Semantic and QoS based Web Service Selection using a Multi Agent System

Sathyavathy P, Sneha C, Uma B, Swamynathan S and Geetha T V

Department of Computer Science and Engineering,
College of Engineering, Guindy,
Anna University,
Chennai – 600025, INDIA

{sathyavathy.p@gmail.com, sneha.chandrababu@gmail.com,
uma.balu@gmail.com, swamyns@annauniv.edu, rctamil@annauniv.edu}

Abstract. Currently all web services are published and found by making use of the WSDL and UDDI techniques. This forces the consumers to find and bind services at design time since WSDL relies on static descriptions. Also the current techniques don’t address runtime service selection based on a dynamic assessment of QoS attributes. Hence in this paper, a Semantic QoS Web Services Agent Framework (SQoSWSAF) architecture has been proposed and implemented that can dynamically select a web service based on user’s requirements, QoS constraints and preferences. This requires developing service and QoS ontologies and adding semantic descriptions to web services using these ontologies, which enables the automation of web service discovery, selection and composition. Dynamic service selection based on user constraints requires an agent-based system. Thus a multi-agent framework, which involves an agent querying the web service registry for the set of advertised services and selecting the best service, is implemented in the SQoSWSAF system.

1 Introduction

Web services are software components or resources that can be accessed using standard XML-based messaging across a network environment such as the Internet. The Semantic Web Services vision is to describe Web Service’s capabilities and content in an unambiguous, computer- interpretable language and improve the quality and robustness of existing tasks, such as Web service discovery and invocation [7]. Semantic Web Services [8] will also enable a broad range of new automation tasks that humans previously performed, including automated composition, interoperation, execution monitoring, and recovery. As multiple web services offering similar functionality are emerging, QoS will become one of the key factors in service competition and selection [1]. Also, when web services populate into more domains, QoS-guarantees will be required by more applications for their proper operations.

An agent is anything that can be viewed as perceiving its environment through sensors and acting through effectors. From the end-user perspective a software agent is a program that assists people and acts on their behalf. Agents function by allowing
people to delegate work to them. Combining these two definitions, we consider software agents to be small software tools that exist to help people doing a specific kind of work, to simplify given tasks and automate varying assignments. To complete the given tasks software agents possess many different abilities – the most important one, the ability to communicate with other systems and agents. The autonomous nature of agents also offers flexibility in dealing with uncertain situations and localized failures. The Multi Agent System (MAS) provides an environment for the agents to exist in and perform their activities. Dynamic service selection [1] based on user constraints requires an agent-based system.

Ontology is a specification of a conceptualization. Also ontology formulates an exhaustive, rigorous conceptual schema within a given domain. It is a hierarchical data structure containing all relevant entities, relations and rules. To represent a conceptualization a representation language is need. Web Ontology Language (OWL) is one such language for defining and instantiating web ontologies.

Current systems for web service discovery do not enable dynamic service selection based on QoS parameters and service’s semantic descriptions. Furthermore, consumers might have specific QoS constraints and requirements. Hence a system (SQoSWSAF) is proposed which is capable of dynamically selecting the service which best suits the user’s QoS requirements and needs, using a multi-agent system. A multi-agent system is needed because the goal of the entire system is subdivided into interacting sub goals and must be assigned to autonomous entities.

Chintan Patel, et al. in their paper [11], discusses about a QoS model for Web Service selection, binding and execution, but the semantic description of a web service described in our work is not considered for selecting a web service.

The rest of this paper is organized as follows. Section 2 provides a brief introduction to SQoSWSAF architecture. Then the dynamic web service selection process is discussed in Section 3. Section 4 explains the implementation details. Results are discussed in Section 5. Finally, in Section 6, the conclusions and future work are summarized.

2 SQoSWSAF Architecture

When a consumer application built with SQoSWSAF (Semantic QoS Web Services Agent Framework) wants to access the services provided, agents are employed to communicate with the services. The following section discusses the integration of the Web Services with the richer semantics of the Semantic Web through the use of expressive languages for service description. QoS awareness is brought into the SQoSWSAF by allowing both service providers and service requestors to specify QoS related statements to enable QoS aware service lookup, and service selection and service consumption. Thus service requestors can dynamically bind to services which provide the best QoS offer to their specific requirements at a time. The consumer application and the semantic web services shares the same ontology. Thus, similar to QoS, run time service selection based on semantic information is also enabled. Fig. 1 shows the proposed SQoSWSAF architecture.
2.1 Service Consumer

The authorized user submits his QoS constraints and requirements to the Intelligent Agent. The stored user profile and preferences are loaded onto the GUI. The user can select his domain of requirements and the services for which he wants to specify his preferences. The user may also redefine his QoS preferences of services. The Intelligent Agent and the service providers share the same service and QoS ontologies. The user provides his current constraints, preferences, service, input and output requirements in the domain he has selected and submits them. Then, these user’s requirements are sent to the appropriate Planner Agent. The Planner Agent performs composition of services and has a generic plan which is executed on receiving a request message from the Intelligent Agent. There is a separate Planner Agent for each domain specified by the Intelligent Agent, for example, Travel Planner Agent for the travel domain. Depending on the plan, the respective services are requested to the Retrieval Agent by the appropriate Planner Agent. A generic plan has a set of predefined instructions that is domain dependent. For example, in travel domain, the plan defined includes a condition such as, if the travel distance is less than 300 kilometers, book a taxi, else book a flight.

2.2 Service Providers

The Service Providers create web services. The starting point for developing a Web service is the service endpoint interface. A service endpoint interface (SEI) is an interface that declares the methods that a client can invoke on the service. On building and deploying the service, the WSDL file of the web service is automatically generated. The semantics of a web service are described using OWL-S and are also deployed.

2.3 Registry

The services created are published in an UDDI registry. Publishing these services in the registry promotes the discovery of the services by querying the registry. The OWL-S service instance, profile, process model and grounding files that describes a web service are manually created, that adds semantic information to web services. This file is published along with the WSDL file.
2.4 Service Broker

The Retrieval Agent queries the Registry and receives the OWL-S descriptions of the advertised services. These service descriptions along with the user’s constraints are sent to the Reasoning Agent, which performs the inferencing function. The advertised service semantics and QoS information along with the user’s constraints are also loaded into the Knowledge Base of the Reasoning Agent. Then, inferencing and reasoning is performed based on the concepts (semantics) and QoS information to find the service that best satisfies the user’s requirements. The best service’s semantic descriptions are passed to the Planner Agent. The Planner Agent continues with its service composition and returns the plan to the user through the Intelligent Agent.

Fig. 1. Semantic QoS Web Services Agent Framework Architecture
3 Dynamic Web Service Selection

On receiving the set of advertised services from the Retrieval Agent the Reasoner Agent dynamically selects the web service that best suits the user’s requirements. The user’s requirements are asserted in the Knowledge Base. For each of the advertised service, semantic and QoS matching of the service profile with the user’s requirements is performed. Then the set of advertised services that satisfy the user’s requirements are analysed and the best service is selected. Matching of user’s requirements with the service profile of the advertised service is performed as follows. Two types of matching is performed, they are Profile matching and QoS matching.

3.1 Profile Matching

Profile matching determines how good the service category of the advertised service matches that of the user’s request. Let A and B denote the service classification of the user’s request and advertised service. Both A and B are concepts defined in some service classifying ontology. The matching degrees for profile matching include exact match (3), subsumes (2), unclassified (1) and fail (0). If the profile matching degree is exact match then the concepts A and B are equivalent. If it is subsumes then the concept B is subsumed by the concept A. Here concept B denotes a more specific concept than A. If the matching degree is unclassified then either concept A or B is not classified, i.e. one of them is a direct instance of the class Profile, which belongs to OWL-S 1.0 specification. Otherwise, the two concepts A and B are in no relation with each other. Fig. 2 shows the Profile matching and Fig. 3 shows Concept matching.

The Inferencer holds the Knowledge Base and performs concept matching of two concepts. Two concepts are equivalent (3) if they belong to same class in the service classification hierarchy. The concept match result for two concepts A and B is subsumes_invert (1) if concept A is a more specific concept than concept B.
int profile_matching(String serviceRequest, String serviceAdvertise, 
Inferencer inferencer) {
    if (serviceRequest ==null || serviceAdvertise ==null)
        return UNCLASSIFIED;
    if(serviceRequest.equals("http://www.daml.org/services/owl-
    s/1.0/Profile.owl#Profile") ||
    serviceAdvertise.equals("http://www.daml.org/services/owl-
    s/1.0/Profile.owl#Profile"))
        return UNCLASSIFIED;
    else {
        int match = inferencer.conceptMatch(serviceRequest, 
        serviceAdvertise); 
        if (match == EQUIVALENT)
            return MATCH;
        if (match == SUBSUMES)
            return SUBSUMES;
        else
            return FAIL;
    }
}

Fig. 2. Profile Matching

int conceptMatch(String conceptA, String conceptB) {
    if (conceptA.equals(conceptB) || sameClass(conceptA,conceptB))
        return EQUIVALENT;
    else if (subsumes(conceptA,conceptB))
        return SUBSUMES;
    else if (subsumes(conceptB,conceptA))
        return SUBSUMES_INVERT;
    else
        return FAIL;
}

Fig. 3. Concept Matching

3.2 QoS Matching

QoS matching determines how good the QoS levels of the advertised service match 
that of the QoS levels of the user’s request. The matching degrees of QoS matching 
are equivalent to that of the concept matching degrees i.e. equivalent, subsumes,
subsumes_invert and fail but the order of priority of the matching degree is as follows: subsumes (3), equivalent (2), subsumes_invert (1) and fail (0). Fig. 4 shows the QoS matching procedure.

The serviceParameter construct of OWL-S 1.0 specification is used for adding QoS information to web services. For example, Fig. 5 shows the snippet of service profile of where QoS information is added to a web service.

The set of serviceParameter names of the advertised service is identified as advQoSList. Similarly the set of requested QoS parameters is reqQoSList. The reqQoStype and advQoStype specifies the type of the quality parameter such as reliability, response time and throughput. The reqQoSsparameter and advQoSsparameter describes the value of the quality parameters as a concept, which is defined in some QoS ontology.

If a quality parameter is monotonically_increasing, then the greater the value of the parameter the better, for example, reliability is a monotonically_increasing quality parameter. On the other hand, if a parameter is decaying, then the lesser the value of the parameter the better, for example response time is a decaying quality parameter. Depending on the quality parameter’s attribute, the QoS matching is performed.

From the set of advertised services that satisfy the user’s requirements (i.e., those advertised services that doesn’t return degree fail [0]), the best service is selected as follows. The set of services that provide a maximum profile degree are selected. From this set, the set of services with the maximum QoS degree is selected and the first service in this set is selected as the best service.

```c
int QoS_matching(reqQoSList, advQoSList, Inferencer inferencer) {  
    int mindegree=SUBSUMES;  
    for each reqQoS, reqQoStype and reqQoSsparameter in reqQoSList  
        for each advQoS, advQoStype and advQoSsparameter in advQoSList  
            if(reqQoStype==advQoStype) {  
                Get the list of superclasses of the advQoStype;  
                for each superclass in superclasses  
                    if(superclass==“monotonically_increasing”) {  
                        matchdegree=inferencer.conceptMatch(advQoSsparameter,reqQoSsparameter);  
                        switch(matchdegree) {  
                            case SUBSUMES: break;  
                            case EQUIVALENT:  
                                if(mindegree>2)  
                                    mindegree=EQUIVALENT;  
                                    break;  
                            case SUBSUMES_INVERT:  
                                if(mindegree>1)  
                                    mindegree=SUSMUSES_INVERT  
                                    break;  
                            case FAIL: return FAIL;  
                        }  
                    }  
            }  
}  
```
else if (superclass=="decaying") {
    matchdegree=inferencer.conceptMatch(reqQoSsparameter,advQoSsparameter);
    switch(matchdegree) {
        case SUBSUMES: break;
        case EQUIVALENT:
            if(mindegree>2)
                mindegree=EQUIVALENT;
            break;
        case SUBSUMES_INVERT:
            if(mindegree>1)
                mindegree= SUBSUMES_INVERT;
            break;
        case FAIL:
            return FAIL;
    } //end if advQoStype == reqQoStype
    return mindegree;
}

Fig. 4. QoS Matching

<?xml version="1.0" encoding="UTF-8"?>
<profile:serviceParameter>
    <qos:Reliability rdf:ID="QoS-Reliability">
        <profile:serviceParameterName>BookTicketServiceReliability</profile:serviceParameterName>
        <profile:sParameter rdf:resource="http://localhost:1170/ontology/QoS.owl#lessthanorequal50"/>
    </qos:Reliability>
</profile:serviceParameter>

Fig. 5. Addition of QoS information to a web service using the serviceParameter tag of OWL-S 1.0 specification

4 Implementation

The SQoSWSAF system can be demonstrated with a comprehensive scenario that involves travel domain. Automatic selection of the service that best suits the user’s requirements is performed. A user interface for giving the user’s travel requirements is provided.

The user selects his mode of travel (flight, taxi) and specifies his preferences on the class in which he wants to travel and the maximum expenditure. Also, he is allowed to specify the type of service he wants along with specific QoS requirements
such as reliability, response time and throughput all of which follows the same ontology as that of the service provider.

This sample application involves travel domain. Hence the Intelligent Agent selects the Travel Planner Agent. The Travel Planner Agent has the generic plan for traveling. Depending on the distance to be traveled, the Travel Planner Agent chooses the mode of travel. The user’s requirements and QoS preferences are sent to the Service Broker, which chooses the best service and returns its OWL-S descriptions. The SQoSWSAF system implements the selection of the best flight service using the FlightQoS ontology and service ontology. This work can be easily extended to include other services and domains. Evaluating SQoSWSAF system and its Service and QoS ontology involves scenarios including consumers with different QoS and service requirements.

The SQoSWSAF is implemented using the following technologies. The web services are created using JAX-RPC and are built and deployed in the Sun Java Application Server Edition 8, which is a J2EE 1.4 platform compatible server for the development and deployment of J2EE applications and Java Web Services [11]. The Service and QoS ontologies are developed using the Web Ontology Language (OWL) [5]. Fig. 6 shows the Profile Hierarchy of the web services in the travel domain. OWL-S 1.0 is the language used for describing and adding semantics for web services. It consists of a service instance, one or more profile instances, process model and grounding [6].

The service profile specifies, "what the service does", i.e., it gives the type of information needed by a service-seeking agent to determine whether the service meets its needs. The ServiceProfile is used for advertising, registry, discovery, and matchmaking. Once a potential service-using agent has located a service appropriate for its need, the ServiceModel and ServiceGrounding gives enough information for an agent to make use of the selected service. Service Process Model describes the name and IOPEs (Input Output Preconditions Effects) of a process. Processes can be atomic processes or composite processes. Service grounding specifies the details of how an agent can access a service and service-specific details such as port numbers used in contacting the service.

The JAXR API is used for publishing the services in the Registry. The JWSDP Registry Server implements UDDI Version 2 to provide a UDDI Registry in the private environment. It can be used with JAXR API as a test registry for Web Services development. The Reasoner Agent uses OWLJESSKB, which is a description logic reasoner for the W3C's Ontology Web Language, for selection of the best service. The semantics of the language is implemented using JESS, the Java Expert System Shell. Jess supports the development of rule-based expert systems. The agents are developed using JADE (Java Agent DEvelopment Framework), which is a robust and efficient environment for distributed agent systems and is a software framework fully implemented in Java language.

The defquery construct of JESS is used for querying the Knowledge Base. The defquery construct allows creating a special kind of rule with no right-hand-side. While rules act spontaneously, queries are used to search the knowledge base under direct program control. A rule is activated once for each matching set of facts, while a query returns a java.util.Iterator of all the matches.
5 Results and Discussion

To examine the accuracy of the selection process the QoS advertisement profiles of three web services in travel domain were considered. A sample user’s service and QoS requirement is also considered. The service broker chooses the best service from the advertised service profile and the user’s request.

Fig. 7 shows the user’s service requirements and QoS preferences. AirlineTicketing and LandlineTicketing are subclasses of Travel as specified in the service classification, i.e. Profile Hierarchy, in Fig. 6. Fig. 8 shows the service profile of the three advertised services.
The reliability, throughput and response time are common QoS parameters defined in the QoS upper ontology. The domain specific QoS parameters are defined in QoS lower ontology. Fig. 9 shows the best service selection process.

Fig. 7. User's Requirements

```
ProfileHierarchy:AirlineTicketing
qos:Reliability # greaterthan80
qos:Throughput # greaterthan90
qos:ResponseTime # lessthanorequal10
flightqos:FirstClass# lessthan15000
```

ADVERTISED SERVICE 1

```
ProfileHierarchy:AirlineTicketing
qos:Reliability # greaterthan80
qos:Throughput # greaterthan95
qos:ResponseTime # lessthanorequal15
flightqos:FirstClass# lessthan12000
```

ADVERTISED SERVICE 2

```
ProfileHierarchy:AirlineTicketing
qos:Reliability # greaterthan90qos:Throughput
#greaterthan95qos:ResponseTime # lessthanorequal15flightqos:FirstClass
#lessthan12000
```

ADVERTISED SERVICE 3

```
ProfileHierarchy:Travel
qos:Reliability #greaterthan50
qos:Throughput #greaterthan95
qos:ResponseTime #lessthanorequal160
flightqos:FirstClass #lessthan12000
```

Fig. 8. Snippet of Service Profile of three Advertised Services
Advertised Service URL:
http://localhost:1170/airindia/TicketBookingService.owl
Profile Degree : 3 [ PROFILE EQUIVALENT ]
QoS Degree : 2 [ QoS EQUIVALENT ]

Advertised Service URL:
http://localhost:1170/sahara/TicketBookingService.owl
Profile Degree : 3 [ PROFILE EQUIVALENT ]
QoS Degree : 3 [ QoS SUBSUMES ]

Advertised Service URL:
http://localhost:1170/jetair/TicketBookingService.owl
Profile Degree : 2 [ PROFILE SUBSUMES ]
QoS Degree : 1 [ QoS SUBSUMES_INVERT ]

Best Service URL:
http://localhost:1170/sahara/TicketBookingService.owl

Fig. 9. Best Web Service Selection

6 Conclusions and Future Work

The SQoSWSAF system is implemented using the travel domain. Service ontologies
are developed for the travel domain, which can be extended easily to include other
domains. Upper and middle level QoS ontologies are also developed which are
domain independent while the lower QoS ontologies are service specific. Several web
services, which are travel domain specific, have been created, built and deployed in
the Sun Java Application Server Edition 8. OWL-S service descriptions of the web
services developed are created manually to add semantics to web services by
importing the above-mentioned OWL ontologies. These services are then registered
in the registry using the JAXR API.

SQoSWSAF system makes use of JADE to provide a multi-agent environment that
enables dynamic service discovery, selection and composition. An agent provides an
interface to the user, through which the user can specify his service requirements,
QoS constraints and preferences. This agent has the ability to record dynamically the
changes in the user’s preferences. Depending on the service requirements the
appropriate planner agent is selected and an ACL message with the user’s
requirements and constraints is sent to that planner agent. The planner agent has
generic stored procedures for its particular domain and depending upon this generic
plan service requests are sent to the Service Broker.

The Service Broker retrieves the set of advertised services from the registry using
JAXR API and their OWL-S service descriptions are retrieved. These OWL-S service
descriptions are parsed using OWLJESSKB, which translates them into triples and
asserts them in the Knowledge Base. The inferencer performs the matching of the
advertised services with the service request by querying the knowledge base and dynamically selecting the service that best satisfies the users’ requirements.

The evaluation of SQuoSWSAF is performed using consumers with different QoS preferences and service requirements and services with different QoS advertisements and service specification. By artificially controlling the service qualities (such as reliability, response time, throughput, etc), the accuracy of the SQuoSWSAF system is verified, i.e., whether each consumer selects the right service given his preferences. As a service QoS drops below its consumer’s QoS requirements, the agent will no longer select it; when the service’s qualities improve, it will again be considered for selection.

The SQuoSWSAF can be extended further to include automatic service execution. After the user approves the selected service, the Intelligent Agent has to find and bind to the appropriate service proxy agent. The service proxy agent is responsible for invocation of the selected web service. The proxy agent is service specific, i.e. there exists a proxy agent for each service provided by an organization. Also, the Intelligent Agent and the proxy agent should share the same ontology for communication. Improving the Planner Agent can be a further enhancement to the SQuoSWSAF system. The Planner Agent can be extended to perform automatic service composition of the selected services using the generic plan and service semantics.

References

6. OWL-S The Semantic markup for web services, http://www.daml.org/services/owl-s/1.0/owls.pdf