An architecture for building Web Services with Quality-of-Service features

Julio Fernández Vilas, José Pazos Arias, Ana Fernández Vilas

Dep.. of Telematic Engineering, University of Vigo,
Campus Universitario s/n, 36200 Vigo, Spain
Tel. +34 986 813 868
{jfvilas, jose, avilas}@det.uvigo.es

Abstract. As the web services technology become more fashionable, we will meet with the need for systems able to guarantee a given level of quality of service (QoS). This need is caused by factors like response time, availability, or throughput, which are variables completely unpredictable when using Internet. In addition, when a client can use multiple providers, we need a system to automatically qualify, select, and use a concrete provider. This paper will show a new technique that allows creating web services with QoS features. This new technique is based on the virtualization of the real web services used to serve the client requests. What we propose is to create new *virtual* web services that will be the ones invoked by the clients. On the server side, the *real* web services are invoked in order to accomplish the primary invocation.

1 Introduction

Web services are not a mature technology nowadays, so problems can arise which can make you lose control on some critical aspects of your applications, such as QoS, ACID properties [1], availability, etc. In order to provide a solution that can solve these problems, we have defined an architecture for the web services technology based on virtualization techniques. Apart from other capabilities that are summarized in section 6, and our recent contribution to high availability in web services architecture [2], the proposed architecture enables implementing any QoS model based on user-defined metrics.

After this introduction, the remainder of the paper is structured as follows. Section 2 introduces virtualization technique. An implementation of the proposed QoS architecture is discussed in section 3. Section 4 hold a specific discussions about the qualification and evaluation processes.

2 Virtualization

Using virtualization, one or more web services can be grouped inside a unique wrapper, which is then published as a standard web service, so clients use the new

virtual web service as a standard web service, that is, there is no difference between real and virtual from the client's point of view. Wrapping a group of web services has nothing to do with the well-known web service wrappers, which are software components used to isolate the communication layer (SOAP, HTTP…) of the web services from the web services implementation. The wrapper term is used to refer to a virtual view of a set of web services, that is, clients have a unique view of that group. The virtual view is in fact published as a standard WSDL document.

Virtualizing a web service requires a change in the standard web services architecture, since a virtual web service must reside in an intermediate element different from the client and the provider.

2.1 Architectural change

There is a big dependence between clients and servers, originated in SOAP messages. A simple change in the name or the type of a parameter will cause a change in the way SOAP messages are produced. The invocations will not run properly if client proxies and client applications remain unchanged, due to the use of different SOAP messaging schemas. From this point of view, clients and servers are strongly coupled (like a method or a function inside a program). This strong coupling is caused by the nature of invocations. In the standard web services architecture, clients use "direct invocations" to invoke web services. What we propose here is to change the architecture, moving the invocation model **from direct to indirect**. We also suggest the use of intermediate elements inside the standard architecture (as proposed in [3] and [4]) that can receive, process, and re-route SOAP messages.

2.2 VWS Components

Our proposed architecture for the use of Virtual Web Services (VWS) determines the need of three components: a **client** (the one who needs a service), a **service provider** (the one that offers and publishes a web service), and a **VWS engine**. In our VWS architecture, invocations are not performed directly from the client to the provider, but rather the client performs an invocation to the VWS engine, who acts as an intermediary, and the engine performs an invocation to the provider.

The VWS engine can be something as simple as adding some kind of decision capabilities to the client proxies (Fig. 1). Alternatively, it can be something as complex as a dedicated server (Fig. 2). This document explains how to deploy web services with QoS features using a complex VWS engine.

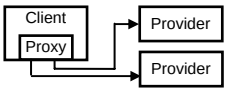

Fig. **1**. Simple VWS Engine.

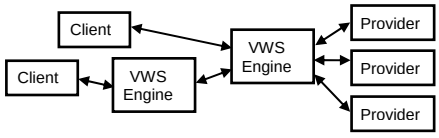

Fig. **2**. Complex VWS Engine.

2.3 Implementing Virtualization

Virtualization will lead us to use a new kind of services: **virtual web services** (VWS). Any application that is able to use a standard web service can be bound to a virtual

web service. Our virtualization technology provides a definition language (an XML-based language). This language (VWSDL, VWS Definition Language) is used to write VWS documents. Clients **do not use** VWS documents, since these documents are just a definition of an implementation of a virtual service, and this definition is only useful to VWS engines. VWS documents are used to describe virtual web services, and they must contain, at least, a list of methods provided by the service. In addition, for each method published inside a service, the input and output parameters, and their corresponding data types must be specified.

A method definition also contains a list of web services and methods that can be invoked in order to complete the execution of a method. All those web services and their methods represent in fact the implementation of a virtual method.

3 QoS implementation

For the VWS engine to select the best provider on each moment, we need to use systems that gather statistical information about providers and its services. Our virtualization architecture proposes the use of three elements that enable the selection of the most suitable service and provider: (1) a qualifying method for each one of the services and providers; (2) a way to obtain historical qualifications of previous invocations; and (3) an element in charge of computing the qualifications and perform an evaluation using the information gathered previously.

A correct operation of the assign-and-compute-qualifications system depends directly on choosing the appropriate variables, like availability, reliability, or performance [5]. Variables can be either variables that define the quality of invocations (variables associated to a web service), or variables associated to a provider (that must be taken into account before and after an invocation).

The election of the right units for the variables is a mandatory step before using a qualification system, so a process to convert qualitative variables into quantitative variables is needed. In addition, mathematical functions should be applied in order to have all variables within the same scale and using the same units.

3.1 Expressions

When a client application needs to invoke a service, it has to be decided which would be the most appropriate provider. What our VWS architecture proposes is to build an expression that combines all we know about a given web service by using variables, like in expression (1).

$$0.6*A + 0.3*C + 0.1*S \quad (1)$$

where A is the availability, C is the cost, and S is the security level. To be able to write expressions in which we can mix different types of variables, having each one its own unit (milliseconds, dollars…), we must use conversion functions in order to represent all the variables within the same scale. In addition, we should keep in mind the fact that some variables maximize the value of the whole expression by means of their maximum values (availability, for instance), while other variables do the same with their minimum values (cost, for instance). In order to unify scales, ranges and

units for all variables used in an expression, functions `adjust` and `reverseAdjust` can be used. Expression (1) can be rewritten as expression (2) using these new functions.

$$0.6 * adjust(A) + 0.3 * reverseAdjust(C) + 0.1 * adjust(S) \quad (2)$$

Three additional methods will allow writing complex expressions: complex variables, time variables, and external variables. A **complex variable** is a new variable that is created using a combination of simple ones. In expression (3), a variable defined as G is used to hold a value that represents a global score for a provider, using A, C, and S. **Time-variables** allow writing expressions that involve present and past values. For example, expression (4) can be used to obtain a global qualification for a provider, where time is taken into account by using two global-score values (G_{i-1} and G_i) (a weighting factor (W) is used as a stabilization mechanism).

$$G = 0.6 * adjust(A) + 0.3 * reverseAdjust(C) + 0.1 * adjust(S) \quad (3)$$

$$adjust\ ((\ W*G_{i-1} + G_i\ )\ /\ (W + 1)) \quad (4)$$

Our virtualization architecture does not impose any kind of constraint when writing expressions or using variables, and so we can use **external variables** managed by an external entity. We could write an expression (5) that includes an external status variable (T) indicating the status of the web service. If a web service is "out-of-service", we can set T to zero, and the VWS engine will stop selecting it.

$$T\ *(\ 0.4*reverseAdjust(C) + 0.6*adjust(A)\ ) \quad (5)$$

We propose the use of two additional mechanisms in order to give the clients a chance to drive the evaluation process performed at the VWS engine. The first mechanism is based on the use of **classes of service** (CoS). A CoS is created by writing a concrete expression inside a method definition. More than one CoS can be defined, identifying each of them with a name. Once several classes of service have been defined inside a VWS document, clients can specify what CoS to use in each invocation by using SOAP extensions.

The second extension mechanism allows the client applications to **prune** the list of eligible web services. Clients can send, attached to the SOAP requests, a set of conditions that implementation web services must fulfill. This way, when the VWS engine has to perform an evaluation, it will consider only those services that fulfill client's conditions.

3.2 Qualification and Evaluation

Variables are used inside our solution to achieve the automation of two processes: a qualification process and an evaluation process. This automation is achieved by using expressions. Every time a web service is invoked, the VWS engine should update the statistical data related to that service.

The process of updating the statistical database must be performed in an automated way, and so the variables used in the process should be immediately computed after a service invocation. Actually, variables that cannot be computed from a service invocation are usually **provider-related** variables, and not **service-related** ones. Provider-related variables should be considered as **external variables**.

Provider-related variables can be either quantitative or qualitative, so the same conversion process used in service-related variables can be applied to provider-related ones. The evaluation process of a provider-related variable can be performed the same as the service-related variables evaluation. Actually, both types of variables can be used in the same expressions. However, the qualification of a provider cannot be performed in an automatic way, because the values of provider-related variables can be unknown during the lifecycle of an invocation. From the engine's point of view, **there is no difference between one type of variables and the other**.

5 Related Work

Several works on QoS have been introduced [6, 7], and there are also software companies working on developing infrastructures that can provide QoS features [8, 9]. All these research lines are centered in defining metrics that can be used to qualify service providers. What we have shown here is not a set of metrics or a way to qualify a web service or a provider. What we propose is an architecture that can be used to implement QoS that does not enforce the use of a concrete collection of metrics. The people who build the VWS documents have the chance to decide what variables they want to use. They must only write the appropriate formulae inside the VWS documents and provide the necessary data to evaluate the expressions.

A significant amount of works on QoS are focused on defining variables related to the operational aspects of the web services, that is, reliability, performance, response time, etc. All of them are variables related to the implementation and execution of a web service. In contrast, our solution allows using any kind of variable, which can be a web-service-related variable or a provider-related one (delivering time, package quality…)

Some approaches to the QoS problem [7] do extend standard languages used in the current web services architecture. On the other hand, one of the main features of our VWS-based architecture is its **compatibility** with existing and future web services standards, since our approach do not add any language to the standard architecture.

6 Conclusions and Further work

Our virtualization architecture does not impose any kind of restrictions to a QoS model: the overall performance of the proposed architecture will depend on the variables and formulae used to build the system. It has to be noted that the VWS engine introduces a new overhead inside the execution architecture, since requests must be received and re-routed to the appropriate provider. However, this overhead is insignificant when compared to the benefits obtained with our QoS implementation architecture, since the main goal of our purpose is to guarantee a **given level of quality of service** in invocations. The VWS technology is the base for other works that extend the use of our model. Regarding these other features of our model:

- We can use the VWS documents to build composite web services. We are defining a collection of different types of invocations. Our goal is to develop a web services programming language that supports basic programming structures (if-then-else, while-do, etc.) and advanced issues like callback, or asynchronism.
- Virtualization can also be used to build highly available web services by building clusters of web services. The construction of these clusters is based on the use of VWS documents. This work is described in [2].
- Our proposed architecture also provides a way to implement SLA support on standard web services architecture. Purposes like [10] (where measurements must be taken on the client or on the provider) or like [11] (where a centric measurement element cannot be implemented using the actual architecture), can be implemented using our intermediary-based architecture.
- Unlike BPEL4WS [12], WSCI [13], or BPML[14], we propose VWSDL and a new architecture as a mechanism to give the web services technology new features like QoS, high availability, etc., not only process management.

References

1. A. Elmagarmid, "Database Transaction Models for Advanced Applications", *Morgan Kaufmann*, 1992
2. J. Fernandez, J. Pazos and A. Fernandez, "High availability wit clusters of Web Services", *Proceedings of the APWeb'04*, April 2004, pp. 644-653
3. D. Booth et al., "Web Services Architecture", *W3C Working Draft*, 2003, http://www.w3.org/TR/ws-arch
4. D. Fensel, C. Bussler, "The Web Services Modelling Framework WSMF", *1st meeting of the Semantic Web enabled Web Services workgroup*, 2002
5. D. A. Menascé, "QoS Issues in Web Services", *IEEE Internet Computing*, vol. 6, no. 6, Nov./Dec. 2002, pp. 72-75
6. A. Mani and A. Nagarajan "Understanding quality of service for Web services", *IBM dW*, 2002, http://www-106.ibm.com/developerworks/library/ws-quality.html
7. S. Weller, "Web services qualification", *IBM Developer Works*, 2002, http://www-106.ibm.com/developerworks/library/ws-qual/
8. "QoS for Web Services Defined", *Santra Technologies*, 2002, http://www.santra.com/knowledge/?id=qos
9. "High Availability with QoS", *IBM and CISCO*, 2000, http://www-1.ibm.com/servers/eserver/zseries/library/specsheets/high_availability_qos.html
10. Sahai, A. et al., "Automated SLA Monitoring for Web Services", *13th IFIP/IEEE Intl. Workshop on Distributed Systems (DSOM 2002)*, pp. 28-41
11. Ludwig, H. et al., "Web Service Level Agreement (WSLA) Language Specification", *IBM*, August 2002
12. Curbera, F. et al., "Business Process Execution Language for WS", 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/,
13. Arkin, A. et al., "Web Service Choreography Interface 1.0", *BEA, Intalio, SAP & Sun*, 2002, http://ftpna2.bea.com/pub/downloads/wsci-spec-10.pdf
14. Arkin A. et al., "Business Process Modeling Language (BPML) specification", *BPMI*, June 2002, http://www.bpmi.org/bpml-spec.esp