AN AUTOMATED END-TO-END MULTI-AGENT QOS BASED ARCHITECTURE FOR SELECTION OF GEOSPATIAL WEB SERVICES

Manan Shah*, Yogesh Verma[+], R Nandakumar[+]

*M.Tech. Student, DDIT, Nadiad, Gujarat; [+]Space Applications Centre, Indian Space Research Organisation, Ahmedabad–380015, India
(mananshah88@gmail.com), (yogeshverma, nandakumar)@sac.isro.gov.in

Commission IV/5: Distributed and Web-Based Geoinformation Services and Applications

KEYWORDS- Fuzzy, Geospatial Web Services, Multi-Agent System, Quality of Service (QoS), SOA, SOAP, UDDI, WSDL.

ABSTRACT:

Over the past decade, Service-Oriented Architecture (SOA) and Web services have gained wide popularity and acceptance from researchers and industries all over the world. SOA makes it easy to build business applications with common services, and it provides like: reduced integration expense, better asset reuse, higher business agility, and reduction of business risk.
Building of framework for acquiring useful geospatial information for potential users is a crucial problem faced by the GIS domain. Geospatial Web services solve this problem. With the help of web service technology, geospatial web services can provide useful geospatial information to potential users in a better way than traditional geographic information system (GIS). A geospatial Web service is a modular application designed to enable the discovery, access, and chaining of geospatial information and services across the web that are often both computation and data-intensive that involve diverse sources of data and complex processing functions.
With the proliferation of web services published over the internet, multiple web services may provide similar functionality, but with different non-functional properties. Thus, Quality of Service (QoS) offers a metric to differentiate the services and their service providers. In a quality-driven selection of web services, it is important to consider non-functional properties of the web service so as to satisfy the constraints or requirements of the end users. The main intent of this paper is to build an automated end-to-end multi-agent based solution to provide the best-fit web service to service requester based on QoS.

1. INTRODUCTION

Service Oriented Architecture (SOA) can be achieved through Web services, which are self-contained, self-describing, modular applications that can be published, located and invoked across the Web. With the support of a set of widespread industry-accepted standards like Web Service Description Language (WSDL), Universal Description Discovery & Integration (UDDI) and Simple Object Access Protocol (SOAP), Web services are easy to facilitate Enterprise Application Integration (EAI) [1].

SOA comprises of three main actors namely service provider, service requestor and service broker/registry as depicted in Figure 1.

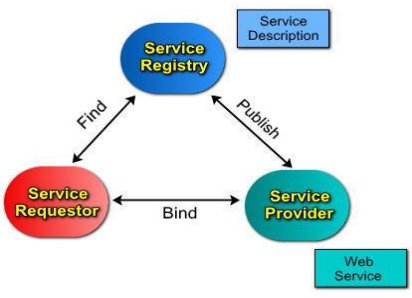


Figure 1.Service Oriented Architecture

Service providers create and publish the web service to the registry with service broker. The service requestor can discover any service from the registry and bind its application/service to Service Provider's web service. Service Broker has registry in which it maintains all the information regarding the web services as well as service providers. For discovery, service requestor passes its functional requirement to the broker, and on the basis of the requirements, the discovered list will be returned back to the service requestor. Service requestor can select any web service from the list and bind it to its application or service.

The large number of web service providers throughout the globe, have produced numerous web services providing similar functionality. This necessitates the use of tools and techniques to search suitable services available over the web. Quality of Service (QoS) is one of the decisive factors in selecting the desired web service for the requester. In selecting a web service for use, it is important to consider non-functional properties of the web service so as to satisfy the constraints or requirements of users.

The twin challenges of suitable discovery & selection leads us to take up this research problem of best-fit web service among similar web services based on functional as well as QoS parameters.

We have used the software agents. It is autonomous software entities and can react with other software entities, including humans, machines, and other software agents in various environments and across various platforms. Multi-agent systems are composed of agents coordinated through their relationships with one another [2].

Section 2 covers the related work. While section 3, describes the developed system. Section 4 illustrates the simulation results carried out on weather web service. Section 5 describes the conclusion & future scope.

2. QOS PARAMETERS

The service discovery is based on the UDDI, in which the services can be searched by using functional attributes. For the non functional attributes i.e. Quality of Service (QoS) there are various approaches or methods suggested by researchers. Lots of models and frameworks have been proposed for discovery and selection of web services based on QoS parameters as explained below.

The QoS parameters are adopted by extending the *conceptual Publishing & Discovery models* [3] [4]. Such models are mostly associated with UDDI directories. Ran et al. [3] proposed a four roles model for publishing QoS by extending the UDDI registry and a Web Services QoS certifier. The difference between the new UDDI registry and the standard one is that the new UDDI registry has information about the functional description and its associated QoS attributes. This proposal supports two directions; publishing and discovering as well as verification and certification. Serhani et al. [4] added a new component, a broker, to the original Web Service framework and used a QoS-enabled UDDI called *UDDIe* registry. The QoS-enabled *UDDIe* is a registry, which supports the publication and discovery of the QoS aware services. It supports the idea of blue pages which enables the discovery of Web Services based on user defined attributes.

Another approach is through *Web Services Description & Handling models*. Such models were associated with policy e.g. WS-Policy, which is a specification that allows Web Services to advertise their capabilities, requirements, and general characteristics in a flexible and extensible grammar using XML format [5]. A policy assertion is a requirement or rule which describes Web Services behavior and gives it a better robustness and extendibility. WS-Policy framework is used to include some QoS properties such as security, reliable messaging and transaction. Mathes et al [6] has proposed an approach to use the WS-Policy to include other QoS attributes by extending it.

Another approach is a *combination of both* the above mentioned approaches. Garcia et al. [7] proposed a combination of UDDI and WS-Policy approach. In their work, the authors extend the UDDI registry to include WS-Policy and add a broker to the standardized UDDI architecture. Ontology Web Language (OWL) as well as ABLE Rule Language (ARL) is used to enrich QoS policies with semantic information. Such enrichments allow more flexible interactions between policies.

Another approach is *SLA (Service Level Agreement)* e.g. WSLA. IBM proposes Web Service Level Agreements (WSLA) which is an XML specification of SLAs for Web Services, focusing on QoS constraints. We can not only specify the Service Level Objectives (SLO) of a service and its service operations, but also the measurement directives and measurement endpoints for each quality dimension. WSLA represents a configuration for a SLA management and monitoring system.

Another approach is Model like *DAMLS*: provided an upper ontology for semantic description of web services, including specification of functionalities and QoS constraints [8].

The main drawbacks of these models/frameworks are that they are not validating the QoS parameters provided by the service providers. Also, there is no end-to-end solution for solving the best-fit web service amongst the available similar services at broker side. We are proposing an automated end-to-end Multi-Agent QoS based architecture (AMAQ) to be implemented at broker side for selection of web services.

3. AMAQ SYSTEM IMPLEMENTATION

The Automated Multi-Agent QoS based (AMAQ) System Architecture and algorithms are described in this section.

3.1 AMAQ Architecture

The proposed architecture and functionalities are shown in Figure 2.

Web Service Registry: The Service Provider (SP) shall register the web services into registry. All the information related to web service and service provider shall be stored in the web-service registry. Service provider shall submit and update *(optional)* the Web Service QoS parameters which are to be stored in database along with *.wsdl file*. By reading that description Service Requester (SR) can use the service of SP. SP can also send QoS parameters through the Service Receptor Agent i.e. *serviceRecAgent*. This *serviceRecAgent* aids in registration of service and invokes the Parsing Agent, *parseAgent* and QoS Measurement Agent, *QoSAgent* that will aid in WSDL & QoS parameters validation.

WSDL Validation: The *.wsdl* file contains the description of the web service, provided by service provider that is stored into registry. Checking and validation of the *.wsdl* file shall be done by the Parsing Agent, *parseAgent*. In the case of incomplete information in *.wsdl*, service provider can get the feedback from the registry. Based on feedback, SP can submit the proper *.wsdl*.

QoS Parameters Validation: The QoS parameters shall be calculated by QoS Measurement Agent, *QoSAgent* and the differentiated result (Measured QoS – QoS provided by SP) shall be stored in database.

When SR places a discovery request, the Request Receptor Agent, *reqRecAgent* shall collect the functional and QoS requirements and store internally to its data structure and invoke the Discovery Agent, *discoveryAgent*.

Web Service Discovery: The SR shall discover the web service from the registry. For the discovery, SR has to provide the required web service functionalities as well as non-functional parameters. The *discoveryAgent* shall match the SR requirements with the available result sets from the QoS & Web service registry database and provide the discovered web services list to the Decision Agent, *decisionAgent*.

Web Service Selection & Ranking: The *decisionAgent* shall select & rank the web service(s) using Fuzzy Methods.

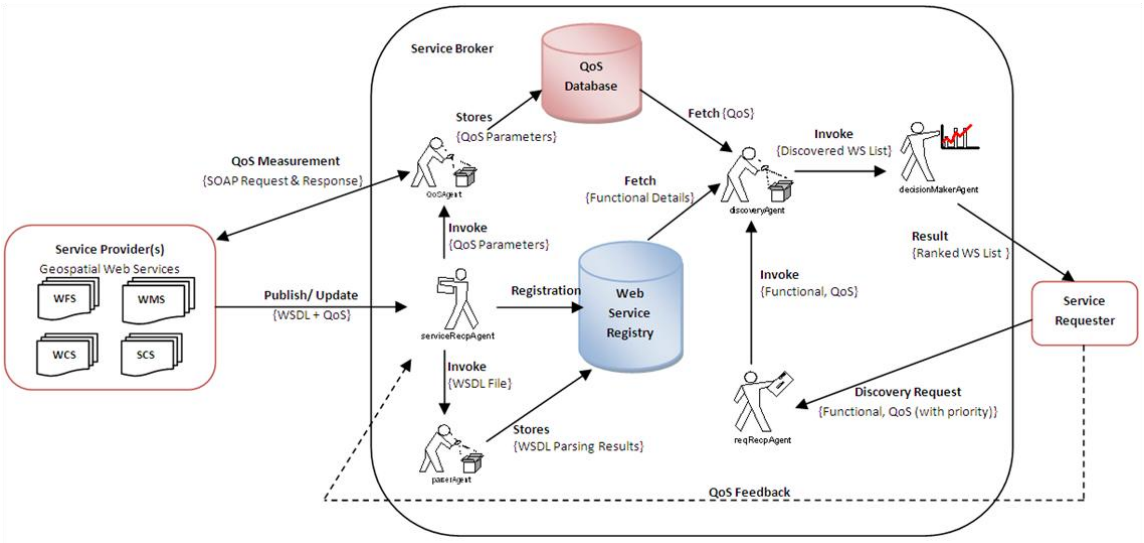


Figure 2. Automated End-to-End Multi-Agent QoS based Architecture

3.2 AMAQ Algorithms

The following algorithms were developed for realising the proposed architecture namely *serviceRecpAgent*, *reqRecAgent*, *QoSAgent*, *parseAgent*, *discoveryAgent* and *decisionAgent*.

3.2.1 *serviceRecepAgent*

Input: web service registration details, *.wsdl* file, QoS details.

Output: Stores the web service details.

Procedure:

1) *serviceRecpAgent* gets the input from SP.
2) Stores the web service details to UDDI.
3) Invokes the *parseAgent* and *QoSAgent*.

3.2.2 *parseAgent*

Input: *.wsdl* file.

Output: Parsing result.

Procedure:

1) *parseAgent* takes the input from the *serviceRecpAgent*.
2) For validation, this agent performs the following steps:
   a. First it obtains a *WSDLFactory* instance via the static *newInstance* method of *WSDLFactory* as defined in JWSDL API.
   b. Here, the purpose is only parsing the *.wsdl* documents. So after creating an object, it uses *newWSDLReader* method, to create the desired object. Any JAXP or JAXP-compliant XML parser can be used in the parsing of *.wsdl* documents.
   c. By using *getBinding()*, *getPortType()*, *getOperations()*,*getMessages()* methods, *parserAgent* retrieves all the information.
   d. Any failure in the retrieving procedure, agent considers as *invalid .wsdl* otherwise *valid .wsdl*.
3) Stores the parsing results in the QoS database.
4) In the case of invalid *.wsdl*, missing values or reason for invalidation will be displayed. And in the case of valid *.wsdl*, the wsdl details will be displayed.

3.2.3 *QoSAgent*

Input: *acess_url* of web service.

Output: QoS result

Procedure:

1) Loop (For each and every registered web service from UDDI)
   a. A *QoSAgent* fetches *acess_url* for web service.
   b. Send one request (data packet) to that URL and wait until it gets the response.
   c. Check the response code.
      i. IF response code = 1000 then IP-acess_url is not available. Then, *service_availibility* = false.
      ii. Else IF response code = 503 then IP-PORT is available but the web service is not found on that location. Then, *service_availibility* = false.
      iii. Else IF response code= 200 then web service is available. Then, *service_availibility* = true.
   d. Calculates the *service_throughput* from *service_availibility*.
   e. Stores the QoS result in a QoS database.

3.2.4 *reqRecAgent*

Input: functional description for discovery, QoS details

Procedure:

1) *reqRecAgent* gets the input from service requestor.
2) According to the requirement, it invokes the *discoveryAgent*.

3.2.5 *discoveryAgent*

Input: Functional requirements, QoS requirements.

Output: Discovered web service list.

Procedure:

1) *discoveryAgent* gets the input from the *serviceRecpAgent*.
2) Loop (For each and every registered web service from UDDI)
   a. Match the service details with requirements.
   b. Check parsing results of the service.

c. IF details matched and parsing result indicate *valid .wsdl*, then put this web service in a discovered list.

3) Sends the discovered list to the *decisionMakerAgent*

3.2.6 *decisionMakerAgent*

Input: Discovered web service list.
Output: Ranked web service list.
Procedure:

1) *decisionMakerAgent* gets the discovered web service list from the *discoveryAgent*.
2) Loop (For each and every service listed in discovered list)
   a. Fetch the QoS (availability and throughput) results of service.
   b. Calculate input membership levels.
   c. Find the fuzzy rules which can be applied to the fuzzy inputs. Evaluate the rules by using center of gravity method (COG). These rules are as per Table 1.
   d. Defuzzify the fuzzy output to system output.

Rule	Availability	Throughput	QoS
1	Bad	Bad	Bad
2	Bad	Good	Bad
3	Bad	Best	Good
4	Good	Bad	Bad
5	Good	Good	Good
6	Good	Best	Best
7	Best	Bad	Good
8	Best	Good	Best
9	Best	Best	Best

Table 1. Fuzzy Rules

3) Sorts the list according to final numeric value, which gets from defuzzification step and rank it accordingly.
4) Sends the ranked web service list to the service requestor.

4. SIMULATION OF GEO WEATHER WEB SERVICE

For validating the developed system, we have created an environment for simulating the geo-weather web services as described below:

1. **Web Service Creation**: Three geo-weather web services were developed that provide weather information when a corresponding city and country are provided as input. These SOAP based web services were deployed over Web Server.

2. **Registration to UDDI**: An interface to UDDI Server is developed. The developed web services were registered to UDDI Server.

3. **WSDL Parsing**- The wsdl parsing of the sample web services was done and the results were stored.

4. **Discovery & Selection of Web Service**- Based on keyword entered by the service requester the list of best fit web services was generated. For our case, we have entered 'weather' as keyword and the three sample web services were fetched as shown in Figure 3.

   On selection of wsdl results option, two web services were listed that filters the web service that doesn't have syntactically correct wsdl. The fuzzy details related to each web service can be displayed.

5. **Ranking**- Based on the fuzzy calculations as described in section 3.2.6, the web services were ranked and displayed. For simulation, we have considered two input QoS parameters namely (Availability and Throughput).

   The input fuzzy set membership function (Bad, Good, Best) is a triangle form. The range of the input is given below:

   Bad: = {(0, 1) (60, 1) (75, 0)};
   Good: = {(60, 0) (75, 1) (90, 0)};
   Best: = {(75, 0) (100, 1)};

   For output, the same range has been applied. For defuzzification, we have used "Center of Gravity" method. The overall QoS results, depicted in Figure 4 (a-c) and Service requester feedbacks (in context to good, bad) were listed.

6. **Web Service Feedback**- Service requester can provide feedback for the respective web services.

7. **Binding**- We have developed a web application to bind the highest ranked weather web service for fetching the weather information based on city and country.

5. CONCLUSION & FUTURE SCOPE

We have developed an automated end-to-end solution to be deployed at broker side that provides a common framework for service registration, wsdl validation, QoS measurement, discovery and selecting the highest ranked web service using multi-agent system. The solution can be used for intranet as well as internet environment.

Currently, we have considered only two QoS parameters namely availability and throughput for implementation. In future the other domain specific Geospatial QoS parameters namely: *accuracy* of geospatial data, *resolution*, *completeness*, and *data types* also be considered.

References from Books:

[1] Web services: Principles and technology - by Papazoglou, Michael P.; Harlow, Pearson Education.

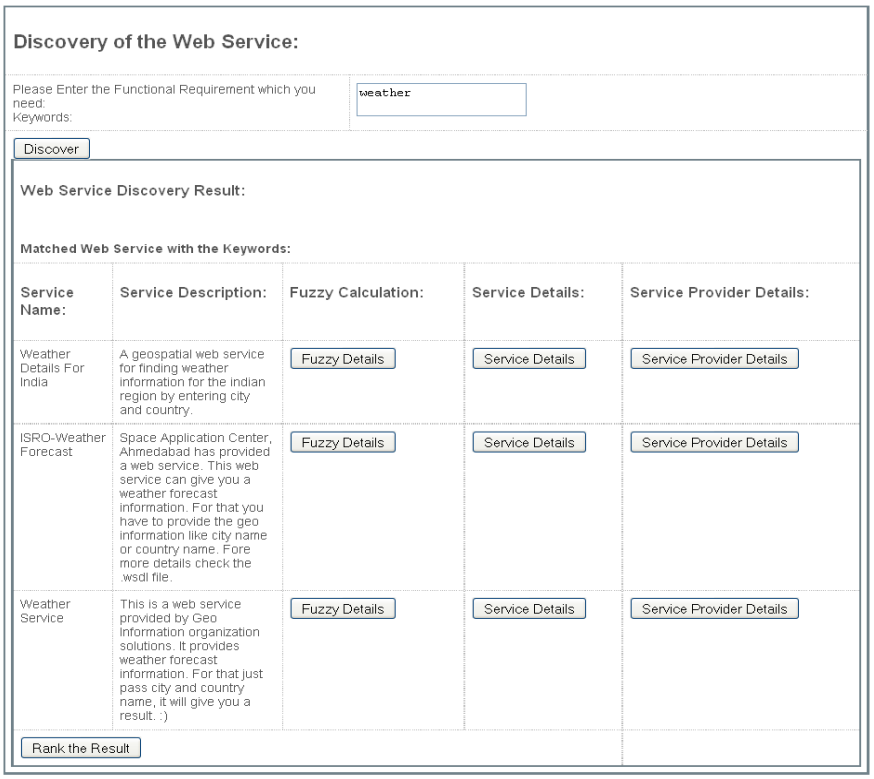
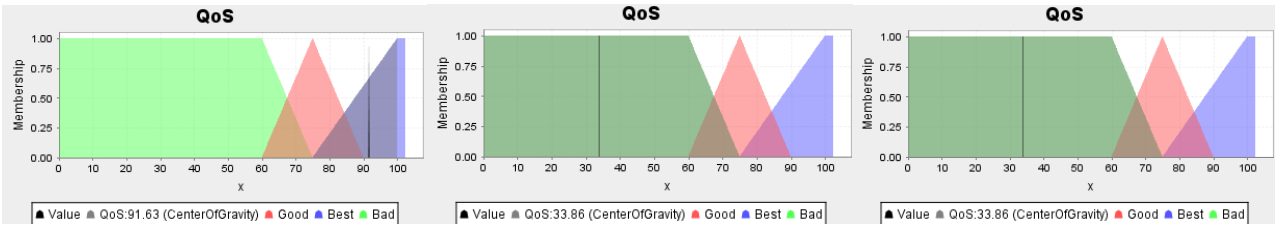


Figure 3. AMAQ System – Discovery & Selection GUI



(a) QoS Result=91.63 (b) QoS Result=33.86 (c) QoS Result=33.86

Figure 4 (a-c). QoS Results of three Sample Web Services

References from Other Literature:

[2] Caglayan, Alper, and Colin Harrison, Agent Sourcebook, John Wiley & Sons, New York, 1997.

[3] Shuping Ran, "A Model for Web Services Discovery with QoS." Published in newsletter, ACM SIGecom Exchanges. Volume 4 Issue 1, Spring, 2003. ACM New York, NY, USA.

[4] A. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui, "A QoS broker based architecture for efficient web services selection", IEEE International Conference on Web Services, 2005, Issue Date: 11-15 July 2005 pp. 113 - 120 vol.1.

[5] Ahmed Al-Moayed, Bernhard Hollunder, "Quality of Service Attributes in Web Services." 2010 Fifth International Conference on Software Engineering Advances (ICSEA), Issue Date: 22-27 Aug. 2010, pp. 367 - 372.

[6] M. Mathes, S. Heinzl, and B. Freisleben, "WSTemporalPolicy: A WS-Policy extension for describing service properties with time constraints". 32nd Annual IEEE Int. Computer Software and Applications COMPSAC '08, 2008, pp. 1180–1186.

[7] D. Z. G. Garcia and M. B. F. de Toledo; "Semantics-enriched QoS policies for web service interactions." in WebMedia '06: Proceedings of the 12th Brazilian Symposium on Multimedia and the web. ACM, 2006, pp. 35–44.

[8] S.M. Babamir, M.R. Shishehchi,S. karimi, "A Broker-Based Architecture for Quality-Driven Web Services Composition", IEEE 2010.

Acknowledgements

Authors thank Shri. Santanu Chowdhury, Deputy Director, Signal & Image Processing (SIPA), Space Applications Centre (SAC), for the constant support and keen interest towards this work. They also thank the internal reviewers for suggesting improvements to an earlier version of this paper. They also thank Shri. A. S. Kiran Kumar, Director, Space Applications Centre for permitting the presentation of this paper during XXII Congress of International Society for Photogrammetry and Remote Sensing (ISPRS-2012) to be held at Melbourne, Australia during August 25- September 1, 2012.