An Approach to Integrated Semantic Service Discovery

Shanshan Jiang and Finn Arve Aagesen

Department of Telematics
Norwegian University of Science and Technology (NTNU)
N-7491 Trondheim, Norway
{ssjiang, finnarve}@item.ntnu.no

Abstract. In a distributed service environment, service discovery is a core functionality to locate the desired services. We propose an integrated semantic service discovery approach based on ontology, which provides matching of functional and non-functional properties. Functional properties are described in terms of operations, inputs, outputs, preconditions and effects, while non-functional properties are specified as business policies, QoS properties and context policies. Ontological inference and rule-based reasoning are applied for automatic and accurate discovery.

1 Introduction

In a distributed service environment, service discovery is a core functionality to locate the desired services. *Service discovery* is a process of finding the desired service(s) by matching *service descriptions* against *service requests*. A *service description* provides service-related information which can be advertised by a service provider and searched during service discovery process. Such information usually includes functional properties and non-functional properties. In this paper, functional properties representing functionality of a service are modeled in terms of operations, inputs, outputs, preconditions and effects, while non-functional properties comprise business policies, Quality of Service (QoS) properties as well as context policies. QoS properties include QoS parameters and QoS policies. A *service request* represents user's service requirements, comprising requirements on functional and non-functional properties.

Ontologies are the basis for adding semantic expressiveness to service descriptions and requirements. An *ontology* is an explicit and formal specification of a shared conceptualization [19]. A service ontology is accordingly an explicit and formal specification of core concepts of the functional and non-functional properties of service. "A *domain ontology* (or *domain-specific ontology*) models a specific domain and represents the particular meanings of terms as they apply to that domain. An *upper ontology* is a model of the common objects that are generally applicable across a wide range of domain ontologies." [1] Ontological relations such as "is-subclass-of" or "part-of" are used for ontological inference.

[1] Wikipedia: `http://en.wikipedia.org/wiki/Ontology_(computer_science)`

Semantic service discovery is a service discovery process based on ontology concepts. By using ontology concepts defined in a service ontology expressively in a service description, semantics of the service description can be defined. These service descriptions are therefore expressive semantic descriptions. At the same time, by having both ontology-based descriptions and requirements, an ontology-enhanced reasoning engine (i.e. capable of ontological inference) can be used to locate services automatically and accurately. *Integrated semantic service discovery* is a semantic service discovery process based on both functional and non-functional properties of the services.

Service discovery has been a hot topic in the last years, and many different approaches have been proposed. In Web Services technology, Web Services are described in WSDL (Web Services Description Language) [22] and advertised in UDDI (Universal Description, Discovery and Integration) [21] registries. UDDI provides only keyword-based discovery (e.g. service category or provider name) and makes no use of semantic information of service behaviour (e.g. semantics of operations, inputs and outputs) defined in the service descriptions during discovery. A number of protocols for service discovery have also been proposed, most notably, SLP (Service Location Protocol) [8], Jini [11], UPnP [7] and Salutation [4]. Service descriptions in these protocols are usually based on categories of predefined service types, interface types, attributed IDs and values, without expressive semantic descriptions to enable reasoning. Thus service discovery is restricted to simple keyword-based category and attribute matching. Other approaches based on Semantic Web technology, such as [13], provide semantic service discovery, but limit their discovery to functional properties only, without considering non-functional properties during the process. Integrated semantic service discovery, however, is important to achieve accurate and satisfactory discovery results.

In this paper, we propose an integrated semantic service discovery approach based on semantic-annotated WSDL [16]. Ontologies are defined in the Web Ontology Language, OWL [12]. Behavioral semantics are added to WSDL file by associating service functionality related elements with links to OWL-based service ontology. Non-functional properties are specified as QoS parameters and rule-based policies comprising business policies, QoS policies and context policies. Furthermore, WS-Policy Framework (Web Services Policy Framework) [6] and WS-PolicyAttachment (Web Services Policy Attachment) [5] are utilized to attach QoS parameters and policies to WSDL-based service descriptions. Service requirements are also expressed using ontology concepts. Based on them, an integrated semantic service discovery procedure is presented, which takes into account selection criteria based on business policies, QoS policies and context policies, as well as user defined service selection criteria in terms of overall QoS scores based on QoS parameters.

The rest of the paper is organized as follows: Sect. 2 discusses service description elements with focus on rule-based policy specifications. Based on it, an integrated semantic service discovery framework is presented in Sect. 3. Related work is discussed in Sect. 4, followed by summary and conclusions in Sect. 5.

2 Service Description Elements

A service description comprises the following elements:

– *Functionality description* in terms of operations, inputs, outputs, preconditions and effects.
– *QoS parameters* offered by the service.
– *Service parameters* such as location and other service specific parameters.
– *Policies* for service discovery, comprising business policies, QoS policies and context policies.

A policy is a rule applied in the decision making process. It is usually conditional criteria against which factual variables are evaluated to determine an appropriate action. Therefore, a rule-based policy representation is a natural choice for policy specification. Ontology-based policy description allows service providers and requestors to describe their policies with respect to a common ontology in terms of meaningful concepts and relations. Furthermore, benefit from semantic enrichment and ontological inference can be achieved. For service discovery process, policies will guide the optimal selection of desired services among several functionally equivalent services.

Formally, an ontology-based policy rule P can be defined as a tuple <r, o, s, c, a>, where r is a reference to a policy ontology, o denotes the organization P belongs to, s denotes the service P applied to, c the conditions, a the actions. WS-Policy framework [6] provides a general purpose model to describe and communicate policies of a Web Service. It places no restrictions on the language used to represent policy expressions. We use an ontology language, OWL, to express the policy based on the upper ontology for policy inspired by [18] and depicted in Fig. 1. The upper ontology defines the concepts used for policy specification and their relations. A *policy* belongs to an *organization* and is applied to a *service.* A *policy* has a *policy domain* and refers to a domain-specific *policy ontology.* A *policy* may have multiple *rule sets*, each of them defines a set of *rules.* The *rule sets* have *rule operators*, such as "ExactlyOne" to specify that only one rule set is applied at a time. Each *rule* is a conditions-and-actions statement, which specifies the *actions* to be performed when *conditions* are evaluated to TRUE. *Conditions* are specified in *expressions* while *actions* are associated with *operations* of a *service. Actions* may also have *expressions* and may require *conditions. Expressions* may have *attributes, literal values, operators* as well as *logical operators. Attributes* can be *service parameters, inputs, outputs* or *QoS parameters* of a service. Therefore, this upper ontology of policy has relations with service ontology, i.e. concepts in the gray area in Fig. 1 also belong to service ontology.

2.1 Business Policies

Business policies are rules related to business concepts. They can be published associated with a service to constrain service discovery process. As an example, consider a delivery policy for an online bookstore, say *BookStoreA*, which specifies that if the number of copies ordered for a book is less than 50, then the

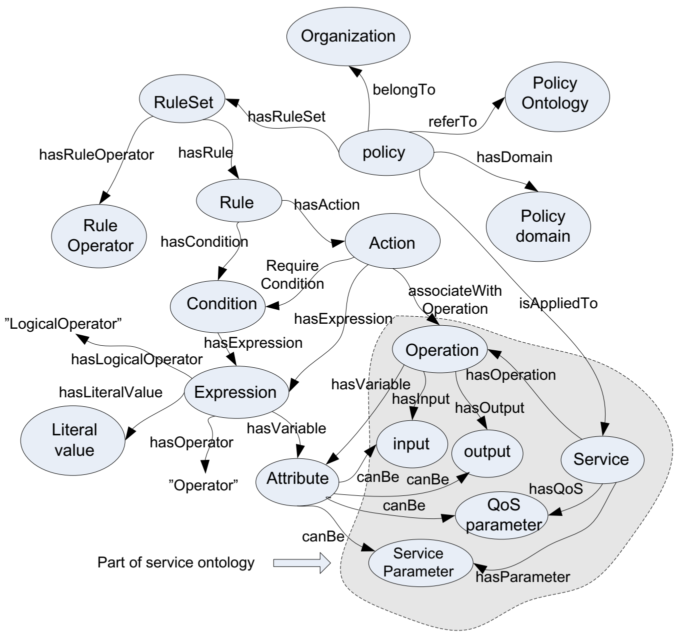

Fig. 1. Upper ontology for rule-based policy

delivery time will be within 5 days; if between 50 and 200, the order will be delivered within 10 days; otherwise, the inventory should be checked before an action is taken. The policy can be expressed in three rules as shown in Fig. 2.

Rule 1 IF ($numberOfCopies \leq 50$)
THEN DeliverBooks ($deliveryDays \leq 5$)
Rule 2 IF ($50 < numberOfCopies \leq 200$)
THEN DeliverBooks ($deliveryDays \leq 10$)
Rule 3 IF ($numberOfCopies > 200$)
THEN CheckInventoryFirst

Fig. 2. Example delivery policy for an online bookstore

A user who orders 100 copies of a textbook to be delivered within 15 days from online bookstores may select the *PurchaseBook* service from *BookStoreA* since his/her request can be matched by the second rule. Figure 3 shows part of the policy specification called *DeliveryPolicyBookStoreA* for an online bookstore *BookStoreA* based on the upper ontology defined in Fig. 1, which corresponds to **Rule 1** in Fig. 2. Note that the namespace *po:* refers to the upper ontology, while the namespace *sp:* refers to the domain-specific policy ontology. Attribute *numberOfCopies* is a service input, attribute *deliveryDays* is a service output, while operation *DeliverBookOperation* is a service operation.

```
xmlns:po="http://examplepolicy.com/policy.owl#"
xmlns:sp="http://ecommerce.com/policy.owl#"
xmlns="http://BookStoreA.com/PolicyRule.owl#"

<sp:DeliveryPolicy rdf:ID="DeliveryPolicyBookStoreA">
  <po:hasRuleSet rdf:ID="RuleSet1">
    <po:hasRuleOperator rdf:resource="po:ExactlyOne" />
    <po:hasRule>
      <po:Rule rdf:ID="Rule1">
        <po:hasCondition rdf:resource="#CheckQuantity1"/>
        <po:hasAction rdf:resource="#DeliverBooks1"/>
      </Rule>
    </po:hasRule>
    ....
  </po:hasRuleSet>
</sp:DeliveryPolicy>

<sp:CheckQuantity rdf:ID="CheckQuantity1">
  <po:hasExpression>
    <po:Expression rdf:ID="ExprCondition1">
      <po:hasVariable>
        <po:Attribute rdf:resource="sp:numberOfCopies"/>
      </po:hasVariable>
      <po:hasOperator rdf:resource="po:isLessThanOrEqual"/>
      <po:hasLiteralValue>
        <po:LiteralValue rdf:ID="LiteralValue1">
          <po:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int">50
          </po:hasValue>
          <po:hasType rdf:resource="po:Integer" />
        </po:LiteralValue>
      </po:hasLiteralValue>
    </po:Expression>
  </po:hasExpression>
</sp:CheckQuantity>

<sp:DeliverBooks rdf:ID="DeliverBooks1">
  <po:associateWithOperation>
    <sp:DeliverBookOperation rdf:ID="DelBkStoreA" />
  </po:associateWithOperation>
  <po:hasExpression>
    <po:Expression rdf:ID="ExprAction1">
      <po:hasVariable>
        <po:Attribute rdf:resource="sp:deliveryDays" />
      </po:hasVariable>
      <po:hasOperator rdf:resource="po:isLessThanOrEqual" />
      <po:hasLiteralValue>
        <po:LiteralValue rdf:ID="LiteralValue2">
          <po:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int">5
          </po:hasValue>
          <po:hasType rdf:resource="po:Integer" />
        </po:LiteralValue>
      </po:hasLiteralValue>
    </po:Expression>
  </po:hasExpression>
  <po:requireCondition rdf:resource="#CheckQuantity1"/>
</sp:DeliverBooks>
```

Fig. 3. Example policy rule specification in OWL

2.2 QoS Properties

QoS is a very important aspect of non-functional properties for a service. In a distributed environment, services with equivalent functionality can be provided by different service providers with substantially varied QoS. How to specify QoS and incorporate it into service discovery process is thus of great importance.

QoS properties can be specified as *QoS parameters* and *QoS policies*. *QoS parameters* are QoS attributes that can be expressed in quantifiable measurements or metrics. A service usually possesses a set of QoS parameters. Though many of them are of dynamic nature, i.e., related to the execution of the service, a service can still advertise its guaranteed QoS in service description. *QoS policies* are rules related to QoS parameters. Rule-based QoS policies have often been used in network-related services, e.g. network and system management services. A service can provide different QoS classes of service depending on the service classes users subscribed. For example, a *GoldClass* user may have access to *GoldClassService*, which guarantees a set of QoS parameters, such as bandwidth and response time, much better than a user in *SilverClass*. A QoS policy for service discovery can similarly be specified as "If a user belongs to *GoldClass*, then the service provided guarantees a set of QoS parameters."

QoS parameters can be classified into different categories, e.g., scalability, capacity, performance, reliability, availability, etc. There are several efforts to define and categorize QoS parameters in terms of classifications or ontologies [10] [15]. Fig. 4 shows part of a QoS ontology based on [10] for illustration purpose.

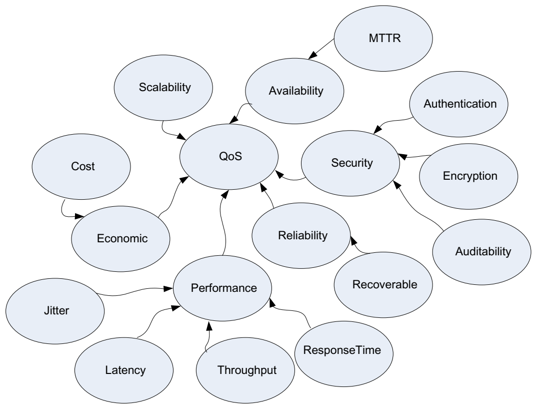

Fig. 4. Part of a QoS ontology (arrows indicate subClassOf relationship)

Not all attributes in the QoS ontology is relevant in a specific service discovery process, for example, a user may consider some of the QoS parameters valuable in his or her request. The matching procedure for service discovery therefore needs to take this into account by calculating the user specified QoS selection criteria. We adopt the QoS ranking approach proposed in [14], which defines a *quality matrix* to represent the values of user specified QoS parameters for all candidate services and an *overall QoS score function* to calculate overall QoS satisfactory values.

A *quality matrix*, $\Phi = \{V(Q_{ij}); 1 \leq i \leq m; 1 \leq j \leq n\}$, is defined as a collection of quality attribute-values for a set of candidate services, where $V(Q_{ij})$ represents the value of the i^{th} QoS attribute for the j^{th} candidate service. These values are obtained from candidate service descriptions and mapped to a scale between 0 and 1.

An *overall QoS score function* is defined as

$$f_{QoS}(Service_j) = \sum_{i=1}^{m} (V(Q_{ij}) \times Weight_i)$$

where m is the number of QoS attributes in Φ, Weight$_i$ is the weight value (specified by user) for each attribute.

The f_{QoS} score is calculated for each candidate service, and if the f_{QoS} score is greater than some user defined threshold, the corresponding service will be selected. Take an example, a user requesting an online streaming video service considers *throughput, response time* and *availability* more valuable than other QoS parameters and specifies the QoS selection criteria as Weight$_{Throughput}$ = 0.8, Weight$_{ResponseTime}$ = 0.9, Weight$_{Availability}$ = 0.7, and a threshold score value U$_{Threshold}$ = 1.5. Assume there are three candidate services for online streaming video, S_1, S_2 and S_3, and the quality matrix is:

$$\Phi = \left(\begin{array}{rccc} & S_1 & S_2 & S_3 \\ \mathit{Throughput} & 0.90 & 0.80 & 0.50 \\ \mathit{ResponseTime} & 0.90 & 0.80 & 0.60 \\ \mathit{Availability} & 0.90 & 0.50 & 0.40 \end{array} \right)$$

After calculation of their respective f_{QoS} scores, only S_1 and S_2 will be selected. Further assume that the user specifies to rank the services based on *Cost* in ascending order, and the *Cost* of S_1 is greater than that of S_2 , the results returning to the user will be $\{S_2, S_1\}$, specifying that S_2 is a better choice than S_1 for the user's purpose.

2.3 Context Policies

Context policies are rules related to context information. Some context information can greatly affect the selection of services. For instance, for a home food delivery service, the user's location is an important aspect for selecting possible service providers. In addition, depending on service types, different context information should be considered. Examples of context information include location, time, connection (e.g., if the user is accessible via a wireless or wired connection), user's feeling, presence, and user's habits and hobbies.

Context policies can be specified in the same format as business policies and QoS policies. For example, a service provider for a home food delivery service may specify a location-based policy as "only deliver food within the same city". This location policy can be specified as:

IF ($UserLocation.city = ServiceProviderLocation.city$)
THEN Service can be provided.

Some services are context-aware, others are not. For context-aware services, it is preferable that the context information can be automatically integrated into the service request even though the user does not specify them explicitly. For instance, there are several approaches to identify the location of a mobile user. One method to position the mobile user is to leverage the SS7 network to derive location. Another example is user profiles, which usually define user preferences, such as habits, hobbies, and other personalized information, such as access rights and startup applications. However, mechanisms for obtaining such context information are outside the scope of this paper. We just demand that context information should be included as a part of service request wherever possible so that context policies can be used during service discovery.

3 Integrated Semantic Service Discovery Framework

3.1 Integrated Semantic Service Description

There have been efforts to add semantics to service descriptions. Two major approaches based on ontology are OWL-S [3] and semantic-annotated WSDL [16]. OWL-S uses an OWL-based ontology for describing Web Services and supports service discovery at the semantic level. The semantic-annotated WSDL approach relates concepts in WSDL to OWL ontologies in Web Services descriptions, i.e., WSDL or UDDI. We adopt semantic-annotated WSDL to describe services, because WSDL has been accepted as the industry standard for Web Services description and most of the existing Web Services support WSDL standards. This has the advantage of having widely acceptance without adding significant complexity.

Figure 5 demonstrates the semantic annotation for our integrated service description approach. An ontology-based semantic service description is represented as a semantic-annotated WSDL file, with links to the ontology definition and WS-Policy file for policy definitions and provided QoS parameters. This semantic-annotated WSDL is an XML-formatted Web Service description document based on WSDL, and is extended with OWL-based ontologies to add semantics to WSDL elements. The WSDL file *PurchaseBookService.wsdl* specifies the functional properties in terms of operations, inputs and outputs. Preonditions and effects for the operation can also be specified [17]. The concepts of them are referred to concepts in the service ontology. Policies for service discovery are specified in OWL file *PolicyRule.owl*, while WS-Policy framework and WS-PolicyAttachment are utilized to attach them to WSDL. In order to incorporate QoS parameters into WSDL without significant changes to existing WSDL structure, QoS parameters and other service parameters are also specified in an OWL file *QoS.owl*, and attached to WSDL using the same mechanism as the policy file. In detail, this means that all policies related to the service as well as QoS and service parameters can be specified in one XML file, *Policy.xml*, with links to respective OWL files, as shown in Fig. 6. This policy specification can then be attached to WSDL description, as shown in Fig. 7. As to be noted, we assume there is a shared ontology for each service domain, the same stands for policy and QoS ontologies. At the same time, a local ontology can be extended based on shared ontology to accommodate special needs.

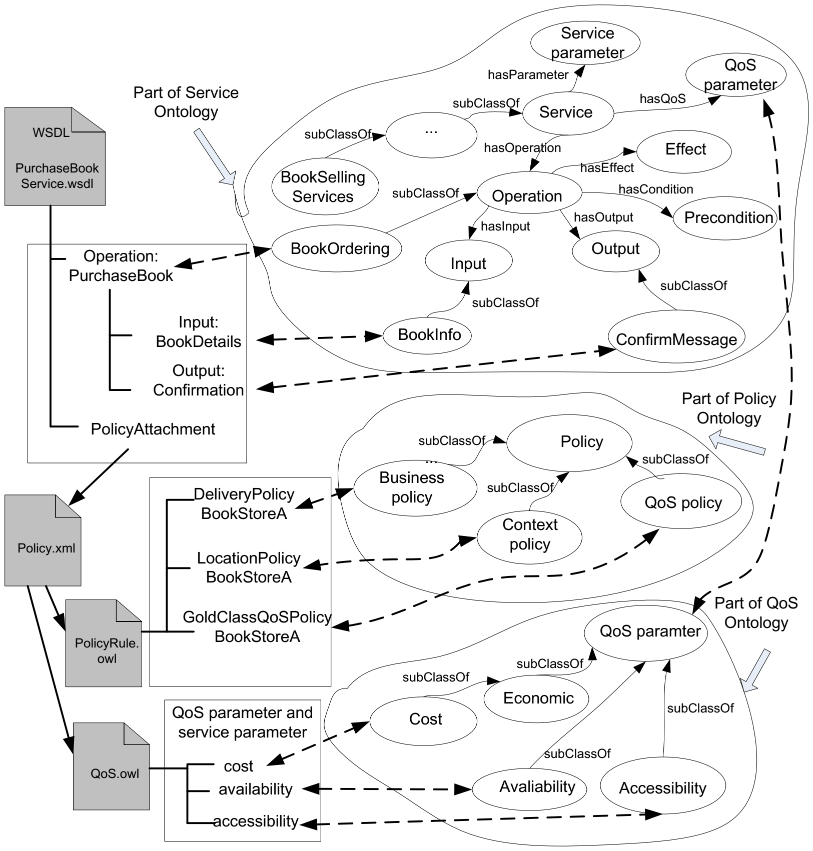

Fig. 5. Semantic annotated service description for integrated service discovery

```
<wsp:Policy Name="PurchaseBook">
  <wsp:All>
    <!– Business policy –>
    <po:DeliveryPolicy>http://BookStoreA.com/PolicyRule.owl#DelieveryPolicyBookStoreA
    </po:DeliveryPolicy>
    <!– Context policy –>
    <po:LocationPolicy>http://BookStoreA.com/PolicyRule.owl#LocationPolicyBookStoreA
    </po:LocationPolicy>
    <!– QoS policy –>
    <po:QoSPolicy>http://BookStoreA.com/PolicyRule.owl#GoldClassQoSPolicyBookStoreA
    </po:QoSPolicy>
    <!– QoS parameters and other service parameters –>
    <po:QoSParameters>http://BookStoreA.com/QoS.owl#QoSParametersBookStoreA
    </po:QoSParameters>
    <!– other policy for PurchaseBook, e.g. security policy –>
    ...
  </wsp:All>
</wsp:Policy>
```

Fig. 6. *Policy.xml* - All policy specification for Web Service *PurchaseBookService*

```
<wsp:PolicyAttachment>
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:ServiceName Name="PurchaseBookService" />
      <wsa:PortType Name="PurchaseBookPortType" />
      <wsa:Address URI="http://BookStoreA.com/PurchaseBookService" />
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  <wsp:PolicyReference URI="http://BookStoreA.com/Policy.xml" />
</wsp:PolicyAttachment>
```

Fig. 7. Attaching policy specification to WSDL file *PurchaseBookService.wsdl*

3.2 Integrated Semantic Service Requirement

A user request specifies the functional requirements, non-functional requirements and user defined selection criteria in terms of preferred QoS parameters and their weights. Such request is also based on ontology concepts. A request template can be provided. This user request is combined with automatically obtained context information to produce an integrated service requirement specification in the form of an *integrated semantic service request*, which comprises the following information:

– *Functional requirements* in terms of operations, inputs, outputs, preconditions and effects.
– *Non-functional requirements*, such as QoS constraints.
– *Context information* obtained automatically by the system.
– *User specified selection criteria*, i.e. QoS parameters and their weights as well as user specified ranking criteria. This allows for personalized service ranking.

3.3 Integrated Semantic Service Discovery Procedure

Integrated semantic service discovery process can be arranged in two major steps. The first step is to find out the services that meet the functional requirements based on matching of functional properties. As there is usually more than one service matching the functional requirements, a set of candidate services are obtained. The next step is therefore to select the most appropriate ones from these candidates based on non-functional properties and rank them according to user defined criteria.

The whole discovery process is carried out by a reasoning engine. When an integrated semantic service request is sent to the reasoning engine, the engine will first determine the candidate services that offer the requested functionality based on matching of functional properties. We adopt a procedure based on ontological inference and degree of match [13]. This procedure typically uses subsumption reasoning to find similarity between service descriptions and service requests based on operations, inputs and outputs. Preconditions and effects can also be used for matching. During the second step, policies will be checked and applied to further select services among the candidate services. The semantic-annotated WSDL files of candidate services contain links to all policy specification file (e.g. *Policy.xml*), which can be referenced to retrieve the related policy rules (*PolicyRule.owl*) as well as QoS and service parameters (*QoS.owl*). Rule-based reasoning can then be applied to determine satisfied matching. After that, overall QoS scores for those candidate services are calculated based on user defined selection criteria (i.e. based on selected QoS parameters and their respective weights) as described in Sect.2.2. All the matches will be returned according to user specified ranking criteria.

Ontological inference and rule-based reasoning are applied during semantic service discovery process. The reasoning engine which carries out the above procedure is based on XDD [23] - a knowledge representation framework - and XET [2] - a powerful computing and reasoning engine for XDD. Work has already been done, based on this representation framework and reasoning engine, for dynamic service configuration [1], composition [20], and management [9], proving the practicability and reasoning power of such reasoning engine.

XDD (XML Declarative Description) is an expressive XML rule-based knowledge representation, which extends ordinary, well-formed XML elements by incorporation of variables for an enhancement of expressive power and representation of information into so called *XML expressions*. A description in XDD is a set of XML expressions and the XML elements' relationships in terms of *XML clauses*. XML expressions represent facts, while XML clauses express rules, conditional relationships, constraints and ontological axioms. Applying XDD framework, Ontology-annotated WSDL descriptions, concepts and properties in OWL-based ontologies, service parameters and QoS parameters can all be represented as facts using XML unit clauses. Ontological relations and axioms as well as policy rules for service discovery can be represented as rules using XML non-unit clauses. Rules can also be defined for ontological inference and querying in XDD. Service requests can be represented as XDD query clauses using XML clauses, which specify the patterns as well as the selection conditions of the queries. This means all information and rules for integrated semantic service discovery can be directly represented as XDD descriptions. Furthermore, XDD descriptions can be computed and reasoned using XET (XML Equivalent Transformation), a Java-based reasoning engine that transforms the query clause by the XDD-based rules based on equivalent transformation [2]. Therefore, by expressing ontologies and rules directly in XDD and executing service discovery queries using XET-based engines, we eliminate the overhead of transforming between ontology language and rule-based representation, and can achieve both ontological inference and rule-based reasoning, two fundamental functionalities for semantic service discovery process.

4 Related Work

Several approaches for ontology-based semantic service discovery have been proposed, based on OWL-S [13] or semantic-annotated WSDL [16]. However, both of them only apply ontology for matching on the operational interfaces (i.e. input and output parameters of the operations of the Web Services). In addition, both of them lack mechanisms to represent non-functional properties based on rule-based policies. We extend the semantic matching and selection based on non-functional properties, i.e. ontology-based policy rules and QoS parameters.

Sriharee et al. [18] proposed to discover Web Services based on business rules policy using WS-Policy and ontology, but without further consideration of QoS attributes. For incorporating QoS attributes with service discovery, Zhou et al. [24] proposed a DAML-QoS ontology for specifying various QoS properties and metrics. However, there was no provision for the users to specify ranking criteria (based on non-functional properties) for service selection. The framework proposed by Pathak et al. [14] provides QoS-based service selection; however, there was no consideration for policy rules during the discovery process. Maximilien et al. [10] proposed a framework and ontology for service selection also considering QoS properties, but there was no provision for user-specified ranking criteria in service request.

5 Conclusions

An approach to integrated semantic service discovery is presented. We first describe how non-functional properties are expressed based on business policies, QoS policies and context policies as well as QoS parameters. We then present our approach for adding semantics to service description for both functional and non-functional properties based on ontologies. We further show how service request can be integrated with context information and personalized ranking criteria. Based on them, an integrated semantic service discovery procedure based on both functional and non-functional properties is presented.

We base our work on widely accepted standards in Web Services and Semantic Web, i.e., WSDL, OWL and WS-Policy. The integrated semantic service discovery approach is a rather generic one, and can be applied in a centralized or distributed environment. We are working towards mechanisms to apply this approach to autonomic environments with distributed, self-organizing and scale-free communications. Issues about how the service descriptions are stored and organized as well as how they are accessed need further exploration.

Shared ontologies are assumed for service descriptions and service requests in our approach. If different ontologies are used, ontology mapping should be carried out to build up correspondence between ontologies used for service descriptions and those used for service requests.

References

1. F. A. Aagesen, P. Supadulchai, C. Anutariya, and M. M. Shiaa. Configuration management for an adaptable service system. In *IFIP Int'l Conference on Metropolitan Area Networks, Architecture, Protocols, Control and Management, proceedings*, Ho ChiMinh City, VietNam, 2005.
2. C. Anutariya, V. Wuwongse, and V. Wattanapailin. An equivalent-transformation-based xml rule language. In *Int'l Workshop Rule Markup Languages for Business Rules in the Semantic Web, proceedings*, Sardinia, Italy, 2002.

3. The OWL Services Coalition. Owl-s: Semantic markup for web services, 2003. `http://www.daml.org/services/owl-s/1.0/owl-s.html`.
4. The Salutation Consortium. Salutation architecture specification version 2.0c, 1999. `http://www.salutation.org/`.
5. S. Bajaj et al. Web services policy attachment, 2006. `http://www-128.ibm.com/developerworks/library/specification/ws-polatt/`.
6. S. Bajaj et al. Web services policy framework (ws-policy), 2006. `http://www-128.ibm.com/developerworks/library/specification/ws-polfram/`.
7. UPnP Forum. Upnp device architecture version 1.0, 2000. `http://www.upnp.org/`.
8. E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC2608, 1999.
9. S. Jiang, M. M. Shiaa, and F. A. Aagesen. An approach for dynamic service management. In *EUNICE'04, Proceedings*, Tampere, Finland, 2004.
10. E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8(5):84–93, 2004.
11. Sun Microsystems. Jini architecture specification version 2.0, 2003. `http://www.jini.org/`.
12. OWL. Owl web ontology language overview. W3C Recommendation, Feb 2004. `http://www.w3.org/TR/owl-features/`.
13. M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *First Int. Semantic Web Conf., Proceedings*, 2002.
14. J. Pathak, N. Koul, D. Caragea, and V. Honavar. A framework for semantic web services discovery. In *WIDM'05, Proceedings*, 2005.
15. S. Ran. A model for web services discovery with qos. *ACM SIGecom Exchanges*, 4(1):1–10, 2003.
16. K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *ICWS'03, Proceedings*, 2003.
17. N. Sriharee and T. Senivongse. Discovering web services using behavioural constraints and ontology. In *DAIS'03*, volume 2893 of *LNCS*, pages 248–259. Springer, 2003.
18. N. Sriharee, T. Senivongse, K. Verma, and S. Sheth. On using ws-policy, ontology, and rule reasoning to discover web services. In *INTELLCOMM 2004, Proceedings*, volume 3283 of *LNCS*, pages 246–255, Bangkok, Thailand, 2004. Springer.
19. R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, 1998.
20. P. Supadulchai and F. A. Aagesen. A framework for dynamic service composition. In *First Int'l IEEE Workshop on Autonomic Communications and Computing, Proceedings*, Taormina, Italy, 2005.
21. uddi.org. Universal description, discovery and integration of web services. `http://www.uddi.org/`.
22. W3C. Web services description language (wsdl)1.1, 2001. `http://www.w3.org/TR/wsdl`.
23. V. Wuwongse, C. Anutariya, K. Akama, and E. Natajeewarawat. Xml declarative description: A language for the semantic web. *IEEE Intelligent Systems*, 16(3):54–65, 2001.
24. C. Zhou, L. Chia, and B. Lee. Service discovery and measurement based on daml-qos ontology. In *Special Interest Tracks and Posters of 14th World Wide Web Conference, Proceedings*, 2005.