# VWS: Applying virtualization techniques to Web Services

*Julio Fernandez Vilas, Jose Pazos Arias and Ana Fernandez Vilas*

Department of Telematic Engineering, University of Vigo, Vigo, Spain

**Summary**
What web services can offer nowadays is insufficient to build complex applications based on web services. Several technological aspects have not been standardized yet, and the study of some of them is still at a very early stage. This paper presents a new technique that can be applied to web services technology in order to be able to build web services with features like QoS, high availability, proxy/firewall usage or SLA management, among others. This technique is based on the virtualization of the real web services used to serve the client requests, creating new virtual web services that will be the ones invoked by the clients. At the back-end, the implementation web services (the real ones) will be invoked in order to fulfill the primary invocation.

***Key words:***
*Web services, virtualization, QoS, high availability.*

## Introduction

At this moment, we can easily use web services to integrate different systems, that is, as an enterprise application integration (EAI) tool. Integration is the main area where web services are being broadly used, due fundamentally to the high level of acceptance of the standards around web services technology.

Nowadays, as web services are not a mature technology, some problems arise that can result in loss of control on some critical aspects of applications, such as QoS, availability, error management, multi-provider etc. These new problems, which are impossible to obviate, prevent web services technology from being massively adopted.

Throughout this paper, we will analyze the lacks of the current web services architecture that impede the total exploitation of the technology, and we will present an alternative architecture which is fully compatible with the current one (in fact, it has been designed to coexist with it). This new architecture allows building web services that support a range of necessities (availability, multiple providers, quality of service, etc.) not covered by the current architecture in a standard way.

The remainder of the paper is structured as follows. Section 2 analyzes some problems of the current architecture. Section 3 introduces the virtualization technique, whose implementation is discussed in section 4, together with other related issues. Several uses of the virtualization technique are commented in section 5, while section 6 holds a specific discussion about the different ways the architecture can be implemented. Finally, section 7 discusses related work in this area, and section 8 presents our conclusions and future work.

Manuscript received May 5, 2006.
Manuscript revised May 25, 2006.

## 2 Current Situation

In the basic architecture proposed by the W3C, we can differentiate three main roles: client, provider and discovery agency. Two new elements were introduced in the last revision of the architecture [1]: human beings representing the client and the provider, referred to as requester human and provider human, respectively. These humans are responsible for agreeing on the service's semantics and negotiating its description. Eventually, after reaching an agreement, the provider entity will publish a WSDL document containing the description of the service, which will be used by the client entity to perform a bind process.

This architecture is divided into four layers: communications, messages, service descriptions and processes. This division is actually a way to organize the technological needs of web services for them to become a complete technology (reliability, transactionality, quality of service, availability, etc.), as long as all the infrastructure and the languages needed to support application development in the same conditions as other equivalent technologies (DCOM [2] or EJB [3]).

The following subsections analyze several characteristics and problems of web services within the current architecture.

### 2.1 Roles in Standard Architecture

The current architecture has several problems that stem from the limitation in the definition of roles:

- Dynamic binding cannot be done in a standard way, because the binding process must be carried out at design-time, not being it possible to perform it at runtime. A possible way to perform dynamic bind is based on the use of metadata [4], which impede the natural use of web services

- When a client is bound to a provider, the execution of the client application remains bound to the web service of the service provider for the whole client application's life. If the client decides to change his provider, he will find the necessity of binding a new web service to the client application. This would imply re-developing code, performing binding again and adapting the client application's logic.
- To use more than one provider of the same functionally equivalent web service inside an application, every web service must be bound individually to the client application. Additionally, the code needed to handle the invocations and its errors must also be developed.

This kind of problems arise from the non-distinction of two clearly distinguishable roles: "service provider" and "web service provider". For example, a financial broker is a service provider (he provides a trading service in a stock market). Such a provider can offer his services through the telephone, through a network of branch offices or through a web service. In the latter case, the provider of the web service need not be the service provider.

## 2.2 Invocation Model

In the web services technology, a unique invocation model is considered: direct invocation. According to this model, client applications invoke web services **directly**, in such a way that if any of the provider's components (network, applications server, database server, etc.) misbehaves, the invocation will fail. As a result, the technology has a limited availability, because requests cannot be re-routed to alternative web service providers. In addition, controlling quality of service and service level agreements is responsibility of the client or the provider, due to the absence of an intermediate element in charge of it.

A solution to these problems can be achieved by changing the invocation model from direct to **indirect**, by means of introducing an intermediate element in the invocations. This idea is being used in some commercial initiatives, such as IBM's Web Services Gateway [5], Xtradyne's WS-DBC [6] or EntireX XML Mediator [7] from Software AG. All of them are characterized by the use of proprietary and single-purpose solutions designed to solve specific problems (security, message transformation…). The use of intermediaries is also reflected in the last architecture proposed by the W3C.

## 2.3 Asynchronous Invocations

Another lack of the current architecture is the possibility to make asynchronous invocations in a simple way. Therefore, when a client makes an invocation, it must wait for the response, which is usually a SOAP message over HTTP. This kind of invocations is not valid to build business processes (also referred to as long-running transactions) whose duration may be long and unpredictable. Although some initiatives exist, like BPEL4WS [8], that provide support for business processes, their asynchrony is achieved through polling mechanisms. Other initiatives, like WS-Callback [9], SOAP Conversation Protocol (SCP) [10] or Asynchronous Web Services Protocol (AWSP) [11] have been specifically designed to support asynchronous invocations with no polling, i.e. the web service replies to the client application as soon as the answer is ready. In this case, the client must be accessible from the Internet, in order to receive the call-back from the web service. On the other hand, these initiatives did not consolidate properly due, possibly, to a lack of institutional support.

## 2.4 Error Control and Error Management

An invocation of a service normally follows this scheme:

- Invoke the service.
- Check the error obtained as a result of the invocation.
- If there has been an error, decide whether the invocation can be retried.
- Retry a finite number of times, if so decided.
- If the invocation cannot be carried out, return an error to the next higher logical level of the client application or to the user if current one is the highest level.

If, in a given moment, a client decides to change a service provider, the application must be modified by deleting the references and data types of the current web service provider to add those of the new one, being it also necessary to rebuild the proxies. If the client aim is to enhance the application with greater functionalities and availability, he must opt for adding more than one web service provider to the client application, which complicates management. The application must be modified for every new provider that is added, leading to the following invocation scheme:

- Invoke the service.
- Check the error obtained as a result of the invocation.
- If there has been an error, decide whether the invocation can be retried.
- Retry a finite number of times, if so decided.

- If the invocation cannot be carried out and more providers are available, retry the invocation with the next one.
- If the invocation has not been possible with any of the providers within predefined interval, return an error to the next higher logical level of the client application or to the user if the current is the highest level.

All this logic for error control and management must be implemented by the developers of the client applications. Moreover, it becomes an ad-hoc programming that must be created for every new provider.

### 2.5 High Availability

Achieving high availability with web services is still an open issue. Nowadays, the solutions being adopted aim at improving the availability of the implementation of the web service, not the availability of the web service as a whole. In other words, they use high-availability systems (whether clusters or fault-tolerant systems) to improve the availability of the components responsible for the implementation of the service. This allows improving the availability from the provider's point of view: if a node of a cluster fails, the web service can continue working at any other node; similarly, if a fault-tolerant system is used, the failure of one of the components will not affect the global operation of the system. The situation is greatly different from the client's point of view, because availability is zero if the invocation of the service fails (network errors, server failures, maintenance tasks, etc.), regardless of whether the provider implements high-availability techniques or not.

These problems arise from the fact of protecting the implementation of the service, and not the elements that client applications use: its interface.

### 2.6 Quality of Service

Features such as a client being able to do invocations specifying a given QoS level, or a provider offering a range of different QoS levels to its clients, are undoubtedly necessary, but not possible nowadays with the current architecture. Even though there exist several works in this direction [12], they have not materialized into implementable proposals and some of them [13, 14] are theoretical studies on the metric systems to use.

The current architecture, with no intermediaries, limits the QoS theories to the QoS capabilities of the underlying protocols (HTTP, TCP/IP). In [15] WS-QoS is introduced as a mechanism to achieve QoS features in web services, based on the use of a "service broker". WS-QoS is a specific proposal to solve the QoS problem.

QoS works in the web services area just care about the metrics of the implementation technology (response time, throughput…), but they do not consider metrics related to provider entities (price, quality, confidence…).

### 2.7 Service Level Agreement

The capability of a system to offer the chance to use service level agreements (SLA) is absolutely determined by the ability of that system to implement QoS techniques. This is because the SLAs must be stated in terms of measurable variables, for which reliable mechanisms are needed in order to provide objective measurements of the necessary metrics. These metrics are closely related to those used when applying QoS models, like response time, availability, security, cost, etc.

At the same time, as we said before, it is necessary to separate the roles of "service provider" and "web service provider", so that service-level agreements can be written considering metrics that reflect separately the operation of the web service itself and the operation of its provider. This need of separation is therefore applied to the quality of service implementation necessities, that is, we must be able to measure the behavior of a web service and of its provider in QoS terms.

There are several works on SLA, but they are only theoretical works about a SLA framework [16, 17] or SLA specification languages [18].

## 3 Virtualization

The technique we propose, namely virtualization, is based on grouping one or more web services inside a unique wrapper, which is then published as a standard web service. Clients use the new virtual web service as a standard one, i.e. there is no difference between real and virtual from the client's point of view. With virtualization, some additional logic can be performed out of the client applications (error management, provider selection, etc.), and this way the software complexity is radically reduced, since developers need only to care about business logic.

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 of the web services (SOAP, HTTP …) from the web services implementation. The wrapper term is used here to refer to a virtual view of a set of web services, so that 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 web services architecture, since a virtual web service must reside in an intermediate element different from the client and the provider.

### 3.1 Architectural Change

Let us consider a client application bound to a specific web service. Let us suppose that the provider modifies the parameters sent/received at the web service. After modifying the business logic, the provider should rebuild the wrapper classes and the WSDL document. A simple change in the name, the type or the namespace 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, since SOAP messages sent from clients to servers and back do not follow the same schema. 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, changing the invocation model from direct to indirect. We also suggest the use of intermediate elements inside the standard architecture (as proposed in [1]) to receive, process, and re-route SOAP messages.

The architecture we propose is mainly based on the introduction of a **virtualization layer**. This layer must be seen as a non-intrusive element; in other words, its introduction must respect, at least, the following directives:

- It must not alter the current infrastructure, that is, it must coexist with the languages and protocols currently in use.
- It must not have an effect on the web services over which it is built. There must be no need to modify a web service to adapt it to the new architecture; on the contrary, the new architecture must be able to adapt itself to the services existing within the current one.
- It must not affect client applications. It must be possible to use the virtual views of the services used by client applications the same way as real web services.
- It must not alter the way providers operate at present, allowing a web service to comply with the standard architecture and with the new one at the same time.

### 3.2 VWS Components

The architecture we propose for the use of virtual web services (VWS) determines the existence of five elements:

- **Client**: the entity that needs a service. This is equivalent to a client in a SOA structure [19].
- **Delegate client**: the agent on whom the client delegates the responsibility for executing a service. Generally speaking, it represents client applications.
- **Service provider**: the entity that offers a service. It must be seen as a provider in a SOA architecture.
- **Delegate provider**: the agent on whom the provider delegates the responsibility for offering a service. In web services technology, a delegate provider is a web service.
- **Engine or intermediary**: the entity in charge of putting delegate clients and delegate providers in contact. It performs communication processes between both delegates following different algorithms, and pursuing different objectives depending on the runtime environment.

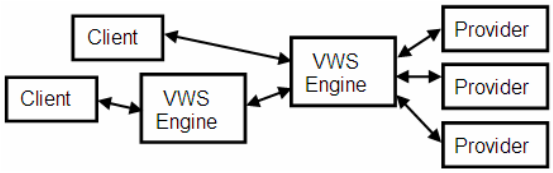


Fig. 1 Complex VWS engine.

A engine inside our proposed architecture can be something as simple as adding some kind of decision capabilities to the client proxies. Alternatively, it can be something as complex as a dedicated server (fig. 1). An engine of this kind is referred to as a VWS engine.

A VWS engine is not a virtual server. It is a standard one, and it should be implemented the same way as a server used to process standard web service invocations. The term "VWS engine" refers to the capability of a server to understand virtual services definitions. That is, it can receive, process, and respond to standard web service invocations, but, in order to process a request, the engine uses VWS descriptions to select and invoke the most suitable web service provider.

## 4 Implementing Virtualization

Virtualization will guide us to the use of a new kind of services: virtual web services (VWS). Any application that is capable of using a standard web service can be bound to a virtual web service. Conversely, any provider component that can be exposed as a standard web service can also be published as a virtual web service. Our virtualization technology provides an XML-based definition language, called VWSDL (VWS Definition Language), for writing VWSDL documents. Clients do not use VWSDL documents, since these documents are just a

definition of an implementation of a virtual service, and that definition is only useful to a VWS engine. Our proposed language has been defined as a stand-alone language, that is, not as a WSDL extension, since it is intended for a distinct use. Even though the main objective of WSDL and VWSDL is the same (describe a web service), the way services are described are completely different.

```
<method name="getPrice" type="equivalent"
        default="COS#1">
  <select name="COS#1"
   expression="0.7*adjust(av)+0.3*adjust(rt)" />

  <select name="COS#2"
   expression="reverseAdjust(cost)" />

    <cache from="21:30:00" to="23:00:00"
           check="data" />
    <cache from="08:00:00" to="15:00:00"
           check="content" />

  <input>
    <parm name="ticker" type="xsd:string"/>
  </input>

  <output>
    <parm name="name" type="xsd:string"/>
    <parm name="value" type="xsd:float"/>
    <parm name="date" type="xsd:date"/>
  </output>

  <invoke id="inv2" method="sendItToMe"
          location="http://a.com/s1.wsdl">
    <mapin name="map1i" type="transform" />
    <mapout name="map1o" type="transform" />
  </invoke>
  <...>
</method>
```

Fig. 2 Example of a virtual method declaration (VWSDL fragment).

VWS documents are used to describe virtual web services, and they must contain, at least, a list of the methods provided by the service (the virtual equivalence of a WSDL portType) using the `method` elements, as shown in fig. 2. In addition, for each method published inside a service, the input and output parameters, and their corresponding data types must be specified (just as with WSDL, a `types` section exists to allow the definition of data structures following a specific schema, using XML Schema Definition, XSD).

`method` elements are also used to specify a list of web services (delegate providers) and methods that can be invoked in order to accomplish the execution of a virtual method. In fact, all those web services and their methods represent the **implementation of a virtual method**, and they can be specified (inside the `method` element) using as many `invoke` elements as needed (fig. 2).

Each `method` element has its own `name` and `type` attributes. The `type` attribute is used to specify the type of implementation that is being defined for the method. For instance, the `sequence` value states that, in order to execute a virtual method, the engine must invoke a set of delegate providers (specified inside `invoke` elements) following a predefined sequence. Other values for the type attribute can be used to specify different engine behaviors (equivalent, alternate, parallel, iterative…).

In order to understand the operation of a VWS engine, let us consider a virtual method defined using the `equivalent` type, which is used to build a virtual method by using a set of "equivalent" providers with the same functionality. At runtime, when an invocation arrives at a VWS engine, the engine must first analyze the request in order to select the corresponding VWSDL document. Then using an expression selected from a `select` element of the document, the engine must evaluate a set of providers (the ones specified inside `invoke` elements) according to that expression. The provider with the highest scoring according to the expression will be selected and invoked (after properly transforming parameter, if needed). If the invocation fails, the engine can select another provider or return a SOAP fault message to a client, depending on the definitions contained inside the `method` element.

## 4.1 Parameter Handling

Parameters in SOAP are nominal: input parameters sent to web services use the name of the elements (XML elements), and the same holds for return parameters. Let us suppose that we build a virtual method that receives a parameter called "P1" and returns a parameter called "RP". This represents a little restriction in the way we write the method and invoke elements inside VWS descriptions, since the name of the parameters used in the virtual method must match the ones used in real methods. If a match can be found, the `invoke` elements can be used without a problem. When such a match cannot be found, a `mapin` element should be used to solve this situation, transforming client's messages according to the schema providers expect to receive. For return parameters a `mapout` element can be used. Input and output mappings can be specified by using those VWSDL elements, which point to a transformation specification (typically described using XSL stylesheets).

The VWS engine should check all type assignments described in the VWS documents, and this type-checking process should be made only once per VWS document: when the document is first deployed to the engine.

The use of the mapping elements brings a lot of functionality to the virtualization technique, since mapping parameters between virtual and real services **lessens the coupling level** between client applications and web services.

## 5 Uses of virtualization

The technique we have just described is the basis for solving the problems described in section 2. One of the most important contributions of our proposal, from a structural point of view, is the **role separation** between service provider and web service provider, which results in the introduction of intermediate elements that allow services to be offered by a kind of entities (service providers), while web services can be offered by a different kind (web services providers, i.e. VWS engines).

One of the immediate uses of virtual web services is the simplification of the error control and management processes. Thanks to the use of virtual services, engines can retry invocations (under certain circumstances) using alternative providers when an invocation fails. This type of error control and subsequent recovery is done transparently to the client, who only has to invoke a standard web service (handled by the engine as a virtual one).

Besides error management, differentiating the service provider from the web service provider allows modifying, removing and adding new service providers for the same virtual web service with no need to change the client applications, thus simplifying the developing process of client applications and provider management.

Long-running processes are no longer a concern for clients, because the engine will deal with the invocation of all the web services needed to complete their execution. For the long-running processes to be controlled from the clients, VWSDL allows defining two ways to notify termination. The first is to set a callback URL which the VWS engine will invoke when a process finishes, either normally or abnormally. Second, for those clients that do not support the facility to receive invocations, VWSDL allows asking for the state of the running processes through a "query web service"; this service should be invoked periodically, as in a typical polling system. The use of VWS engines allows the creation of asynchronous web services that are implemented using synchronous ones, since the engine can deal with two kinds of invocations.

The use of expressions in the `select` elements is the base for constructing systems that support Quality of Service techniques. Two mechanisms are used for that purpose: expressions that allow defining different "classes of service" in the VWS engine (like service levels in [16]), and the selection of classes of service to be used by the client (using SOAP-Header elements). The variables that appear in the expressions can be variables related to the web service provider (response time, availability …), variables related to the service provider (delivery time, service cost …), or independent variables managed by a third party entity. At the same time, variables can be quantitative or qualitative, in which case a numerical valuation process [20] must be performed. Finally, expressions must use variables represented in the same units, ranges and scales, for which several adjustment functions are provided by the expression evaluator module of the VWS engine. In [21] we have dealt with the application of virtualization techniques to QoS problems.

Having a way to measure quality of service is an indispensable prerequisite for the use of SLAs. The VWS engine is a component of the architecture capable of controlling the operation of the services it invokes (response time, cost, …), which allows it to be used at the same time as an element capable of checking the fulfillment of certain objectives regarding providers' utilization. In other words, we can put the VWS engine in charge of revising, in each invocation, the fulfillment of the service-level objectives (SLOs) stated in an SLA. In this way of doing things, the client entities and the provider entities must sign an SLA, and sent the VWS engine a description of it. The engine will be in charge of verifying the fulfillment of the SLA and also of notifying, whenever needed, the fail to fulfill any of the conditions in the agreement to both parts, for them to take appropriate actions. In [17], a theoretical structure for a qualification system based on the use of a rating repository is proposed. This theoretical structure can now be implemented using our virtualization architecture.

If we use expressions to represent the operation of a web service (its response time, its availability, etc.), the VWS engine can be seen as an element that distributes work, that is, the engine would act as a controller node for a cluster of web service providers. This functionality allows constructing clusters of web services, offering high-availability systems where not only the implementation of the service is protected, but also its interface. Clusters built this way allow using heterogeneous nodes (clusters where nodes can have different platform implementations), and also to construct Internet-wide clusters. In [22] we have dealt with the construction of clusters of web services.

## 6 Architecture Implementations

There exist several ways to apply the proposed architecture to the current architecture. Those forms represent different implementations and lead to obtaining different functionalities depending on the localization of the VWS engine. Basically, we can distinguish three alternatives, depending on whether the engine is located: (1) in the private network of the client, (2) in the private network of the service provider, and (3) in the Internet, being accessible to both client and service provider.

In case (1) above, (point 1 in fig. 3), the use of an engine offers the following functionalities:

- It makes the software of the client **independent** of that of the delegate provider, because the engine allows both of them to change their interfaces without affecting the other, as long as the engine can map the different structures of the SOAP messages exchanges in the invocations.
- **Provider independence**. Virtual web services allow adding, modifying and removing providers without affecting the client, that is, with any need for the client to modify its applications.
- **Proxy**. The use of the engine allows client applications to invoke Internet web services even when they have cannot connect to anything beyond its private network.
- **Automated error management**. Because the methods of the virtual services are built from a list of web service providers functionally equivalent, the engine can control the errors that take place and, when a provider fails, try to use another one. This goes on unnoticed for client applications.
- **Cache**. Under certain circumstances (configurable for each method of a virtual service), it is possible to cache client requests, noticeably improving response times. Take as an example the case of a web service that offers share prices on closing of financial markets, or a weather forecast web service. We have developed and published a work in this direction that demonstrates the benefits of such a system in a practical way [23].

An engine placed in the provider's private network (point 2 in fig. 4) fundamentally allows building web services cluster systems, where the VWS engine acts as a controller node for the cluster, in charge of receiving requests and routing them to a certain node that is selected depending on the workload of each node (point 3 in fig. 3). Other possible functionalities are:

- **Firewall**. The VWS engine allows providers residing in a private network to be invoked from outside that network, keeping a high security level inside of it.
- When used as a **cluster controller**, it allows to introduce modifications in a node of the cluster while keeping the others unchanged, making it possible to perform software testing with a minimal impact in the construction of a new web service in case of an error.

Last, in case (3) (Internet engine), its main use is that of a broker, that is, the engine acts like an intermediate component in the network that puts clients and service providers in contact (point 3 in fig. 3), moreover offering the following main functionalities:

- **Decoupling** between delegate client and delegate provider, due to the fact that the definition of virtual services makes it possible to modify the interface of the delegate providers (web services) without changing the client software.
- Use of **multiple providers**.
- **Error control** and management.

Given the level of integration between our proposal and the current architecture, the complexity of the implementations can increase indefinitely. For example, point 4 in fig. 3 represents a broker that uses another broker as a web service provider.

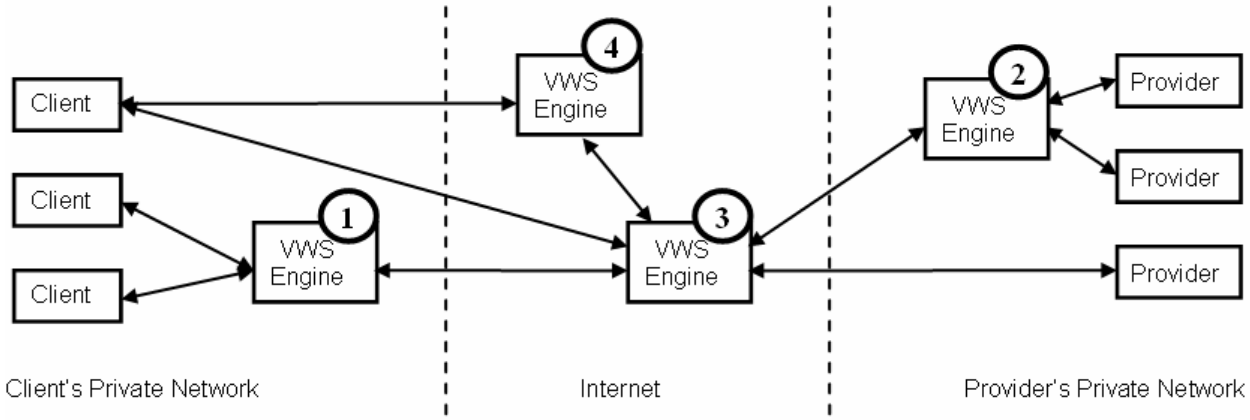


Fig. 3 Sample architecture implementation.

## 7 Related Works

Virtualization is being successfully applied on many other environments, such as storage virtualization, network virtualization and hardware virtualization. What we propose here is to virtualice software, creating new virtual components (VWS) with which we can achieve a degree of decoupling and independence between clients and providers greater than the one we could achieve with standard web services.

The architecture we propose is innovative as a global solution for a range of problems that have only been addressed individually so far. Problems related to SLA management, quality of service or high availability are the subject of study by public and private entities, but the solutions proposed are specific for each one of these issues: architectures and languages to support SLA management [16, 18, 24], metrics for QoS [14, 19] and software and hardware architecture intended to improve the availability of the implementation of web services [25], but not their interface, which is what clients perceive as a web service.

At the same time, the use of intermediate elements (the engines in our proposal) is a technique that is being implemented in some software platforms, but always with a specific use and using proprietary languages and/or systems. For example, WS-DBC [6] uses an intermediate element as a security system, while WS-Gateway [5] isolates the private networks of clients and/or providers, also supporting certain protocol changes (from SOAP to HTTP/POST, for example).

## 8 Conclusions and Further Work

What we propose is to use a common language for the description of virtual web services, which at the same time provides a standard way to construct the interfaces that intermediate elements must offer through standard WSDL documents. We also propose an extension of the standard architecture in order to support VWS in such a way that it be compatible with current architecture.

VWS can help developing web services with rich features like high availability, performance optimization, QoS, error management, etc.

The overall performance of the proposed architecture (whichever its use) will greatly depend on the variables and expressions used for the description of virtual web services. 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 not significant when compared to the benefits obtained with our architecture.

Using VWS developers can build atomic web services that can be published and subsequently consumed by resource-constrained devices like mobile phones or PDA, that is, virtual web services can be used as a personalization mechanism regarding client requirements in order to simplify its use in such device types.

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. This work is in progress, and we are defining a set of different types of invocations. Our goal is to develop a web services programming language (WSPL, as an extension of VWSDL) that supports basic programming structures (if-then-else, do-while, etc.). Its objective is to provide a simple composition method.

- We plan to integrate WSLA with our web service descriptions. This way, a VWS engine can be used to analyze each invocation of a web service and evaluate SLOs after each invocation.

Our proposal is not disruptive in its implementation, because it can coexist with the current architecture with no problems at all. Ideally, in fact, both architectures should coexist, because the standard one shall be used for easy problems in controlled environments, like the invocation of web services in a corporate network. On the other hand, engines become more interesting when any of the implementation scenarios commented in section 6 arises.

## References

[1] D. Booth et al., “Web Services Architecture”, *W3C Working Group Note*, 2004, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[2] “Distributed Component Object Model (DCOM)”, http://www. microsoft.com/com/tech/DCOM.asp

[3] “Enterprise JavaBeans (EJB)”, http://java.sun.com/products/ejb

[4] “Web Services Invocation Framework (WSIF)”, http://ws.apache.org/wsif

[5] C. Venkatapathy and S. Holdsworth, “An introduction to Web Services Gateway”, *IBM*, 2002, http://www-106.ibm.com/developerworks/webservices/library/ws-gateway/

[6] G. Brose, “Securing Web Services with SOAP security proxies”, *Proceedings of the International Conference on Web Services*, June 2003, pp. 231-234

[7] EntireX XML Mediator, http://www.softwareag.com/Corporate/products/entirex/

[8] F. Curbera et al., “Business Process Execution Language for WS”, 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[9] Y. Golan, M. Nottingham and D. Orchard, "WS-Callback Protocol", 2003, http://dev2dev.bea.com/technologies/ webservices/WS-CallBack-0_9.jsp
[10] D. Bau and D. Orchard, "SOAP Conversation Protocol (SCP)", http://dev2dev.bea.com/pub/a/2002/06/SOAP Conversation.html
[11] K. Swenson and J. Ricker, "Asynchronous Web Services Protocol (AWSP)", http://xml.coverpages.org/ AWSP-Draft20020405.pdf
[12] A. Mani and A. Nagarajan "Understanding quality of service for Web services", *IBM Developer Works*, 2002, http://www-106.ibm.com/developerworks/library/ws-quality.html
[13] "QoS for Web Services Defined", *Santra Technologies*, 2003, http://www.santra.com/knowledge/?id=qos
[14] D. A. Menascé, "QoS Issues in Web Services", *IEEE Internet Computing*, vol. 6(6), Nov./Dec. 2002, pp. 72-75
[15] M. Tian, A. Gramm, H. Ritter and J. Schiller, "Efficient Selection and Monitoring of QoS-aware Web services with the WS-QoS Framework", *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*, pp. 152-158, 2004
[16] A. Dan, H. Ludwig and G. Pacifici, "Web Services Differentiation with Service Level Agreements", *IBM*, May 2003, http://www-106.ibm.com/developerworks/library/ws-slafram/
[17] S. Weller, "Web services qualification", *IBM Developper Works*, 2002, http://www-106.ibm.com/developerworks/ library/ws-qual/
[18] H. Ludwig, A. Keller, A. Dan, R. P. King and R. Franck, "Web Service Level Agreement (WSLA) Language Specification", http://www.research.ibm.com/wsla, 2003
[19] Douglas K. Barry, "Web Services and Service-Oriented Architectures", *Morgan Kauffman*, 2003
[20] L. Zadeh, "Knowledge Representation in Fuzzy Logic", *IEEE Transactions on Knowledge and Data Engineering 1(1)*, pp. 89-100, 1989
[21] J. Fernandez, J. Pazos and A. Fernandez, "An architecture for building web services with QoS features", *Proceedings of the 5th Web-Age Information Management (WAIM'04)*, July 2004
[22] J. Fernandez, J. Pazos and A. Fernandez, "High availability with clusters of Web Services", *Proceedings of the 6th Asia Pacific Web Conference (APWeb'04)*, April 2004
[23] J. Fernandez, J. Pazos and A. Fernandez, "Optimizing web services performance using caching", *Proceedings of the International Conference on Next-Generation Web Services Practices (NWeSP'05)*, August 2005
[24] Sahai, A. et al., "Automated SLA Monitoring for Web Services", *13th IFIP/IEEE International Wokshop on Distributed Systems (DSOM 2002)*, pp. 28-41
[25] "High Availability with QoS", *IBM and CISCO*, 2000, http://www-1.ibm.com/servers/eserver/zseries/library/ specsheets/high_availability_qos.html

**Julio Fernandez** received Master degree in Computer Science from the University of A Coruna (Spain-UDC) in 1992. He has been working as a mainframe system administrator from 1990 to 2004. He is the director of "Mainframe System" department since 2004 at a Spanish savings bank (Caixa Galicia). He has been working on a Ph.D. on web services since 2002, and he is currently working on a universal integration framework based on web services.

**Jose Pazos Arias** received Master degree in Telecommunications Engineering from the Polytechnic University of Madrid (Spain-UPM) in 1995. He received Ph. D. degree in Computer Science from the Department of Telematics Engineering, the Polytechnic University of Madrid (Spain-UPM) in 1995. He is the current director of the Networking and Software Engineering Group in the University of Vigo, since 1998. He is currently working on middleware and applications for Interactive Digital TV.

**Ana Fernandez** received Master degree in Telecommunications Engineering from the University of Vigo (Spain-UVigo) in 1996. She received Ph. D. degree in Computer Science from the University of Vigo in 2002. She is currently working in the Interactive Digital TV laboratory, since 2002. She is at present an Associate Professor in the Department of Telematics Engineering at the University of Vigo. She is currently working on web services technologies and ubiquitous computing environments.