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 such contexts, especially in a business environment, quality of service (QoS) and non-functional properties are usually the most important criteria to decide whether a specific resource will be used. To address this problem, we present an approach to the semantic description and discovery of Web services which specifically takes into account their QoS properties. Our solution uses a robust trust and reputation model to provide an accurate picture of the actual QoS to user. The search engine is based on an algebraic discovery model and uses adaptive query-processing techniques to parallelize expensive operators. Architecturally, the engine can be run as a centralized service for small-scale environments or can be distributed among any number of cooperating registry providers.

Keywords: Semantic Web service; service discovery; QoS; trust; reputation

Reference to this paper should be made as follows: Vu, L.-H., Hauswirth, M., Porto, F., and Aberer, K. (200x) '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. x, No. x, pp.xxx–xxx.

Biographical notes: *Le-Hung Vu* is a PhD student at the Distributed Information System Laboratory of the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland. His research interests include Semantic Web service discovery, QoS for Web services, and trust and reputation management in decentralized environments.

Manfred Hauswirth is a senior researcher at the Distributed Information Systems Laboratory, EPFL, Switzerland. His main research interests are large-scale distributed systems, peer-to-peer systems, sensor networks, self-organizing systems, Web services, e-commerce systems, and publish/subscribe systems.

Fabio Porto is a senior researcher at the Database Laboratory of EPFL, Switzerland. His research domain includes distributed query processing systems, query optimization, reasoning systems and the Semantic Web.

Karl Aberer is a full professor and head of the Distributed Information Systems Laboratory, EPFL, Switzerland. His main research interests are distributed information management, peer-to-peer computing, Semantic Web and self-organizing information systems.

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 are supposed to be delegated to the computer. This not only requires expressive means to describe Web services semantically which is a major ongoing research effort, for example, WSMO (http://www.wmso.org), but also puts more complex re-

quirements onto discovery.

Most of the ongoing efforts address functional aspects. However, in a business environment, usually the non-functional and QoS properties, such as price, performance, throughput, reliability, availability, trust, etc., are actually the first ones to be applied in deciding whether a specific resource will be used and only if they are 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 can 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 and QoS 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 facilitates the application of 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 Vu et al (2005a) 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.

The Web Service Modeling Ontology (WSMO) model is used as a proof-of-concept of our proposed discovery approach. However, the approach is generally applicable to other models, e.g., OWL-S (http://www.w3.org/submission/owl-s). 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 (Burstein et al (2005)). Without constraining general applicability, we assume in the following 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, monitoring of service QoS, etc., are handled by existing (WSMX) components, and that the interaction between our discovery component and other components is done via pre-defined APIs.

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

2 QUALITY OF SERVICE MODEL

2.1 Modeling QoS Parameters

In our perspective, a Web service is a web-based interface providing an electronic description of a concrete service, which may offer the functionality of a software component, for example, a data backup service, or a real-life (business) activity such as book shipping. Therefore, the notion of QoS in our work is generic, ranging from application-level QoS properties to 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. 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 the QoS parameters (criterion) offered by the provider. Thus, we model a Web service description as $F \wedge Q$, where F is a functional criterion and Q is a QoS criterion. Based on real-world requirements and the most relevant QoS speci-

fication standards such as WSLA (Ludwig et al (2003)), QML (Frolund and Koisten (1998)), and WSOL (Tosic (2004)), we need to provide a semantic description for the following important information of a QoS criterion:

- The description of the QoS parameters and the associated required level of QoS, e.g., parameter names and textual descriptions, possible values of the parameters, respective measurement units, associated evaluation methods, etc. We model this part as $C'(qi)$, in which C' is a concept expression that constrains the instance qi of a QoS concept in the QoS domain ontology.
- The necessary and sufficient conditions that the provider engages to offer the QoS values for offering the QoS instances specified by the above QoS level $C'(qi)$, for instance, the price of the service, the minimum network connection and system requirements, the maximal number of requests per time unit, etc. This part is expressed by *cnd*, an axiom over instances of a set of concepts specifying a real-world scenario.

Thus, a QoS criterion is modeled as set of tuples, each of which has the form $\langle C'(qi), cnd\rangle$. For brevity, we denote *cnd* as the *context* to achieve the *QoS level* $C'(qi)$ henceforth. As an example, the constraint C' could be used to describe the average response-time value of a Web service while *cnd* could be used to describe the contextual conditions, e.g., network connection speed, that a client must have in order to get the specified average response-time.

Symmetrically to a Web service description, a service discovery request (also called a user query, a service query, or a user's goal in the descriptions to follow) consists of the description of the functional and QoS criteria a Web service should offer to fulfill a user's needs. In this context, a user query is specified as $F \wedge Q$, 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 by $C'(qi)$ the QoS level they are seeking for and provide information *cnd* regarding the state of the real-world they are able to agree with when searching for a service with that specified QoS level.

2.2 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 time 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 user and the service provider.

Supposing that the corresponding QoS criterion of the user are $Q = \{\langle C'_1(qi_1), cnd_1\rangle, \ldots, \langle C'_n(qi_n), cnd_n\rangle\}$, where $C'_j(qi_j)$ and cnd_j are respectively the required instance constraints and the respective real-world condition of the ontological QoS concept q_j, $1 \leq j \leq n$, a corresponding QoS report is represented formally 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 report timestamp. Each term $qs_j = \langle qi_j, ucnd_j\rangle$, $1 \leq j \leq n$ includes qi_j as the instance of a QoS concept q_j and $ucnd_j$ as an axiom over the set of ontological instances describing the real-world scenario 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 being the knowledge base of the system constructed by composing ontologies referenced in the service and user query's descriptions.

2.3 Modeling the QoS Semantic Matchmaking

The above definitions lead to a symmetric representation of the Web service description and the user query, expressed as $F \wedge Q$. Nevertheless, F and Q have different meanings. 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 Keller et al (2005) and Li and Horrocks (2003). 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 μ_Q is slightly different. Given a pair of QoS criteria in a Web service description Q_{ws} and in a user query Q_{uq}, where, for example, $Q_{ws} = \langle C'_{ws}(qi_{ws}), cnd_{ws}\rangle$ and $Q_{uq} = \langle C'_{uq}(qi_{uq}), cnd_{uq}\rangle$, $\mu_Q(Q_{ws}, Q_{uq})$ represents a match iff, given the above knowledge base KB: (1) $cnd_{uq} \sqsubseteq cnd_{ws}$; (2) $C'_{ws}(qi_{ws}) \sqsubseteq C'_{uq}(qi_{uq})$; and (3) $KB \models C'_{uq}(\widehat{qi_{ws}})$. Here $\widehat{qi_{ws}}$ is the QoS instance representing the actual QoS capability of a service in the context cnd_{ws}, which is estimated by our reputation-based trust management model based on the set of QoS instances collected from the feedback of users under similar environmental conditions, i.e., those QoS reports of the form $\langle u, ws, t, qs\rangle$, where $\exists\langle qi, ucnd_j\rangle \in qs$ s.t. $ucnd_j \sqsubseteq cnd_{ws}$. Note that the subsumption in (1) follows an inverse direction with respect to (2). This reflects the fact that the real-world scenario (or client-side conditions) 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 is also based on the historical performance values of the Web services.

2.4 WSMO QoS Modeling Example

The usefulness of this representation model in practical applications is illustrated in the following scenario, which we adapted from a real-world case study of our DIP project (http://dip.semanticweb.org). 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 produced information is assured to be as fresh as the actual values in the stock market, i.e., at most 10 minutes old. The service is only accessible in the TechGate building in Vienna city, Austria, 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 associated QoS conditions for this service in the WSMO conceptual model, using the WSML-Rule language syntax for ontological modeling. The axiom *providedDataFreshness* is the realization of the notation QoS constraint $C'(qi)$ on instances of the QoS concept *DataFreshness*. Similarly, the axiom *dataFreshnessContext* corresponds to the context *cnd* that the Web service provider engages to offer the values claimed in the above axiom *providedDataFreshness*.

```
WebService EUStockMarketMainIndexes
importsOntology {StockMarketOntology}
capability EUStockMarketMainIndexesCapability
...
Interface subscribedServiceInterface
   importsOntology {StockMarketQoSOntology,
                    UserInformationOntology}
   nonFunctionalProperties
      dc#relation hasValue {providedDataFreshness,
                            dataFreshnessContext}
   endNonFunctionalProperties
   ...
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 in a WSMO Web Service description (a more complete description and related ontologies for this example are provided at http://lsirpeople.epfl.ch/lhvu/ontologies/)

Let us now assume a user who is currently at Vienna International Airport and would like to obtain statistics on the European stock market which should not be 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 QoS required level $C'(qi)$ of the user and his environment *cnd* are respectively expressed by the axioms *requestedDataFreshness* and *userDataFreshnessContext*. It is important to note that the description of the client environment, i.e., user's location and credit card number, which could be necessary for the service discovery process, is also incorporated into the query. This is done by using appropriate GUI tools to query the QoS domain ontology and derive the related environmental concepts for the parameter *DataFreshness*. 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.

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

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

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 service described in Figure 1. During the testing of this service (before integrating it into the her company's data analysis workflow engine), she finds out that the provided information values are not as fresh as the current statistics in the market (via other information sources). With the help of the portal, she can submit a complaint on this problem to the service provider via a QoS report. An example of the semantic description being part of such a report is shown in Figure 3.

3 CENTRALIZED DISCOVERY APPROACH

Our search engine can be run both in a centralized and in a distributed configuration. For the centralized discovery

```
instance  observedDataFreshness
  memberOf StockMarketQoSOntology#DataFreshness
  nonFunctionalProperties
     dc#relation hasValue qosDataFreshnessContext
  endNonFunctionalProperties
  hasMeanValue hasValue 20.0

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

Figure 3: Example of a QoS Report

solution, a service query of a user (or a program interacting with the discovery component via the API), which includes both functional and QoS properties of the required service, is submitted to the system. The basic discovery operation is the evaluation of a user's goal against all available Web service descriptions in the central (in terms of distribution, 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 processors, 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 QoS-based service matchmaking and service ranking operators also use the actual QoS values as estimated by our reputation-based trust management system, which will be discussed in Section 3.2.

3.1 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 be performed only on those service descriptions which potentially match the query. To achieve this, we perform a categorization 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 (Vu et al (2005a)). 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 and 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, these are the service capability and service interface description). All service descriptions and queries are then classified into different categories, depending on their characteristics vectors. 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, Castano et al (2001). 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, in the *StockMarketOntology* of the Stock Market use case described in the previous section, 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 (Bloom (1970)) 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 contain 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 bit-wise OR. The pseudo-code for this step is shown in Algorithm 2.

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.

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
```

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 available ontology mediators to homogenize different ontologies in the service repository if necessary. An ontology mediator is a set of rules mapping between two ontologies, a functionality, which has been realized already 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 3.3.

3.2 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 an 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 2.1, 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 in 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 $\exists\langle q_{jkt}, ucnd_{jkt}\rangle \in qs$ and $ucnd_{jkt} \sqsubseteq cnd_{jk}$. The reputation-based estimation of the actual quality 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 Vu et al (2005b).

In order to improve the accuracy of this QoS evaluation, the feedback mechanism 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. Additionally, 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 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. This proposed dishonesty detection algorithm has been 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 (Vu et al (2005b)).

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 (Miller et al (2005)). 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.

3.3 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 (Porto et al (2005)). Within the discovery process we distinguish independent operators with clear semantics and to which one may associate estimated evaluation costs. 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. Thus we derive an operator tree producing the smallest estimated cost for a given discovery query.

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 service description candidates for matching with the user query. Our current implementation uses Bloom filtering as described in Section 3.1. Web service descriptions whose Bloom keys 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 a user query. Matchmaking is implemented as described in Section 2.3.
- *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.
- *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 reasoning and introducing some dynamic optimization techniques. The reasoning task is invoked as part of the *match* operator and deserves special attention as it can become a bottleneck for the execution. Thus 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 in execution environment conditions. We cope with these three issues by introducing control operators into the operator tree that manage data transfer, data materialization, reasoning task parallelization and scheduling, etc. For brevity reasons, we refer the interested reader to our previous work (Porto et al (2005)) describing the parallelization and adaptive execution strategies in detail.

The ranking operator ρ is realized as follows: Let us consider a user query G with QoS requirements $Q_G = \{\langle C'_1(qi_1), cnd_1\rangle, \ldots, \langle C'_n(qi_n), cnd_n\rangle\}$, where $C'_k(qi_k)$, $1 \leq k \leq n$, represents the required QoS level of a QoS concept q_k in a QoS ontology, and cnd_k is the user's associated context to achieve C'_k, respectively. Suppose that the list of services that match the above query is $L_G = \{S_1, S_2, \ldots, S_m\}$. As described in Section 2.3, the QoS semantic matchmaking μ_Q requires that for every S_i, all QoS concepts q_k's 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}, where $cnd_k \sqsubseteq cnd_{ik}$.

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 3.2. We denote $\rho(S_i) \succeq \rho(S_j)$, or S_i has the partial higher rank than S_j, in terms of QoS and with respect to the user query G, iff $\Phi(i,j) = \sum_{k=1}^{n} w_k . 1_{\{\widehat{q_{ik}} \geq \widehat{q_{jk}}\}} \geq \xi_G$, 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 the 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. The global rank of a service depends on the number of higher ranks it gets when compared to all other services. Thus, the global rank score of the service S_i is $T_i = \sum_{S_j \in L_G, j \neq i} 1_{\{\rho(S_i) \succeq \rho(S_j)\}}$. The ranking operator ρ is a sort of the list L_G in descending order of T_i. 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 domain ontology.

4 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.

4.1 Semantic Query Routing

To route queries in a distributed setting, we first need to identify all relevant discovery locations and forward the query to these nodes. 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 to use a structured overlay network on top of the network of peer discovery engines based on the indexing of the data stored at each peer (Figure 4).

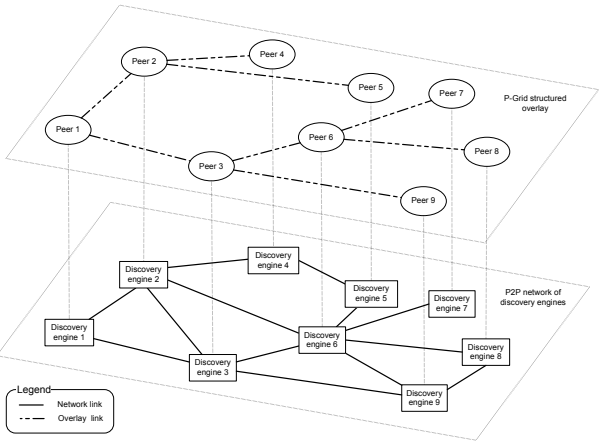

Figure 4: Structured overlay on top of discovery nodes

Search 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.

4.2 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 its own services. Therefore, we also have to estimate the trustworthiness of the various partial search results returned from the different discovery nodes. Our proposal to deal with this problem is as follows: After the invocation of a Web service S which is provided by a node q, the user may report a QoS assessment on this service through a QoS report R. The report submission mechanism should ensure that R is sent to all nodes (or peers) p_i in the list L_P of peers involved in the semantic query routing algorithm. 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 route. This helps 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 digitally signed by both the service provider and the service user. Digital signatures also ensure that nodes can only read but not tamper with 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 dishonesty detection techniques, such as the trust-distrust propagation algorithm we proposed in Vu et al (2005b), to filter out possibly cheating 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 discovery node 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 discovery nodes 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 $r_q^i = \frac{\sum_{\forall R \in R_q} e^{-\lambda t} 1_{\{Q_d \geq Q_a\}}}{\sum_{\forall R \in R_q} e^{-\lambda t}}$, 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 $1_{\{Q_d \geq Q_a\}}$ evaluates to 1 if the user considers all QoS requirements to be fulfilled by the service and 0 otherwise. Its computation is straight-

forward by comparing values in the reports with those in the service advertisements.

5 SOFTWARE ARCHITECTURE

Figure 5 shows the high-level software 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*.

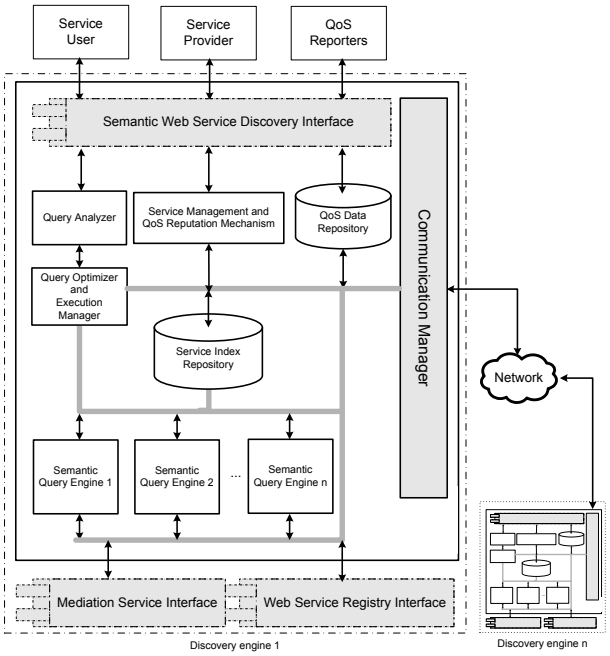

Figure 5: The search engine architecture

The architecture consists of the following components hosted by each node:

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 3.3. 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, they depends on two following external components: an optional mediation service and a mandatory service repository.

Mediation Service Interface: This is the access point to the mediation services possibly provided by other components, e.g., the mediation service of the WSMX framework. Mediation is incorporated into our design via a standardized interface.

Web Service Registry Interface: This interface provides access to the web service registry to be used, which provides us with functionalities to manage the storage and retrieval of web service descriptions, ontologies and associated mediators.

6 DISCUSSION OF OUR APPROACH

6.1 Discovery Cost Analysis

We anticipate that the main cost of the discovery process is local service discovery for two reasons: (1) A discovery component is able to pre-compute and maintain 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 analysis of the efficiency 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 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. 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 $T_1 \simeq t_{CL} + t_\mu 1_{S_1 \sqcap G}$, 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 is scheduled to run in parallel on N_e different nodes, the time of doing the semantic matching of N_S web service descriptions against the query is $\Psi = I_c + T_1 + t_\mu(\frac{kN_S}{K_S N_e} - 1) + t_R \simeq t_B + t_{Opt} + t_{CL} + t_\mu 1_{S_1 \sqcap G} + t_\mu(\frac{kN_S}{K_S N_e} - 1) + t_R$, where N_e is computed according to the scheduling algorithm, K_S is the number of service classes, k is the number of service classes matched with the query, and t_R is the cost of the ranking operator.

Suppose each ontology has 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 following 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, finding the respective concept groups for each concept in the service query can 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 (Baader et al (2003), 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 $O(P \log P)$, where P is the number of all available nodes in the system (Porto et al (2005)). 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 pair-wise comparison of services. Thus $t_R \approx (l_r l_g)^2 \tau_r$.

Therefore, we have $\Psi \simeq l_g \tau_b + t_{Opt} + \tau_c + e^{N_p N_c l_g l_s} \tau_\mu 1_{S_1 \sqcap G} + e^{N_p N_c l_g l_s} \tau_\mu (\frac{kN_S}{K_S N_e} - 1) + (l_r l_g)^2 \tau_r$. 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 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 $(l_r l_g)^2 \tau_r$. 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 unexpected 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.

6.2 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 Vu et al (2005b). 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 the services (3%–5%) can significantly 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 ser-

vices 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 if they want to achieve great influence 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.

7 RELATED WORK

The traditional UDDI standard (http://uddi.org/pubs/uddi-v3.0.2-20041019.htm) does not mention QoS for Web services, but many proposals have been devised to extend the original model and describe Web services' quality capabilities, e.g., Ludwig et al (2003), Frolund and Koisten (1998), and Tosic (2004). Chen et al (2003) suggest using dedicated servers to collect the feedback of consumers and then predict future performance of published services. Bilgin and Singh (2004) propose an extended implementation of the UDDI standard to store QoS data submitted by either service providers or consumers and suggest a query language (SWSQL) to manipulate, publish, rate and select those QoS data from the repository. According to Kalepu et al (2004), 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. Liu et al (2004) rate services in terms of their quality 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 Emekci et al (2004), 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. Other solutions such as Tian et al (2003); Ran (2003); Ouzzani and Bouguettaya (2004); Patel et al (2003); Maximilien and Singh (2002) 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.

Regarding decentralized solutions for service discovery, Verma et al (2005) and Schlosser et al (2002) propose a distribution of Semantic Web service descriptions based on a classification system expressed in service or registry ontologies. Kashani et al (2004) uses an *unstructured* P2P network as the service repository, which would be not very highly scalable and efficient in terms of search and update costs. Schmidt and Parashar (2004) 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 a one-dimensional indexing space and hash it onto the underlying DHT-based storage system. Emekci et al (2004) suggests the discovery of services based on the process structure of the services, but we consider these as less important in our case, since they are difficult to use in queries and unlikely to be the primary selection criteria in searches, and thus not critical in terms of indexing.

8 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 can be used both as a centralized discovery component or as a decentralized registry system consisting of cooperating discovery nodes (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

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

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

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.

Burstein, M. et al (2005) 'A Semantic Web Services Architecture', *IEEE Internet Computing. Vol. 9, No. 5*, September, October 2005.

Castano, S., De Antonellis, V., and di Vimercati, S. D. C. (2001)'Global Viewing of Heterogeneous Data Sources', *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 2, pp. 277-297, Mar/Apr, 2001.

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.

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

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.

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

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

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.

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

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

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.

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.

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

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

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

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.

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.

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

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.

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

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.

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.

Tosic, V. (2005) 'Service Offerings for XML Web Services and Their Management Applications', *Ph.D. dissertation, Department of Systems and Computer Engineering, Carleton University*, Canada, 2004.

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.*

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

Vu, L.-H., Hauswirth, M., Aberer, K. (2005b) '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.