Quality of Service for Web Services

SAEED ARABAN and LEON STERLING
Department of Computer Science and Software Engineering
The University of Melbourne, Melbourne 3010
AUSTRALIA

Abstract: - The World Wide Web is evolving from being a pure information repository to a more functional and service oriented platform thanks to technologies such as Web Services. This technology offers a homogeneous representation of Web elements and the ways they are communicating that make it possible to deal with the inherent structural and behavioural heterogeneities of the Web. A Web service can be seen as an autonomous functional element that is loosely coupled to other Web services and can be discovered and deployed in Web-based applications. Autonomity and loose coupling make Web services a viable light weight complementary component-based approach for design and development of dynamic distributed systems for more heavy weight solutions such as OMG's CORBA and Microsoft's DCOM. In this paper, we take the position that if Web Services are going to be considered as reusable commercial of-the-shelf (COTS) components, their Quality of Service (QoS) needs to be expressed explicitly and measured independently. More specifically, we present and discuss possible quality aspects that need to be represented and quantified for Web Services.

Keywords: Web Services, Quality of Service, Metrics, Component Reuse.

1. Introduction

The Word Wide Web (WWW) is evolving from its original role as a pure information repository for mainly human users to an increasingly functional platform for cross-platform application-to-application (A2A) interoperation and business-to-business (B2B) communication. Web Services technology is a relatively recent approach aiming at providing a hemogenous interface layer around fundamentally distributed, heterogeneous and dynamic functionalities in the WWW environment. In this model, new Web-based applications and services are built by mixing and matching existing services. This means great opportunities for large-scale black-box Component of the Shelf (COTS) reuse and a market for value-added services.

A Web service can be viewed as an abstraction for an autonomous web-based component that performs a well defined function. A service is represented by its interface that encapsulates a collection of network enabled operations along with all the interaction details, such as message formats (that details the operations), communication protocol and location, using XML based notation. The interface also hides all the implementation details (e.g. programming language, software or hardware platform, etc.) of the service. This model encourages distributed systems based on loosely coupled and cross-platform reusable components. However, developing systems with predictable and measurable level of reliability, performance and security is not a trivial task in this model.

Using a Web Service by a client usually means a longer-term dependency to a Service Provider. Thus, if the Web Services model is going to be adopted as the main stream design and development model for web-based A2A and B2B business critical applications, it needs to go beyond just specifying and providing functionalities and address issues related to the quality of the service. For example:
How can a Service Requestor be sure that a Web Service that matches his functional requirements will be available and responsive?
How can a Service Requestor know what to expect from the service in terms of its performance and scalability?
How a client can trust that the service is operated in a secure environment, that the Service Provider guards his data and does not allow it to be made available thoughtlessly or maliciously to any third party?
How easy is the service to use?

The issue here is when entering into any form of relationship or agreement; the parties cannot assume that the other party is implicitly to be trusted to satisfy the expectations.
Thus, we need a way of determining beforehand whether a candidate Web Service can satisfy our

functional and quality requirements. Ideally, this information should take the form of a specification as part of the service interface that tells us *what* the service provide without entering into the details of *how*. Further, the specification should provide parameters against which the Web Service can be verified and validated, thus providing a kind of *contract* between the Web Service and its clients.

This paper focuses on the quality aspects of the Web Services. Obviously, the quality of Web Service-based systems is directly affected by the Quality of Service (QoS) of their underlaying Web Services. Here, we take the position that quality attributes of Web Services (e.g. performance, reliability and security) are vital factors in making or breaking of a service in a competitive market of services. Thus, they should be defined, quantified and represented as part of the service interface in such a way that they can be understood by human and/or software requestors (clients). This way, the quality of the overall system may be measured and controlled.

The rest of this paper is organised as follow. Section 2 presents a brief introduction to Web Services model and its elements. Section 3 establishes a metric framework for quality of Web Services. In section 4 we draw conclusions and future work direction.

2. The Web Services Model

The Web Services architecture covers three elements: roles, operations and artifacts
[3]. Figure 1 illustrates these elements.

The architecture is based upon the interaction of three roles: Service Provider, Service Registry and Service Requestor. The interactions involve the Publish, Find and Bind operations. Together, these roles and operations act upon the Web Services artifacts: the Web Service software module, its description (also including a description of the Service Provider) and the Client Application.

The actors in the Web Services model are as follows. Their roles can be described from a business or an architectural perspective:
Service Provider:
From a business perspective, this is the owner of the service.
From an architectural perspective, this is the platform that provides access to the service.
Service Requestor:
From a business perspective, this is the business that requires certain functions that are covered by the corresponding service.
From an architectural perspective, this is the client application that is looking for and then invoking the service.
Service Registry:
From a business perspective, this is the owner of a registry service.
From an architectural perspective, this is a platform that provides access to registered service information.

The Service Registry is a searchable repository of service descriptions. Service Providers can publish their service descriptions, including a description of the provider's business. Service Requestors can find services and obtain binding information (from the service descriptions). This information is used during development for static binding or during execution for dynamic binding. Dynamically bound Service Requestors access the Service Registry on every execution. For statically bound Service Requestors, access to the Service Registry is performed only one time to get the information. On static binding, the role of the Service Registry is even optional – the Service Requestor could obtain the service description directly from the Service Provider or from other sources besides a Service Registry (e.g. WWW site or FTP site).

The Web Services model comprises the following operations:

Publish: To make it possible for a Service Requestor to find a Web Service and access it, the Service Provider has to publish it to a Service Registry.

Find: In the find operation, the Service Requestor retrieves a service description by inquiring the Service Registry. The find operation can occur in two different lifecycle phases for the Service Requestor: at design time to retrieve the service interface description for the client application development (static binding), or at runtime to get the service's binding and location description for invocation (dynamic binding).

Bind: In the bind operation, the Service Requestor invokes the service at runtime using the binding details in the service description to locate, contact and invoke the service.

The artifacts in the Web Services model are the objects that are produced and dealt with in the context of Web Services. These are as follows:

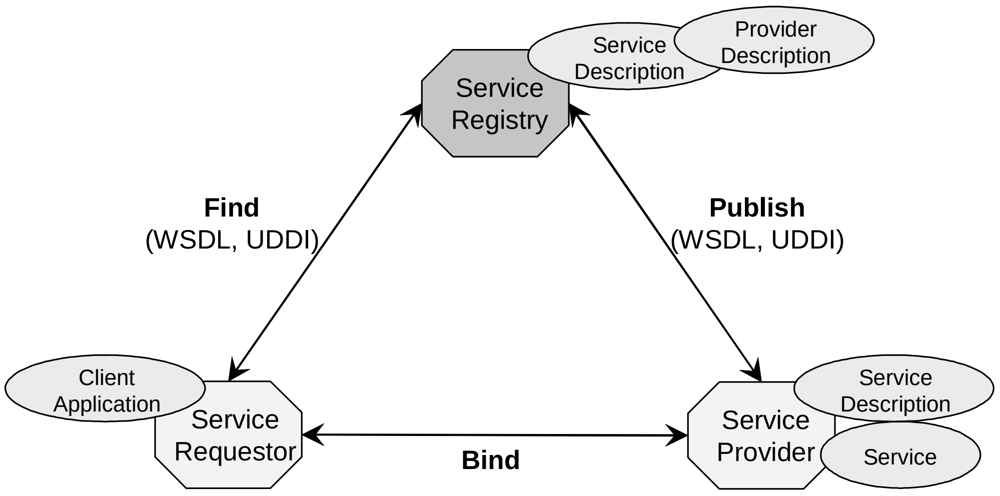

Figure 1: The Web Services Model.

Service: This is the implementation of a software module deployed on a network accessible platform provided by the Service Provider to be invoked by a Service Requestor.

Service Description: The service description contains the details of the interface and implementation of the service. The *service interface* description comprises information about the operations provided by a service, as well as their parameters. It can be compared with the signature of a method. Additionally, the protocol used for communication with the Web Service is described. The *service implementation* description contains information about the location where the service is exposed, i.e. the network address of the endpoint providing the service. The complete service description is published to a Service Registry by the Service Provider to make the service accessible to Service Requestors. It includes the service's data types, operations, binding information and network location, as provided by the service interface and implementation descriptions. It could also include information about the Service Provider, service and provider categorization and other metadata to facilitate discovery and utilization by the Service Requestor.

Client Application: This is the application implemented by the Service Requestor that uses the functionality of the Web Service and invokes it at runtime.

The Web Services framework relies on emerging XML-based technologies, such as Web Services Definition Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description, Discovery and Integration (UDDI), to provide an open, flexible and extensible environment for representation, communication and integration of the Web Services.

Web Services allow applications to be integrated at a higher level in the protocol stack, based more on service semantics and less on network protocol semantics, thus enabling loose integration of business functions [3]. These characteristics are ideal for connecting business functions across the Web both between enterprises and within enterprises. They provide a unifying programming model so that application integration inside and outside the enterprise can be done with a common approach, leveraging a common infrastructure.

The integration and application of Web Services can be done in an incremental manner, using existing languages and platforms and by adopting existing legacy applications. Moreover, Web Services compliment Java2 Enterprise Edition (J2EE), Common Object Request Broker Architecture (CORBA) and other standards for integration with more tightly coupled distributed and non-distributed applications. Web Services are a technology for deploying and providing access to business functions over the Web; J2EE, CORBA and other standards are technologies for implementing Web Services.
2.1 Contract Aware Web Services
As mentioned in the introduction, explicit contracts between the service requestors and the service providers can clarify the obligations and benefits of each party; this is the principle of design by contract [4]. Beugnard et al. [2] defins four levels of contract in a component-based software development

environment, which we believe are also valid in the context of Web Services model.

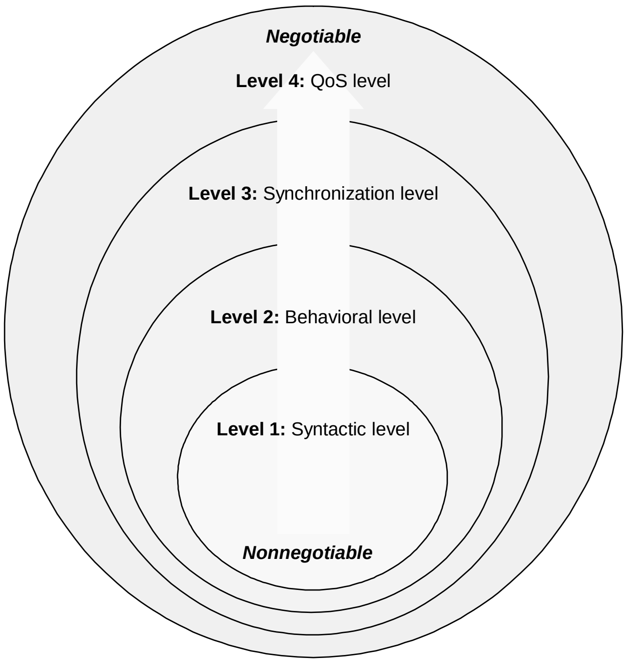

Figure 2: The four contract levels.

The four contract levels are: basic or syntactic, behavioral, synchronization, and quantitative (see Figure 2). At the basic level, the input and output parameters of a service and their types are defined and may be verified. At the behavioral level, behavior of a service may be specified using pre- and post-conditions. A synchronization contract specifies the dependencies between services of a component, such as sequence, parallelism, or shuffle. A QoS contract may be negotiated statically or dynamically between a service provider and service requestor.

We believe that Web Services must be made contract aware. In other words, they must be able to support contracts at all levels. However, to the best of knowledge, there is no XML standard for supporting contracts above the basic level. Thus, there is a need for such standard if XML-based technologies, such as Web Services, are going to be more effective.

3. Quality of Service Model and Measures

In this section we discuss quality attributes relevant to the QoS of a Web Service. We also provide definitions and explore possible metrics for those quality attributes.

Before defining any QoS metrics, we need to identify their application context. One of the measurement pitfalls is rushing to measure what is convenient or easy to measure rather than measuring what is needed or relevant. Such metrics often fail because the resulting data is not useful or relevant to their audience and what they need [5]. A measurement can be more relevant and successful if it is designed with the goals of its target audience in mind.

The Goal Question Metrics (GQM) is a framework for deriving relevant metrics according to the goals of our measurement [6, 7]. We use this framework to set out a context for our quality metrics. In order to apply this framework, we need to explicitly define our goals first. Then, we should ask questions about what do we need to know in order to achieve our goal. Finally, we need to decide about the metrics that may answer our questions.

There are templates for the GQM that can help to identify the goals and ask the right questions [7]. Applying some of those templates, we identified the following goals for our measurement program:
Ability to compare QoS for different Web Services that provide similar service (requestor’s point of view).

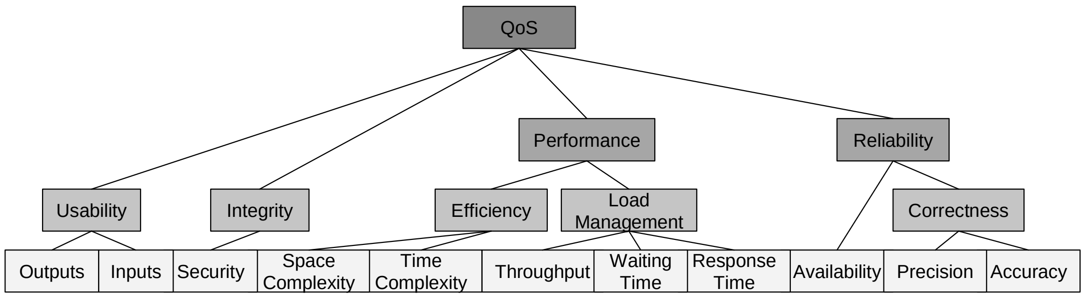

Figure 3: FCM hierarchy for QoS of the Web Services.

Ability to improve the QoS (provider's point of view).
Ability to search and match Web Services based on QoS requirements (requestor and discovery mechanism point of view).

All the above goals can be summarised to just one simple goal:

Evaluating Quality of Service for Web Services

Having this goal in mind, we ask the following questions that may lead us to relevant quality factors:
What are the quality factors for a Web Service?
How can the quality of a Web Service be improved?
To what degree does a Web Service match the quality requirements?

In order to answer the above questions, both *internal* and *external* attributes of Web Services must be considered. Internal attributes of a service are those that can be measured purely based on the service itself regardless of its environment (e.g. time complexity of the algorithm used and coupling to other services). On the other hand, external attributes can only be measured with regards to the service environment and its behaviour (e.g. throughput, response time). Table 1 summarises the quality factors and attributes for Web Services relevant to QoS.

Generally, Service Requestors are more interested in external attributes of Web Services. However, external attributes are more difficult to measure than internal ones. A connection between internal attribute values and external attribute values is widely assumed by the software engineers [Brooks and Yourdon]. There are few models for relating higher-level external qualities to lower-level and usually easier to understand and measure internal attributes [8, 9]. Here, we use McCall's Factor Criteria Metric (FCM) like quality model. Figure 3 is a hierarchy of these quality attributes based on the FCM quality model.

QoS Factors	Internal Attributes	External Attributes
Reliability	*Correctness* (Accuracy, Precision)	Availability Consistency
Performance	*Efficiency* Time Complexity Space Complexity	*Load Management* Response Time, Waiting Time, Throughput
Integrity		Security
Usability	Inputs and Output	

Table 1: Quality of Service factors and attributes for the Web Services.

Now, we need to define the above quality factors and attributes:
Reliability: The ability of a Web Service to perform its required function for a given period of time. A simple measure of reliability is mean-time-between-failure (MTBF): MTBF = MTTF + MTTR, where MTTF is the mean-time-to-failure and MTTR is the mean-time-to-recovery [10]. However, a more sophisticated view may include the following attributes:
Correctness: A Web Service must satisfy its specification and fulfils the requestor's mission objectives. This is an internal attribute of a service, which in turn may depend on:
Accuracy: The difference between the service's result and the actual value. Accuracy of a result may be affected by the algorithm used to implement the service and/or by the temporal characteristics of the service. For example, accuracy of a foreign exchange calculator service may be affected by temporal factors such as sampling frequency and/or volatility of the foreign exchange market.
Precision: The mathematical precision (number of decimal places) of the result.
Availability: The probability that a Web Service being available for operating according to requirements at a given point in time. It can be measured as: (MTTF/

MTTF + MTTR)*100%, in this form, availability measure is somewhat more sensitive to MTTR [10, 11].

Consistency: The results of different implementations of a Web Service (specification) should not be very different in value or effect.

Performance: It may be measured in terms of internal efficiency of a service implementation and/or the efficiency of its environment:

Internal Efficiency: The amount of computing resources required by a service implementation to perform its function. This can be measured as time and space complexity using bigO notation.

Environment Efficiency: Ability of the provider platform to balance the load on the computing resources () according to resource requirements and/or demand for services offered, in such a way that maximises the overall performance of the system, which may be measured in terms of *Response Time, Waiting Time* and *Throughput.*

Integrity: Extent to which access to Web Service or data by unauthorized persons is controlled. *Threat* is the probability (which can be estimated or derived from empirical evidence) that an attack of a specific type will occur within a given time. *Security* is the probability (which can be estimated or derived from empirical evidence) that the attack of a specific type will be repelled. The integrity of a system can then be defined as:

$$integrity = \sum [1 - (\text{threat} \times (1 - \text{security}))]$$

where threat and security are summed over each type of attack [12].

Usability: Effort required for learning, operating, preparing input, and interpreting output of a Web service [13].

Some of the above metrics may be subjective; however, having subjective metrics is better than no metrics at all, as long as they are measured consistently.

4. Conclusions and Future work

.In this paper we argued that if Web Services is going to be the preferred model for A2A and B2B interaction and collaboration, it should be able to support all levels of software contracts. That means, quantifiable quality attributes of the services must be visible to the Service Requestor from the service interface. We also presented a quality of service model that defines quality criteria and related metrics and their relation to higher level quality factors.

The next step is to develop and evaluate actual metrics for measuring the quality attributes within the quality of the service model. Also, the actual XML design needs to be defined for representing the metrics as part of the service description language (WSDL). Another possible research direction is exploring ways of discovering and matching Web services according to the QoS requirements.

References:

1. Snell, J., *Implementing Web Services*, IBM Web Services: IBM Web Services Toolkit V2.3, Documentation.
2. Beugnard, A.J., J.-M.; Plouzeau, N.; Watkins, D., *Making components contract aware.* IEEE Computer, July 1999. **32**(7): p. 38-45.
3. Kreger, H., *Web Services Conceptual Architecture (WSCA 1.0).* 2001, IBM Software Group: IBM Web Services.
4. Meyer, B., *Object-Oriented Software Construction*. 1997: Prentice-Hall.
5. Fenton, N.F. and S.L. Pfleeger, *Software Metrics: A Regorous & Practical Approach*. 2nd ed. 1996: International Thomson Computer Press.
6. Basili, V.R. and D.Weiss, *A methodology for collecting valid software engineering data.* IEEE Transactions on Software Engineering, 1984. **10**(6): p. 728-738.
7. Basili, V.R. and H.D. Rombach, *The TAME project: Towards improvement-oriented software environments.* IEEE Transactions on Software Engineering, 1988. **14**(6): p. 758-773.
8. McCall, J.A., P.K. Richards, and G.F. Walters, *Factors in Software Quality*. 1977, US Rome Air Development Centre.
9. Boehm, B.W., et al., *Characteristics of Software Quality*. Software Technology. 1978, Amsterdam: TRW.
10. Littlewood, B., *Forecasting Software Reliability*, in *Software Reliability: Modelling and Identification*, S. Bittanti, Editor. 1989, Springer-Verlag. p. 141-209.
11. Rook, J., *Software Reliability Handbook*. 1990: Elsevier.
12. Gilb, T., *Principles of Software Project Management*. 1988: Addison-Wesley.
13. Pressman, R.S., *Software Engineering: A Practitioner's Approach*. 5th ed. 2001: McGraw-Hill.