution model extends traditional query execution by supporting program invocation and introducing some dynamic optimization techniques. In fact, the semantic based subsumption reasoning matchmaking can take long to evaluate, as a function of the complexity and of the size of the QoS ontology. Moreover, each web service description may entail as much as $T = F_c + (\sum_{i=1}^{n} \sum_{j=1}^{m} 1 + Q_c)$ invocations of a semantic matchmaking function, where F_c is the number of concepts in the functional definition, n is the number of service interfaces, m is the number of QoS parameters in each interface, and Q_c is the number of disjoint concepts in a QoS parameter condition. As a result, an efficient evaluation of a discovery query must target three main issues: (1) reduce the number of reasoning tasks; (2) reduce the elapsed time for each individual web service description semantic matchmaking evaluation; (3) adapt to variations on execution environment conditions. We cope with these three issues by introducing control operators into the operator tree that manage data transference, data materialization and reasoning task parallelization:

- *cache* χ - stores results of semantic matchmaking evaluations, avoiding useless re-computation. χ manages a local hashtable $H = \langle conceptpair, result \rangle$. For

each evaluation of $R = \mu(C_1, C_2)$, we check for the existence of $R = H\langle C_1, C_2\rangle$, otherwise we compute $R = \mu(C_1, C_2)$ and update the hashtable.

- *split/merge* γ - supports parallel evaluation of a reasoning task by distributing/merging individual tasks/results to another node.
- *cycle* $\circ$ - dynamically schedules reasoning tasks to other semantic query engines based on current node and network throughput measurements. The $\circ$ operator is instantiated into the machine running the execution manager component.
- *apply* α - encapsulates the invocation of functions. In this paper it is being used to model the invocation of a reasoning task (reasoner).

For brevity reasons, we refer the interested reader to our previous work (13) where the parallelization and adaptive execution strategy is described in detail.

The ranking operator ρ is realized as follows. Let us consider a user query G with QoS requirements $Q_G = \{\langle C'_1(q_1), cnd_1\rangle, \ldots, \langle C'_n(q_n), cnd_n\rangle\}$, where q_k, $1 \leq k \leq n$, is a QoS concept in a QoS ontology, C'_k is the required QoS level of q_k, and cnd_k is the associated context to achieve C'_k, respectively. Suppose that the list of services that match the above query are $L_G = \{S_1, S_2, \ldots, S_m\}$. As described in Section 3.4, the QoS semantic matchmaking μ_Q requires that for every S_i, all QoS concepts q_k that are specified in Q_G must appear in the description of S_i. Moreover, the corresponding context for S_i to achieve the QoS level C'_k would be $cnd_{ik} \sqsubseteq cnd_k$. Let $\widehat{q_{ik}}$ be the estimation of the actual QoS capability of S_i in providing the QoS concept q_k under the context cnd_{ik}. This is already computed based on the historical performance statistics of S_i, as described in Section 4.2. The ranking operator ρ is a sort of the list $L_G = \{S_1, S_2, \ldots, S_m\}$ in descending rank order, using the following condition: We denote $S_i \succeq S_j$, or S_i has the higher rank than S_j, in terms of their QoS and with respect to the user query G, iff: $\Phi(i,j) = \sum_{k=1}^{n} w_k.1_{\{\widehat{q_{ik}} \geq \widehat{q_{jk}}\}} > \xi_G$, where $0 \leq w_k \leq 1$ is the level of importance of the quality q_k to the user, $\sum_{k=1}^{n} w_k = 1$, and ξ_G is a threshold which is chosen according to w_k's. $\widehat{q_{ik}}$ and $\widehat{q_{jk}}$ are estimated instances of q_k provided by S_i and S_j with the meaning as explained above. The w_k's and ξ_G are collected from user's preferences in the systems. In other words, we will give higher ranks to services which actually offer the more important QoS concepts at the higher levels to the user. The semantic of the comparison $\widehat{q_{ik}} \geq \widehat{q_{jk}}$ is defined as the ordering relation of two QoS instances in the corresponding QoS ontology. For example, in the *StockMarketQoSOntology* of Figure 4, this comparison between two instances of the *Availability* concepts is reflected by the axiom *greaterQoSAvailability*.

5 DECENTRALIZED DISCOVERY APPROACH

In scenarios with independent providers of service directories a centralized solution is infeasible. Here, a distributed strategy for searching in a network of providers is required which still provides the same functionalities and guarantees as the centralized version. To achieve good scalability, high flexibility, and self-maintenance and additionally avoid unwanted dependencies among the providers, we apply a P2P-based approach for distributed discovery. However, in reality, any distributed infrastructure could be used to realize our solution.

5.1 Semantic Query Routing

To route queries in a distributed setting, we first need to identify all relevant locations and forward the query to these nodes. To ensure the accuracy of the discovery approach, we should be able to identify all relevant discovery locations. Moreover, the distribution of queries should be efficient in terms of message cost and bandwidth consumption in the network. To achieve these requirements, we propose the use of a structured overlay network on top of the network of peer nodes based on the indexing of the data stored at each peer (Figure 6).

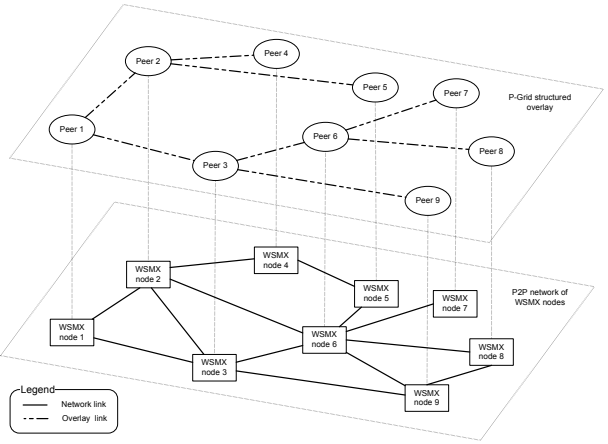

Figure 6: The use of a structured overlay on a network of discovery nodes

The searching for data is done via the overlay routing algorithm to locate the responsible peers holding the requested information. The key idea of our solution to the identification of relevant discovery locations is the indexing of the ontologies used in the service descriptions of the service repositories. This indexing scheme is beneficial for several reasons: (1) A semantic service query using certain ontologies in its description should be forwarded only to those peers with related ontologies, since these peers are likely to store matching services. (2) This solution preserves the privacy of nodes, as it uses only the basic and not the detailed information about the services in a provider's repository, which are generally unavailable for indexing in the network. (3) This basic information are less likely to change than the detailed descriptions of the services. The update cost in a large-scale network, if necessary, would be less costly.

Our semantic query routing also takes into account the trust and reputation of the discovery locations in the sense that a node would not forward queries to those with which it had bad experiences. The evaluation of a peer's reputation is based on the QoS of the services it has provided and will be discussed in more detail in the next section. At each peer receiving a query, another local service discovery process would take place in parallel. After collecting all partial results, which are already ranked by the respective peers, we perform another final ranking before returning the results to the user. In this step, we simply assign higher ranks for services provided by the peers with higher reputation and for those services with higher QoS levels.

5.2 Decentralized Reputation-based Trust Management

In a distributed environment, a provider can have its own service registry for its web services and hence may have incentives to give feedback biased to these services. Therefore, we also have to estimate the trustworthiness of the various partial search results returned from the nodes. Our proposal to deal with this problem is as follows. After the user selects and executes a web service S which is provided by a node q, the user may report a QoS assessment on this service execution through a QoS report R. The report submission mechanism should ensure that R is sent to peers $p_i \in L_P$, i.e., peers involved in the semantic query routing algorithm described above. This enables the fair sharing of QoS experience among the peers who are likely to work in the same domain, since they are destinations of the semantic query routing, and to avoid the case where some p_i and q collude to deter negative feedback. Available cryptography techniques could be applied to ensure that the feedback is authentic, for example, QoS reports could be required to be digitally signed by both the service provider and the service user. Digital signatures also ensure that nodes can only read but not tamper the contents of the QoS reports. The reputation of a peer q is updated periodically by each node $p_i \in L_P$. For doing this, each p_i first needs to apply certain dishonest detection techniques, such as the trust-distrust propagation algorithm we proposed in (11), to filter out possibly cheated reports. Then, p_i evaluates the general reputation value of the peer q according to the set of reports it has about the QoS of the services provided by q. Moreover, the p_i can exchange information about the reputation of q among each other and compute an aggregated reputation value. A node q whose reputation (in the view of p_i) falls below a certain threshold rt_i is put on a blacklist L_D^i of nodes that p_i distrusts. That means that p_i will not forward service queries to q anymore.

The reputation value of a node q should reflect its capability of providing good services in the past, which is expressed in corresponding reports of the users on the QoS of these services. A peer p_i assigns the default reputation for a newly joined peer q as the minimal reputation value in the list of its accepted peers, i.e., $r_q^i = \max\{rt_i, \min_{p_j \in \overline{L_D^i}}\{r_j\}\}$. Such a default value of reputation would create chances for new peers to join the system and cause the least harm to the reputation of the judging peer p_i. Otherwise, p_i computes the reputation of peer q as $r_q^i = \frac{\sum_{\forall R \in R_q} e^{-\lambda t} 1_{\{Q_d > Q_a\}}}{\sum_{\forall R \in R_q} e^{-\lambda t}}$, where R_q is the set of QoS reports in the data repository of p_i that related to the services recommended by q, Q_d and Q_a denote the QoS levels delivered by this service to the user and the respective level advertised by the provider at time t. λ is a constant to reflect the decay of the reports over time. The indicator function $1_{\{Q_d > Q_a\}}$ evaluates to 1 if the user considers all QoS requirements to be fulfilled by the service and 0 otherwise. Its computation is straightforward by comparing values in the report with those in the service advertisements.

6 ARCHITECTURE

Figure 7 shows the high-level architecture of our search engine. Each participating node hosts the same basic functionalities and cooperates with other sites over the network, i.e., the individual discovery nodes form a *discovery overlay network*.

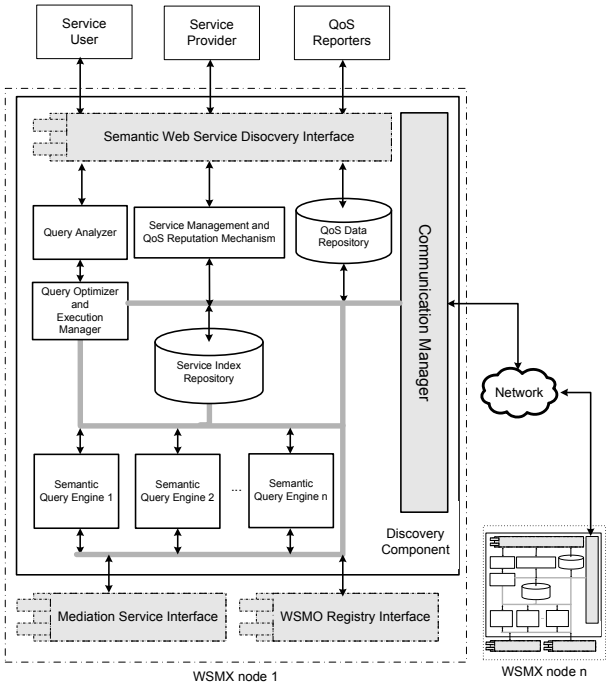

Figure 7: The search engine architecture

The architecture of each individual node consists of the following components:

Semantic Web Service Discovery Interface: This is the central access point for all available functionalities (searching, querying reputation information, providing feedback, system management, etc.)

Query Analyzer: Upon receiving a user request, e.g., a WSMO Goal, this component extracts the relevant information, i.e., the required functional and QoS specifications, and generates an internal representation for the query optimizer.

Query Optimizer and Execution Manager: This component uses the output of the query analyzer and produces an execution plan for the semantic matchmaking, using the algebraic operators defined in Section 4.3. The key idea is that we use the query optimizer to parallelize the semantic matchmaking (and other operators) over different semantic query engines. The execution manager part of the component is responsible for controlling query execution and collecting the final results.

Semantic Query Engine: These components in parallel run parts of the query execution plan assigned to them by the query optimizer / execution manager. All query engines have access to the service index repository, the QoS data repository, and the service registry of the local node via standardized interfaces to get the required data for their execution. They can also use the mediation service to deal with translating among different QoS ontologies during the matchmaking processes.

Service Index Repository: This repository stores the indexes of the web service descriptions obtained from the service registry. During the QoS-enabled service discovery process, information in this database helps the semantic query engines to filter out service descriptions irrelevant to the query by comparing the category index of a service and a user's query. This frees us from doing expensive reasoning on those services.

Service Management and QoS Reputation Mechanism: This component performs the semantic categorization of the semantic web services in the service index repository. It is also responsible for evaluating the QoS of different services, given the data collected from various QoS reporters.

QoS Data Repository: This repository stores the feedback of QoS reporters, the reputation information of all service providers as well as the evaluated QoS information of all QoS-aware web services, i.e., those services with QoS information in their descriptions. The QoS data repository acts as an additional data source for the semantic query engines during query resolution.

Communication Manager: This component enables communication among the nodes of the search engine, e.g., query routing to and result collection from remote nodes.

While these components already provide the core functionalities required to perform the discovery task, it depends on two external components, an optional mediation service and a mandatory service repository. We have defined interfaces which allow us to plug-in arbitrary components here and will briefly discuss the functionalities we expect from these components in the following.

Mediation Service: As we cannot expect all descriptions and queries to use the same vocabularies, this component should provide data mediation between ontological vocabularies. Our solution for ontology heterogeneity relies on three assumptions: (1) User queries and web services most likely use different vocabularies; (2) both users and service providers use controlled vocabularies or ontologies to describe requested and provided services; and (3) there is some data mediation service in place. In our opinion we can optimistically assume that a mapping has already been established between the used terminologies, not only to facilitate our specific discovery problem but also to support the general information exchange process among different partners. In the realization of our approach in the WSMO model, the available mediation services provided by other components of the WSMX framework are incorporated into our design via a standardized interface.

Web Service Repository: As a number of web service repositories have already been provided, we did not want to include yet another one into our system but provide a generic interface to access existing ones. We assume that the repository provides us with functionalities to manage the storage and retrieval of web service descriptions, ontologies and associated mediators. In our framework we use the reference implementation of a WSMO Service Registry, which is accessible via a standardized interface defined by the *WSMO4J* group (2). This basically provides developers with much flexibility in choosing the language to be used in their repository implementation, although limiting inter-operability with client applications.

7 DISCUSSION

7.1 Discovery Cost Analysis

We anticipate that the main cost of the discovery process is local service discovery: (1) A discovery component is able to pre-compute and keep destinations for distributed queries to accelerate the search for relevant discovery locations. (2) The distributed discovery is done in parallel at these locations and the main cost would then be that of the slowest local discovery process. Therefore, in this section we focus on the effectiveness of the centralized discovery.

We adopted a parallel cost model to represent the cost of evaluating a discovery query. It considers the costs of individual operators as well as a global function that models the complete query execution. The global execution model supports two parallel modes: intra-operator and inter-operator. The former instantiates multiple copies of the same operator into different execution nodes, whereas the latter provides a pipelined execution of operators in a branch of the operator tree. In addition, we consider an initialization cost I_c that encompasses the cost of instantiating fragments of the operator tree into different nodes, as well as the cost of transferring initial data to remote nodes. Once the initialization is finished the execution begins by reading input web service descriptions (or tuples). We observe an increase of cost as the first tuple enters the pipeline, until it has achieved a blocking operator. This is called a warming-up cost and is followed by a full pipelined execution with a constant time (if no delay is observed) until the last tuple leaves the system (cooling-down period).

Finally, the ranking operator requires the whole input set to be consumed before it can deliver an ordered result.

For the initialization cost we have the estimation $I_c = t_B + t_{Opt} + t_{CI}$ where t_B is the cost of computing the Bloom key for the user query, t_{Opt} is the cost of the scheduling algorithm for finding the set of most suitable nodes, and t_{CI} is the time for initializing the appropriate nodes for the corresponding operators which is negligible in a centralized environment with well-known operators.

Once the query execution plan has been produced and the system has been initiated, the time of checking whether the first web service description S_1 matches with the user query G is $T_1 = t_{CL} + t_\mu 1_{S_1 \sqcap G}$, where t_{CL} is the time of checking whether the service class of S matches with that of the query, t_μ is the time for doing the semantic matchmaking $\mu_F \wedge \mu_Q$ of the query with the service, if their classes matched with each other, i.e., $1_{S_1 \sqcap G} = 1$.

As our semantic query engines work as a parallel pipeline processing system, and the most expensive stage is the matchmaking μ which are scheduled to run in parallel in different nodes, the time of doing the semantic matching of N_S web service descriptions against the query is $\Psi = I_c + T_1 + t_\mu(\frac{kN_S}{K_S N_e} - 1) + t_R = t_B + t_{Opt} + t_{CL} + t_\mu 1_{S_1 \sqcap G} + t_\mu(\frac{kN_S}{K_S N_e} - 1) + t_R$, where N_e is the number of parallel nodes to run the μ operator according to the scheduling algorithm, K_S is the number of service classes and k is the number of service classes matched with the query, and t_R is the cost of the ranking operator.

Suppose each ontology have an average of N_p partitions each of which has N_c concepts. Furthermore, let the average number of concepts in a web service description and in a user query be l_s and l_g, correspondingly. For the estimation of the execution time of different algorithms, in the followings we also use the notions τ_F to denote the approximate time for doing one operation of the corresponding algorithm F. With the use of a hash table, the finding of the respective concept groups for each concept in the service query could be done with constant time $O(1)$. Hence we have $t_B = l_g.\tau_b$. Since the checking whether the query matches with a service class is a bitwise OR, $t_{CL} = 1.\tau_c$. The time t_μ is of a logic-based reasoning operation, which usually has exponential complexity in terms of the number of concepts in the ontology and of the service and query description ((32), Ch. 9), so we could envisage $t_\mu \simeq e^{N_p N_c l_g l_s}.\tau_\mu$ in the worst case. The cost t_{Opt} of the scheduling algorithm in our scenario is $O(PlogP)$, where P is the number of all available nodes in the system (13). Given a limited number of nodes in our local environment, t_{Opt} is also negligible with respect to the actual execution time of the whole discovery process. The time t_R of the ranking algorithm, given the length of the result list l_r, depends mostly on the sorting of services based on their relevancies to user query. Thus $t_R = l_r l_g log(l_r l_g).\tau_r$.

Therefore, we have $\Psi \simeq l_g.\tau_b + t_{Opt} + \tau_c + e^{N_p N_c l_g l_s}.\tau_\mu.1_{S_1 \sqcap G} + e^{N_p N_c l_g l_s}.\tau_\mu(\frac{kN_S}{K_S N_e} - 1) + l_r l_g log(l_r l_g).\tau_r$. This could be further reduced as $\Psi \simeq e^{N_p N_c l_g l_s}.\tau_\mu \frac{kN_S}{K_S N_e} \simeq t_\mu \frac{kN_S}{K_S N_e}$, since this is the most significant cost in compared with the other terms such as $l_g.\tau_b, \tau_c, e^{N_p N_c l_g l_s}.\tau_\mu.1_{S_1 \sqcap G}$ and $l_r l_g log(l_r l_g).\tau_r$.

On the other hand, the time cost of the discovery process without any enhancement techniques is $\Omega = N_S.t_\mu$. The estimated speed-up of our solution is therefore $S_p = \frac{\Omega}{\Psi} \simeq \frac{N_S.t_\mu}{N_S.t_\mu.\frac{k}{K_S N_e}} = \frac{K_S N_e}{k}$.

As the concept definitions are actually the core of the functional and QoS criteria in a user query, with the semantic categorization of services based on the groups of concepts they operate on, we can have a high selectivity while choosing the matched service classes for a certain request. That is, K_S is expected to be high and k is of much smaller magnitude. Thus the discovery time for one query can be can significantly reduced. Moreover, in an environment where there is an unexpectedly high number of queries, our solution also takes into account the load and throughput statistics of each node in the system and optimizes the execution via the node-scheduling algorithm. Therefore, our total gain of efficiency is extremely relevant in terms of not only the reduction in the response time of each individual query but also the optimization of the overall performance of the system as well.

7.2 Effectiveness of the Reputation-based Trust Management Model

We have implemented the QoS-based selection and ranking algorithm with the application of our reputation-based trust management model and then studied its effectiveness under various settings. We observed the dependency between the quality of selection and ranking results and other factors, such as the percentage of trusted users and reports, the rate of cheating users in the user society and the various behaviors of users. Details of these experiments are discussed in our previous paper (11). Through empirical experiments, our QoS-based service selection and ranking algorithm yields very accurate and reliable results even in extremely hostile environments (up to 80% cheating users with varying cheating behaviors), which is due to facts that the use of trusted third parties monitoring a relatively small fraction of services (from 3% to 5%) can greatly improve the detection of dishonest behavior even in extremely hostile environments. The experiments suggest to deploy trusted agents to monitor the QoS of the most important and most widely-used services in order to get a *"high impact"* effect when estimating behaviors of users. Additionally, we can pre-select the important services to monitor, keep the identities of those specially chosen services secret and change them periodically to make the model even more robust. Thus, cheaters will not know on which services they should report honestly in order to become high-reputation recommenders and have to pay a very high cost to have great influences in the system. In reality, this also helps us to reduce the cost of setting up and maintaining trusted agents as we only need to deploy them to monitor changing sets of services at certain time periods.

8 RELATED WORK

The traditional UDDI standard (5) does not refer to QoS for web services, but many proposals have been devised to extend the original model and describe web services' quality capabilities, e.g., WSLA (6), QML (7), and WSOL (8). The UX architecture (19) suggests using dedicated servers to collect the feedback of consumers and then predict future performance of published services. (20) proposes an extended implementation of the UDDI standard to store QoS data submitted by either service providers or consumers and suggests a query language (SWSQL) to manipulate, publish, rate and select those QoS data from the repository. According to Kalepu et.al. (21), the reputation of a service should be computed as a function of three factors: different ratings made by users, service quality compliance and its verity, i.e., the changes of service quality conformance over time. However, these solutions have not yet addressed the trustworthiness of QoS reports produced by various users, which is important to assure the accuracy of the QoS-based selection and ranking results. (22) rates services in terms of their quality performance with QoS information provided by monitoring services and users. The authors also employ a simple approach of reputation management by identifying every requester to avoid report flooding. In (18), services are allowed to vote for quality and trustworthiness of each other and the service discovery engine utilizes the concept of distinct sum count in sketch theory to compute the QoS reputation for every service. However, these reputation management techniques are still simple and not robust against various cheating behaviors, e.g., collusion among liars with varying actions over time. Consequently, the quality of ranking results of those discovery systems will not be assured if there are lots of colluding dishonest users trying to boost the quality of their own services and badmouthing about others. (23) suggests augmenting service clients with QoS monitoring, analysis and selection capabilities, which is a bit unrealistic as each service consumer would have to take the heavy processing role of both a registry and a reputation system. Other solutions (24; 25; 26; 27; 28) use mainly third-party service brokers or specialized monitoring agents to collect performance of all available services in registries, which would be expensive in reality. Though (28) also raises the issue of accountability of Web Service Agent Proxies, the evaluation of trust and reputation for these agents is still an open problem.

Decentralized solutions for service discovery are also an important issue recently. METEOR-S (14) and HyperCup (15) base the distribution of semantic Web Service descriptions on a classification system expressed in service or registry ontologies. Though it is relatively simple when publishing and updating service description information based on their categories, it would be difficult for users to search for certain services without knowing details of this classification, let alone come up with such a common service or registry ontology. WSPDS (16) uses an *unstructured* P2P network as the service repository, which would be not very highly scalable and efficient in terms of searching and updating costs. (17) indexes service description files (WSDL files) by a set of keywords and uses a Hilbert-space-filling-curve to map the n-dimensional service representation space to an one-dimensional indexing space and hash it onto the underlying DHT-based storage system. However, the issue of characterizing a semantic web service description in a multi-keyword form in order to support semantic discovery of services has not yet been mentioned in this work. Regarding the semantic characterization of web services several properties can be considered, of which the most obvious are the structural properties of the service interface, i.e., the input and output parameters of a service. Other characteristics of web services, in particular the process structure of the service invocation also have been considered, e.g., (18), but we consider these as less important, since they are difficult to use in queries and unlikely to be the primary selection condition in searches, and thus not critical in terms of indexing.

9 CONCLUSIONS

In this paper we have proposed a semantic description model for the QoS of web services, whose expressivity facilitates modeling a wide range of QoS requirements. Our framework includes a solution for dynamic assessment and management of QoS values of web services based on user feedback and performs QoS-enabled discovery and ranking of web services based on their QoS compliance. The ranking is based on a reputation-based trust management mechanism to evaluate the actual QoS of the services, which increases the robustness and accuracy of our solution. We have introduced a scalable and efficient QoS-enabled semantic web service discovery framework, which could be used both as a centralized discovery component or as a decentralized registry system consisting of cooperating discovery nodes, i.e., they form a discovery overlay network. Additionally, our discovery architecture is based on a query processing architecture in which operations are modeled algebraically, enabling query optimization and parallelization of operator execution. The decentralized discovery approach in our framework prescribes a realistic solution for large-scale distributed discovery of web services and addresses the issue of heterogeneous and distributed ontologies.

ACKNOWLEDGMENTS

The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and was supported by the Swiss National Funding Agency OFES as part of the European project DIP (Data, Information, and Process Integration with Semantic Web Services) No 507483. Le-Hung Vu is supported by a scholarship of the Swiss federal government for foreign students.

REFERENCES

[1] The Web Service Modeling Ontology web site, http://www.wmso.org/.

[2] The Web Service Modeling Ontology for Java project's web site, http://wsmo4j.sourceforge.net/.

[3] Ontology Web Language (OWL) for Services, http://www.w3.org/Submission/OWL-S/.

[4] The DIP Integrated Project- Data, Information, and Process Integration with Semantic Web Services, http://dip.semanticweb.org/.

[5] UDDI Spec Technical Committee Draft (2004), http://uddi.org/pubs/uddi-v3.0.2-20041019.htm/, Dated 20041019.

[6] Ludwig, H., Keller, A., Dan, A., King, R.-P., Franck, R., (2003) 'Web Service Level Agreement (WSLA) Language Specification', available online at http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

[7] Frolund, S., Koisten, J. (1998) 'QML: A Language for Quality of Service Specification', http://www.hpl.hp.com/techreports/98/HPL-98-10.html.

[8] Tosic, V. (2004) 'Service Offerings for XML Web Services and Their Management Applications', *Ph.D. dissertation, Department of Systems and Computer Engineering, Carleton University*, Canada, 2004.

[9] Burstein, M., Bussler, C., Zaremba, M., Finin, T., Huhns, M., Paolucci, M., Sheth, A., Williams, S. (2005) 'A Semantic Web Services Architecture', *IEEE Internet Computing. Vol. 9, No. 5*, September, October 2005.

[10] Vu, L.-H., Hauswirth, M., Aberer, K. (2005) 'Towards P2P-based Semantic Web Service Discovery with QoS Support', *Proceeding of Workshop on Business Processes and Services (BPS)*, Nancy, France, 2005.

[11] Vu, L.-H., Hauswirth, M., Aberer, K. (2005) 'QoS-based service selection and ranking with trust and reputation management', *Proceedings of OTM'05, R. Meersman and Z. Tari (Eds.)*, LNCS 3760, p.p. 466-483, 2005.

[12] Miller, N., Resnick, P., and Zeckhauser, R. (2005) 'Eliciting Informative Feedback: The Peer-Prediction Method', *Forthcoming in Management Science*, 2005.

[13] Porto, F., Silva, V. F. V., Dutra, M. L., Bruno, S. (2005) 'An adaptive distributed query processing grid service', *Proceedings of the Workshop on Data Management in Grids, VLDB2005)*, Trondheim, Norway, 2005.

[14] Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller., J. (2005) 'METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services', *Inf. Tech. and Management, 6(1):17–39, 2005.*

[15] Schlosser, M., Sintek, M., Decker, S., Nejdl, W. (2002) 'A scalable and ontology-based P2P infrastructure for semantic web services', *Proceedings of P2P'02*, page 104, Washington, DC, USA, 2002.

[16] Kashani, F. B., Chen, C. C., Shahabi, C. (2004) 'WSPDS: Web services peer-to-peer discovery service', *Proceedings of the International Conference on Internet Computing*, p.p. 733–743, 2004.

[17] Schmidt, C., Parashar, M. (2004) 'A peer-to-peer approach to web service discovery', *Proceedings of the WWW'04, 7(2):211–229, 2004.*

[18] Emekci, F., Sahin, O. D., Agrawal, D., Abbadi, A. E. (2004) 'A peer-to-peer framework for web service discovery with ranking', *Proceedings of the ICWS'04, page 192*, USA, 2004.

[19] Chen, Z., Liang-Tien, C., Silverajan, B., Bu-Sung, L. (2003) 'UX - an architecture providing QoS-aware and federated support for UDDI', *Proceedings of ICWS'03*, 2003.

[20] Bilgin, A. S., Singh, M. P. (2004) 'A DAML-based repository for QoS-aware semantic web service selection', *Proceedings of the ICWS'04, page 368*, USA, 2004.

[21] Kalepu, S., Krishnaswamy, S., Loke, S. W. (2004) 'Reputation = f(user ranking, compliance, verity)', *Proceedings of ICWS'04, page 200*, USA, 2004.

[22] Liu, Y., Ngu, A., Zheng, L. (2004) 'QoS computation and policing in dynamic web service selection', *Proceedings of the WWW conference on Alternate track papers & posters, p.p. 66–73*, USA, 2004.

[23] Day, J., Deters, R. (2004) 'Selecting the best web service', *Proceedings of the IBM Centers for Advanced Study Conference (CASCON'04), p.p. 293–308*, 2004.

[24] Tian, M., Gramm, A., Naumowicz, T., Ritter, H., Schiller, J. (2003) 'A concept for QoS integration in web services', *Proceedings of Fourth International Conference on Web Information Systems Engineering Workshops, Vol. 00, p.p. 149–155*, Italy, 2003.

[25] Ran, S. (2003) 'A model for web services discovery with QoS', *SIGecom Exch., 4(1):1–10*, 2003.

[26] Ouzzani, M., Bouguettaya, A. (2004) 'Efficient access to web services', *IEEE Internet Computing, p.p. 34–44*, March/April 2004.

[27] Patel, C., Supekar, K., Lee, Y. (2003) 'A QoS oriented framework for adaptive management of web service based workflows', *Proceeding of DEXA'03, p.p. 826–835*, 2003.

[28] Maximilien, E. M., Singh, M. P. (2002) 'Reputation and endorsement for web services', *SIGecom Exch., 3(1):24–31*, 2002.

[29] Castano, S., Ferrara, A., Montanelli, S., and Zucchelli, D. (2003) 'HELIOS: a general framework for ontology-based knowledge sharing and evolution in P2P systems', *Proceedings of DEXA'03*, USA, 2003.

[30] Tang, C., Xu, Z., and Dwarkadas, S. (2003) 'Peer-to-peer information retrieval using self-organizing semantic overlay networks', *Proceedings of ACM SIGCOMM'03*, USA, 2003.

[31] Ding, H., Solvberg, I. T., and Lin, Y. (2004) 'A vision on semantic retrieval in P2P network', *Proceedings of AINA'04*, USA, 2004.

[32] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (2003) 'The Description Logic handbook', *Cambridge University Press*, 2003.

[33] Keller, U., Lara, R., Lausen, H., Polleres, A. and Fensel, D. (2005) 'Automatic location of services', *Proceedings of ESWC'05'*, 2005.

[34] Li, L., Horrocks, I. (2003) 'A software framework for matchmaking based on Semantic Web technology', *Proc. of the Twelth Intl. WWW'03*, Hungary, 2003.

[35] Bloom, B. H. (1970) 'Space/Time trade-offs in hash coding with allowable errors', *Communication of the ACM'*, 13(7), p.p. 422–426, 1970.