

Differentiating Web Service Offerings

Halina Kaminski, Khalid Sherdil, Hanan Lutfiyya, Nazim H. Madhavji, and Mark Perry
Computer Science Department
University of Western Ontario, Canada
{hkaminsk, ksherdil, hanan, madhavji, markp@csd.uwo.ca}

Abstract

The advent of Service Oriented Architecture (SOA) paradigm and increasing use of Web Services (WS) implies that the future will see a large number of services transferred between providers and consumers, using many applications or agents working on behalf of humans. Discovering and using the services is the easy part. Negotiating and selecting the best services from amongst the plethora of similar ones, depending on their cost and quality, is the challenging issue. However, existing WS-I standards neither cater to provision of Service Level Agreements (SLAs), nor their exchange between parties. These standards are confined merely to WS description (WSDL). Once WS are discovered and selected, SLAs are merely used to monitor service compliance. We propose a novel method that allows service-providers to dynamically generate the SLAs, and then transfer them to clients for selection amongst competitive service providers. The clients use Application to Application (A2A) communication to choose the best service provider at run time, and then bind to it to available services. Our method complies with all WS-I standards, and hence does not require any modifications to the UDDI or WSDL. Instead of using the SLA as just a contractual document for compliance monitoring of the service, we also use it as a means of service selection. We demonstrate and validate our method using a prototype developed in laboratory settings, which uses multiple ‘Weather Service Providers’ to obtain various indicators for weather forecasting.

1. Introduction

The term *Service-Oriented Architecture (SOA)* expresses a software architectural concept that defines the use of services to support the requirements of software users. Each service provides a set of functions, defining a well-defined functional interface, used to invoke the service. Each service is defined using a description language. Applications are a composition of services. In the SOA model, small pieces of functionality are published, consumed and combined over a network. A service can be discovered and accessed. Although SOA is a concept that is independent of any set of technologies, web services (WS) is a popular technology used to implement and deploy an SLO. The use of the WWW for SOAs is referred to *Application to Application (A2A)* communication. Properties of services include the following: (i) A service should be reusable; (ii) A service may be provided by more than one service provider; (iii) An application may compose services at run-time. Different executions may result in different services from different service providers being used. The discovery and use of services is considered relatively easy. However, determining the best service to use for a specific execution of an application is challenging. In this paper, *best service*, depends on the Quality of Service (QoS) requirements that specify requirements for performance or availability, the resource availability of the host machine on which the service resides, and the cost to the consumer of the service.

A *Service Level Agreement (SLA)* is intended to be a formally written agreement between service providers and their customers. The contents of an SLA may vary for different services, but there are common clauses such as QoS requirements and penalties if QoS requirements are not satisfied. It is not always the case that the QoS requirements provided by a service for a specific consumer is the same for each use of that service by the consumer. In between use of the service by the consumer the resources that were needed to support the QoS requirements for the first invocation of the service may not be available for the second invocation. It may be that there is an increase in the number of consumers and the demand on the system. This suggests that SLAs should be generated at run-time. Each service provider would generate SLAs with terms that the service provider can support. Consumers can then choose the SLA that best satisfies their needs at that time.

This paper describes a novel method that allows service-providers to dynamically generate the SLAs. The consumer uses these SLAs for service selection. The method

complies with all WS-I standards, and hence does not require any modifications to the UDDI or WSDL. The transfer of a SLA between parties is packaged as a service itself, which can then be published on a UDDI registry. We demonstrate and validate our method using a prototype developed in a laboratory setting, which uses multiple ‘Weather Service Providers’ to obtain various indicators for weather forecasting.

The outline of the paper is as follows. Section 2 gives a brief overview of the SOA/WS areas. Section 3 discusses

some related work. Section 4 presents our Weather WS as an example application. Section 5 describes the prototype implementation. This is followed by discussion and conclusion in sections 6 and 7 respectively.

2. Background

SLAs specify the QoS requirements and other contractual obligations. A good SLA addresses the following five key aspects of an effective agreement [1]:

- What the provider is promising
- How the provider will deliver on those promises e.g., QoS requirements.
- Who will measure delivery, and how
- What happens if provider fails to deliver as promised e.g., penalties
- How the SLA will change over time

An example of a specification language for SLAs is WSLA [2]. Such specifications can be monitored by the service provider, the service consumer, or a third party. A WSLA specification contains the QoS parameters on which the service provider and consumer may negotiate or agree upon. These may include attributes used in QoS requirements such as the response time, availability/downtime (%), or even the cost per service invocation (\$) [3].

A service description (e.g. WSDL) defines a WS interface, its complementary WSLA specification defines the agreed-upon performance characteristics and the way to evaluate and measure them. Typically a WSLA is a predefined static document, manually configured, and meant for monitoring and compliance of contractual obligations.

WSLA has a structure that allows for QoS expression. Simply saying, QoS statements specify the desired performance while WSLA presents the best possible performance that can be provided. A WSLA specification also provides input to the measurement and management system responsible for monitoring the compliance of an SLA. There is no agreement on what is the most appropriate form of representation for QoS specifications. This may also vary from application to application. The tendency in existing work is to allow specification of bounds on quantitative QoS parameters [4]. WSLA has elements that go beyond QoS specifications, such as the cost, the third party involvement, and the time period. Following are the descriptions of the five major sections of a WSLA, though they can be more complex when the vendor provides multiple services such as networking, online databases or end user direct support [5]:

Parties: consisting of the signatory parties and supporting parties, their names, addresses and contacts

Service Description: specifies the characteristics of the service and its observable parameters. Examples of such WSLA parameters are “service availability”, “service throughput”, or “service response time”. WSLA parameters are composed of (composite) Metrics, which, in turn, aggregate one or more other metrics, according to a measurement directive or a function

Measurement Directives: specify how an individual metric can be accessed. Typical examples of measurement directives are the uniform resource identifier of a hosted

computer program, a protocol message, or the command for invoking scripts or compiled computer programs

Expressions: specify how a composite metric is computed. Examples of expressions are formulas of arbitrary length containing average, sum, minimum, maximum, and various other arithmetic operators. For every expression, an evaluation period is specified

Obligations: define various guarantees and constraints that may be imposed on the WSLA parameters. The validity period is specified which indicates the time intervals for which a given WSLA parameter is valid. The predicate specifies the threshold and the comparison operator (>, <, =, etc.). The result of the predicate is either “true” or “false”. Actions, finally, are triggered whenever a predicate evaluates to “true,” i.e., a violation of a guarantee has occurred. Examples of Actions are “sending an event to zero or more signatory and supporting parties”, “opening a problem dialog or trouble report”, “payment of penalty”, or “payment of premium.”

3. Related Work

One of the key factors of the service provider’s success is the QoS. It determines the service usability and utility. QoS over Internet has additional challenges due to latter’s dynamic and unpredictable nature. Several research projects try to address the problem of describing QoS in WS in an easy way to offer and manage the services. One of the approaches to providing a description of QoS in a standard form is Web Service Offerings Language (WSOL) [6, 7]. It gives the formal language specification of various constraints, management statements, and classes of service for WS. An alternative approach is another XML-based language SLAng [8]. The authors argue that SLAng provides a format for the negotiation of QoS properties and the means to capture these properties for inclusion in contractual agreements. However, there is a substantial limitation to SLAng caused by its predefined format where the definitions of QoS metrics are built into the schema. The SLAng schema has to be modified to add new metrics. Another approach, WS-QoS [9], attempts to address the specification, selection, and monitoring of QoS for WS. WS-QoS is defined as an XML schema which provides flexibility and makes it easy to be modified. IBM has proposed WS-Agreement with certain automated templates [10] that can be used to aid in creation of WSLA documents. However, all of the above approaches are merely notations to describe the QoS parameters and to monitor them. They neither deal with dynamic selection of WS nor comply to WS-I standards.

A service offering in WSOL is a description of one class of a service within a WS. Hence a WS can be associated with multiple service offerings. For example, a web banking service might offer financial transactions service and base the offerings on the price that the customer is willing to pay. These offerings might have different Throughputs, such as 1000 transactions/sec for a higher price or 200 transactions/sec for a considerably lower price. IBM also proposes a framework for providing differentiated levels of WS to different customers (e.g., Gold, Silver, Bronze) on the basis of responsiveness, availability and performance [11]. This concept of WS on Demand [12] uses automated SLAs, but does not cater to their dynamic transfer between parties.

Another interesting approach suggests extending UDDI to support for the QoS specification. UDDIe – an extension that provides for user defined parameters, allows for using simple Boolean expressions in querying for a service, and supports WS discovery and selection [13]. UDDIe enables some simple specification of QoS and cost for WS, but it has very limited capabilities. The QoS description of the parameters that UDDIe supports, is not detailed enough to be used by service management tools. Quality Requirements Language (QRL) [14] does not limit its use to WS, and hence is very complex. Both QRL and UDDIe do not specify who is responsible for measuring QoS. In addition they do not support different levels of performance for the same service.

4. Weather Web Services

With the increasing effects of devastation caused by nature, such as by the recent massive Katrina hurricane, or even by smaller Tornados, the measurement and forecasting of weather is growing in importance. Whereas the forecasting uses techniques from Artificial Intelligence (AI), the measurement and calculation can be aided by WS. The long term climatic effects such as the El Nino or the Global Warming, effect entire economies due to their impacts on tourism industry, recreational and entertainment life, agriculture sector, etc. IT is already playing an important role in weather forecasting.

The Satellites use Remote Sensing technology, which too is related to the field of Geographical Information Systems (GIS) and IT, but we use the concepts from the Surface and Atmospheric sensors, because they are ideally suited for WS. At the grass root level, various indicators or metrics for weather forecast are collected using numerous small sensors, distributed over wide geographic terrains. Some such weather indicators and their sensors are listed below [15, 16]:

Wind Velocity

- Wind Wane
- Doppler Radar
- Anemometer
- Lidar
- Aerovane

Precipitation (Rainfall & Snow)

- Rain Gauge
- Infrared Vapor Radiation
- Microwave Radars

Pressure

- Aneroid Barometer

Temperature

- Thermistor
- Radiometer

Humidity

- Psychrometer
- Hygrometers

These sensors normally sample data rapidly, and get multiple readings every second making human measurement and collection is tedious and error prone. Therefore Analog-to-Digital Converters (ADC) are used to digitize the values and store them in databases. In the process, these data values have to be carried over (normally a wireless) network to the central database. The WS can be used to get these values not only from the database, but also directly from the sensors. The cost of the web service would then accordingly depend on where the readings are acquired, and how quickly they could be obtained from the sensors.

Whereas we normally get weather services from databases of service providers such as Yahoo! (<http://weather.yahoo.com/>), we can also get them from the ground weather stations or directly from the sensor devices. The sensor may be mobile and ad-hoc, as in the Radiosonde balloon, forming a MANET (Mobile Ad-hoc Network). This is analogous to mobile WS running from PDAs or cell phones [17], for which Application Servers with SOAP Engines need to be developed on J2ME environment running on the KVM. As the world becomes more pervasive, there will be an increasing need for

devices to share services with each other, and negotiate the WSLA automatically amongst themselves.

4.1 Weather Prototype Overview

Using the concepts of Weather Web Services, we developed a prototype application under laboratory settings to demonstrate how services offered from different weather service providers could be dynamically selected. Our system consists of three fictitious weather service providers:

- Dopler
- Climatic
- Meteorologic

In real life scenario, these service providers could be located in different geographical locations around the globe. Each provider offers different weather indicators. We presume that there is one service consumer, which would like to select the best possible service for the following three indicators:

- Temperature
- Humidity
- Wind velocity

A Graphical User Interface (GUI) for the client side application, showing these three indicators, was developed. The readings obtained, in this case, are from the Dopler Weather Service, which has been selected by the client from amongst the three service providers. However, this client side GUI merely shows the final display in our application. What is more important is the process of selection of the service provider, which is discussed below.

4.2 Service Deployment & Publication

The service providers package their Temperature, Humidity and Wind Velocity services in the form of Web Services, and deploy them on Application Servers. The description of the services along with the address of the server is then used to generate the WSDL document. Each of the three service providers publishes its WSDL on the UDDI (We used IBM's UDDI Test Registry, which later was terminated by IBM (on 12/01/2006 – <http://www-306.ibm.com/software/solutions/webservices/uddi/>). Once published, any client in the world could search the UDDI by Organization name (e.g., Dopler), or Service Name (e.g., Weather). This search can be done by any application at run time, and typically retrieves a set of pre-specified number of rows. However, in our prototype, we are presuming that this search is done at Development time, and a set of service providers offering the same services (3 in our case) is pre-selected. This selection is done on basis of the *type* of service provided, and *not* on the basis of QoS.

As described in this paper, one novel aspect of our application is the encapsulation of the WSLA document as a WS itself. Hence the WSDL document published on the UDDI also contains information about the service termed as 'getWSLA().' Using this service, the clients may obtain, at run time, the latest version of the dynamically generated WSLA document from each service provider.

The following is an example of WSLA:

```
<?xml version="1.0" ?>
- <!-- Climatic Weather Systems Inc. SLA for GetHumidity()
Provides a Web Service for forecasting temperature -->
- <SLA xmlns="http://www.ibm.com/wsla"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/wsla c:\Projects\WSLA\wsla.xsd"
name="GetHumidity">
- <Parties>
- <ServiceProvider name="Climatic Systems">
- <Contact>
  <POBox>Hurontario Street</POBox>
```

```

    <City>Toronto, Ontario, Canada</City>
  </Contact>
  - <Action xsi:type="WSDLSOAPActionDescriptionType" name="Notification"
partyName="provider">
    <WSDLFile>Notification.wsdl</WSDLFile>
    <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
    <SOAPOperationName>Notify</SOAPOperationName>
  </Action>
</ServiceProvider>
- <Obligations>
- <ServiceLevelObjective name="g1" serviceObject="WSDLSOAPGetHumidity">
  <Obligated>provider</Obligated>
- <Validity>
  <StartDate>2005-11-30:1400</StartDate>
  <EndDate>2005-12-30:1400</EndDate>
</Validity>
- <Expression>
- <Predicate xsi:type="wsla:Less">
  <SLAParameter>responseTime</SLAParameter>
  <Value>463</Value>
</Predicate>
</Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
.
.
  </Obligations>
</SLA>

```

4.3 Dynamic WSLAs

WSLAs are typically meant to be negotiated and agreed upon prior to the running of B2B applications. In fact, they are usually the *end product* of the negotiation process, and represent a form of a contract for monitoring and compliance purposes. Once the QoS parameters are negotiated and agreed upon between parties, these parameters are manually entered in a tool which generates a static WSLA document. This WSLA is then deployed on the system, and typically a third party is asked to monitor its compliance.

As per WSLA specification document [2], a fixed (non-negotiable) WSLA can be published on some Registry for all other clients. In this way, any client in the world could discover WSLA through the corresponding WSDL document. However, the UDDI Registry doesn't cater to the publishing of WSLAs, and hence this approach would fail in the domain of WS. Our method uses the standard UDDI, and packages the WSLA as a service itself, to be offered to any client globally. Any consumer application could, at run time, find some service provider application along with its latest version of WSLA, and then decide on whether or not to use the service offerings depending upon the parameters obtained from the WSLA. In order to maximize the functionality of WSLA, we have placed the performance expressions in a service description using WSLA language. The file contains the specifications of bounds of parameters and it serves as a commitment for the QoS. By having the WSLA placed within a WS as a service itself, and made available to be retrieved before signing up for other services within the WS, allows the client to evaluate and choose the most appropriate service for his/her needs. Our prototype application demonstrates this concept by allowing a client to discover n different weather service providers, obtain their prevailing WSLAs, and then select the best prevailing WSLA, without any human involvement. Note that the process of negotiation is absent here. This is because the OASIS (<http://www.oasis-open.org/home/index.php>) consortium is currently still working on defining a generic electronic document negotiation protocol.

Our prototype application dynamically generates WSLA documents. Under our laboratory settings, we used some random dynamically changing values for QoS parameters, since we could not use actual system parameters, such as the number of transactions per second (or Throughput, normally in hundreds). However, it is expected that in real industrial settings, these values could be obtained from various System Load Monitors. On basis of the actual Throughput, the peak Response Time, the average

Down-time, etc., the WSLA document can be generated, committing to the consumers the expected QoS. The cost of the service could be a function of these parameters, proportionately decreasing if the QoS worsens due to increased system load. Whenever a new client initiates a session with the service provider, and requests for the WSLA document, our prototype generates WSLA at run-time with sample values of QoS parameters and returns it to the client.

4.4 Dynamic Service Provider Selection

Upon receiving the WSLAs from each service provider, the client can then select the best service provider based on the values of QoS parameters. In our prototype, the client application parses each WSLA document received, to extract the QoS parameters.

The WSLAs obtained from each of the three service providers are shown in the three scroll boxes. The more user-friendly JTree view is also available. These WSLAs are then parsed, and the values of the QoS parameters (Cost, Response Time, Availability) are extracted and displayed for each service provider. Based on the weights provided by the user for these parameters, a total score and ranking is computed for each service provider.

The ranking algorithm may be as complex as one using decision trees or operations research, or as simple as the one as follows:

$$\text{Score} = (W_1 * \text{Availability}) - (W_2 * \text{Cost}) - (W_3 * \text{Response})$$

The user may then manually, or the application may automatically, select the highest ranked service provider, bind to its weather services, and obtain the results for selected cities.

5.0 Implementation

The project was implemented under the J2EE v1.4 environment using Rational Software Architect/Rational Application Developer (RSA/RAD) Integrated Development Environment (IDE), running the Eclipse 3.1 workbench. IBM's Web Sphere was used as the Application Server. The application was developed by a team of four graduate students, and had a size of over 5000 LOC (all metrics were calculated using 'Dependency Finder' at <http://depfind.sourceforge.net/>). We now discuss various implementation approaches pertaining to the parsing, storage, creation and transportation of the WSLAs using Java's APIs for XML (JAX) that come packaged in Sun's Web Services Developer Pack (WSDP).

5.1 Parsing WSLAs

WSDP is a toolkit that allows developers to create, test and deploy WS, XML applications and Java applications with the latest Java technologies (at <http://www.developers.net/>). The WSDP is included in Sun's Java Enterprise System (SJES), best run on Solaris 10 operating system. The WSDP provides developers a suite of API for WS, such as JAXP, JAXB, JAXR, JAX-RPC, SAAJ, etc. These APIs conform to WS standards including SOAP, WSDL, and UDDI.

Our prototype extensively used the JAXP, which has the following advantages [18]:

- Programs can be written independent of the underlying XML processor with the JAXP API
- The underlying XML processor can be replaced without changing a single line of application code

We used the light-weight SAX (Simple API for XML) parser for streaming data and then converted the data to the more complex Document Object Model (DOM) tree, which could then be used for any kind of XML data processing. SAX is a Java native, parser independent, event based API for processing XML documents. SAX handles XML information as a single stream of data. The data stream is unidirectional; hence previously accessed data cannot be re-read. SAX is read-only and as such it can't be used to change a document. However, only a small portion of the document is processed at a

time. Thus, SAX uses less memory compared to a fully constructed DOM tree. SAX was designed around abstract interfaces rather than concrete classes, so it can be layered on top of other parser's existing native APIs. Hence DOM parsers can use SAX to build the internal tree data structures [18].

5.2 Storing WSLAs

By storage we mean the internal data structure representation of the XML content. The DOM is an abstract data structure that represents XML documents as tree nodes. The DOM parser interface is called the 'DocumentBuilder.' A parser takes the serialized representation of the XML document and creates an in-memory graph of nodes. For large documents, DOM can become very heavy. However, it is useful when data from nodes might be needed or modified later.

In addition to JAXP, we also initially used JDOM [20], which is a unique Java toolkit for working with XML, provided to enable development of XML applications. Its design embraces the Java language from syntax to semantics.

JDOM provides a light-weight java centric means of accessing an XML document within Java through a tree structure. As opposed to DOM in which a tree has only nodes, in JDOM an XML element is an instance of Element, an XML attribute is an instance of Attribute, and an XML document itself is an instance of Document. Hence its easy to create any of these without configuring any factory interfaces. Since JDOM performs quickly in small-memory, it incorporates the best of DOM and SAX.

Our prototype eventually used DOM instead of JDOM because the latter doesn't support Parsed Character (PC) data. Though our WSLA documents didn't have any PC data, the XML standards have provision for the same. For future scalability and extensibility purposes, we wanted to develop a tool which could handle *all* the XML documents and not just a sub-set of them.

5.3 Creating WSLAs

There are several ways of creating a WSLA dynamically. One way is to read an initial WSLA static text document, parse it, and develop its DOM tree. The parameter nodes of this tree could then be changed to represent the actual system values. However, this can be slow, particularly when the generated WSLA has to be supplied back to the consumer at runtime. We used a faster approach by using SAAJ (SOAP with Attachments API for Java) in which a SOAP message can be created in RAM, by adding its Header element, Envelope element, and subsequently each Node element. This process is hard-coded directly in the executable and hence is much faster.

5.4 Transporting WSLAs

The WSLAs can be despatched to the consumer in the form of an abstract data structure called 'Document.' This data structure can be sent as an Attachment with the SOAP message using SAAJ. But then the client application should be aware of the description of the 'Document' interface. Hence, for complete interoperability, we decided to despatch the WSLA as a textual string within the SOAP envelope. Interestingly, since WSLA and SOAP are both in XML, we are in effect wrapping an XML document within an XML SOAP envelope. In doing so, one has to be careful because such scenarios often cause problems since the XML metadata (tags) have to act as simple data as well.

6. Discussion

This paper suggests only a simple approach in dynamically selecting WS using WSLAs under the A2A paradigm. We expect that more advanced approaches would have to be used as the nature of services becomes mission critical. The field of WS is only in its infancy, and as new standards are developed, they would facilitate this process. Artificial Intelligence (AI) and other techniques would be needed, possibly involving software mobile agents, for negotiation amongst parties. The clients would also like to monitor the QoS parameters for compliance, such as the response time, to ensure that other WSLA obligations are fulfilled. This requires an extensive monitoring and billing infrastructure, which is detailed in [6]. Despite all these extensions, the foundation framework involving

the WS protocol stack would remain the same, because the very success of the WS framework lies not in any new technological advancement but in world wide acceptance to standards.

Our work describes in depth several implementation approaches, down to the level of data structures, interfaces, and parsing of streaming data, which could be used to efficiently handle the entire process even in real time. Further, we do not go beyond the existing standards, and use the widely accepted java technologies, for implementing our prototype. Hence we are confident that the external validity of our method is high. Our prototype can be extended for exploring various other areas as well, such as the emerging field of the Grid Computing.

The WS can be effectively used to implement the Global Grid Forum's (<http://www.gridforum.org/>) Open Grid Service Architecture (OGSA) capabilities. But Grids require stateful computing, whereas the WS are stateless. Hence the relationship between WS and stateful resources is needed. One approach is to use the 'Implied Resource Pattern' in which the messages to a WS may include a WS-Resource component that identifies a stateful resource in the execution of the message. Our project only involves the stateless WS in which there is merely a single invocation (e.g., `getTemperature`), resulting in complete information transfer from service provider to receiver. As per *one* definition of Grid computing, if in conducting a task, we use services from two different machines on different administrative domains, then it forms a Grid. But if the domain is the same, then it forms a Cluster. Our extended our prototype to make parallel calls to different WS in different domains (Climatic, Doppler and Metereologic). In this way the results are likely to arrive faster than three sequential calls to the same company's server. We used two types of concurrent mechanisms available in Java, namely the Threads and Events. The Threads are basic units of execution, and divide the processor time into concurrent processes, each accessing a different WS almost simultaneously. Event notifications are asynchronous messages, sent from Event Dispatchers to those Event Listeners which are Registered with them (e.g., upon press of a 'Submit' button). Hence they follow the Observer Pattern presented in [19]. By doing so, we demonstrated the extensibility of our work to Grid, though we were not able to validate our results by real industrial data measurements of gain in response time and efficiency. Just as in case of the Grid, our work can be used to explore other areas in the field of M2M computing.

7. Conclusion & Future Work

It is likely that there will be billions of WS invoked daily, through millions of machines or agents, working on behalf of humans. WS availability, reliability, and scalability continue to be a center of the SOA research. Finding an easy method to describe and publish the service offerings remains a goal for researches. In this paper we present a unique approach to the service offering techniques. We suggest that packaging WSLA files that describe QoS parameters, as services, for each service offering, would allow the customer to evaluate and choose the best service available. It was our goal to show that the discovery phase combined with WSLA file-retrieval can bring some benefits to the customers and the potential for the service providers to better advertise the service. Our approach works within the existing WS-I framework, without requiring any changes to the UDDI or WSDL. We hope that the framework presented will create the base for further studies in QoS and in the area of dynamic WS selection.

For future work, we plan to test our prototype on application servers in different cities/domains, using different tools for simulating and measuring heavy load on the servers, and using actual measurements for QoS parameters. We also intend to explore the Stateful Grid Computing using WS. Towards the legal side, we are also working on evolving the WSLA. Finally, we are also working on modeling the service provider selection process using Business Process Execution Language for WS (BPEL4WS).

Acknowledgements

Support from the Natural Science and Engineering Research Council. We thank graduate students Kevin Dane McGregor and Hany El-Yamany for their contribution in developing the prototype.

References

1. Edward Wustenhoff. "Service Level Agreement in the Data Center," Sun BluePrints OnLine, April 2002, Sun Microsystems, <http://www.sun.com/blueprints/0402/sla.pdf> (accessed on 19 May 2006)
2. Heiko Ludwig, et. al. "Web Service Level Agreement (WSLA) Language Specification, v1.0" IBM Corporation, Watson Research Center, <http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf> (accessed on 19 May 2006)
3. Arun Nagarajan, Anbazhagan Mani. "Understanding quality of Service for Web Services," January 2002, <http://www-128.ibm.com/developerworks/library/ws-quality.html?n-ws-1172>, (accessed on 12 December 2005)
4. Glen Dobson. "Quality of Service in Object-Oriented Architectures," <http://digs.sourceforge.net/papers/qos.html> (accessed 20 December 2005)
5. Leopoldi, R. "IT Services Management, A Description of Service Level Agreements," White Paper, RL Consulting, 2002
6. Vladimir Tomic. "Service Offerings for XML Web Services and their Management Applications," Ph.D. Thesis Dissertation, Ottawa-Carlton Institute for Electrical and Computer Engineering, Carlton University, August 2004
7. Tomic, V., Lutfiyya, H., Tang, Y., "On Requirements for Management of Mobile XML Web Services and a Corresponding Management System," 7th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services, TELSIKS '05, IEEE 2005
8. Lamanna D. D., Skene J., Emmerich W., "SLAng: A language for defining Service Level Agreements", The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03), San Juan, Puerto Rico, May 2003
9. Tian, M., Gramm, A., Naumowicz, T., Ritter, H., Jchiller, J. "A Concept for QoS Integration in Web Services," Proc. of the 1st Int. WS Quality Wkshp. -WQW 2003, 4th Int. Conf. on Web Information Systems Engineering – WISE 2003 Rome, Italy, Dec.'03, pp 29-36
10. Heiko Ludwig, et. al. "Template-Based Automated Service Provisioning – Supporting the Agreement-Driven Service Life-Cycle," IBM Research Report RC23660, July 8, 2005, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598
11. Dan, A., Ludwig, H., Pacifici, G., "WS differentiation with SLAs," posted May 2003, <http://www-128.ibm.com/developerworks/webservices/library/ws-slafram/> (accessed on 19 May 2006)
12. Asit Dan, et. al., "WS on Demand: WSLA-driven automated management," IBM Systems Journal, Jan. 2004
13. Shaikh, Ali, A., Rana, O. F., Al-Ali, R., Walker, D. W. "UDDIe: An Extended Registry for Web Services. Proc. of 2003 Symp. on Applications and the Internet (SAINT'03), Wkshp. on Services Oriented Computing: Models, Architectures and Applications, Orlando, USA, Jan. 2003. IEEE-CS (2003) pp 85-89
14. Martin-Diaz, O., Ruiz-Cortes, A., Corchuelo, R., Toro, M. "A Framework for Classifying and Comparing Web Services Procurement Platforms. Proc. of the 1st Int. WS Quality Wkshp. - WQW 2003, WISE 2003 Rome, Italy, Dec 2003 pp. 37-46
15. Greg O'Hare, John Sweeney, Rob Wilby, "Weather, Climate, and Climate Change: Human Perspectives," Pearson/Prentice Hall, 2005, New York, USA
16. C. Donald Ahrens, "Meteorology Today: An Introduction to Weather, Climate, and the Environment," 4th ed., West Publishing Company, St. Paul, MN, USA
17. Gehlen, G., Pham, L. "Mobile Web Services for Peer-to-Peer Applications," Second IEEE Consumer Communications and Networking Conference, CCNC'05, pp. 427-433, IEEE 2005
18. Bodoff Stephanie, Armstrong Eric, Jennifer Ball, Debbie Bode Carson, Ian Evans, Dale Green, Kim Haase, Eric Jendrock. "The J2EE Tutorial," Second Edition, Sun Microsystems, 2005
19. Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John. "Design Patterns – Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995
20. Biggs Wes, Evans Harry. "Simplifying XML programming with JDOM," <http://www-128.ibm.com/developerworks/java/library/j-jdom/> (accessed on 19 May 2006)