# Implementation of Music Store Application Using Web Service Security

Nyunt Win and Khwar Nyo Oo
University of Computer Studies, Yangon, Myanmar
nyuntwin18@gmail.com, khwarnyo2008@gmail.com

## Abstract

*The task of creating and deploying Web Services is different from currently traditional web application. The tendency on all platforms is to automate as much as possible in creating web services under interoperable standards. By using web service description language (WSDL) and Simple Object Access Protocol (SOAP), company can create and describe their web service. Java technology and XML can be combined to deliver web services using JEE as a foundation. Systems based on the Service-Oriented Architecture (SOA) paradigm need to be able to bind to arbitrary (Web) services at run-time. For supporting JEE web service architecture, this thesis will discuss JAX-WS (JAVA for XML-based Web Services) for implementing music store application. This thesis will be implemented by JEE 5, WSDL and Tomcat application server.*

## 1. Introduction

Web service is an infrastructure for developing and deploying distributed applications. Web services are typically intended for applications consumption, in contrast with contemporary Web applications which are meant for human users. Software systems built on top of Service-Oriented Architectures (SOAs) intend to use a triangle of three operations, publish, find and bind, to decouple all roles participating in the system. Two elements of this triangle publish and find particularly, put requirements on the service registry. The third operation, bind, is independent from the service registry: binding has to be handled solely by the service consumer. Existing Web service client frameworks such as Apache Axis 2 and Apache WSIF imply a very strong emphasis on RPC centric and synchronous Web services. This paper will present a message-based dynamic service invocation framework that enables application developers to create service clients which are not coupled to any specific provider.

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms. The Web Service Architecture (WSA), as defined by the W3C, is intended to provide a common definition of a Web Service. WSA involves many layered and interrelated technologies. Some of technology families support Web Services including the XML technologies SOAP, WSDL and UDDI. Web Service basic structure is shown in Figure.

Web Services define as self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web Services have promised to change the Web from a database of static documents to an e-business marketplace. Web Service technology is being adopted in the Business-to-Business commerce applications and even in some Business-to-Consumer commerce applications. The widespread adoption of web services is due to its simplicity and the data interoperability provided by its supporting technology namely XML, SOAP and WSDL [1].

The rest of the paper is organized as follows; we first describe Related Works and Web Service Technologies followed by Proposed System and Design and Implementation. In the following section, we conclude the paper.

## 2. Related Works

The first Java-based Web service framework that incorporated the idea of dynamic service invocation was the Apache project Web Service Invocation Framework (WSIF) [2]. The WSIF dynamic invocation interface is intuitive to use as long as the client application knows the signature of the WSDL operation to invoke. We consider this to be unacceptable precondition for loosely-coupled SOAs – client applications should not have to care about service internals such as the concrete operation name. Another big caveat of WSIF is its notoriously weak support for complex XML Schema types as service parameters or return values. Complex types can only be used if they are “mapped” to an existing Java object beforehand, what is frequently impossible in dynamic invocation scenarios. These problems, along with the fact that the framework is not under active development since 2003 and the relatively bad runtime performance, render WSIF outdated today.

The Apache Axis 2 [3] framework incorporates a lot more SOA concepts than

WSIF: it supports client-side asynchrony and works much more on a document level than the strictly RPC-based WSIF. Even though Axis 2 is still grounded on the usage of client-side stubs it also supports dynamic invocations through the OperationClient or ServiceClient APIs. The disadvantage of these interfaces is that they expect the client application to create the entire payload of the invocation (e.g., the SOAP body) itself. In that case Axis 2 does little more than transfer the invocation to the server. This is not the level of abstraction that we expect from a Web service framework used to construct a SOA client. However, we recognize that the Axis 2 SOAP and REST stacks are well developed and highly performance. We therefore created a Axis 2 service backend as part of our DAIOS prototype in order to combine the advantages of DAIOS and Axis 2: the Axis 2 backend uses the dynamic invocation abstraction of DAIOS, but utilizes the Axis 2 service stack to carry out the actual invocation. Similar problems as present in Axis 2 arise with other recent service frameworks such as Codehaus XFire [4] or XFire's successor, Apache CXF [5].

Ultimately, all of these clientside frameworks are relying on static components to access Web services, with little to no support for truly dynamic invocation scenarios. JAX-WS (Java API for XML-based Web Services) is the latest Java-based Web service specification. JAX-WS is described in JSR (Java Specification Request) 224 [6], and is the official follow-up to the older JAX-RPC [7].

JAX-WS is implemented for instance in the Apache CXF project, and, therefore, exhibits similar problems – although the change in the naming suggests that JAX-WS is less RPC-oriented than its ancestor the specification still focuses on WSDL-to-operation mappings, ignoring the messaging ideas of SOA and Web services. REST is also not discussed explicitly in the JSR, even though the document claims to generally handle XML-based Web services in Java.

## 3. Web Service Technologies

A Web service is an interoperable unit of application logic that transcends programming languages, operating systems, network communication protocols, and data representation dependencies and issues. It is an infrastructure for developing and deploying distributed applications. Web services are typically intended for applications consumption, in contrast with contemporary Web applications which are meant for human users. Web Services are based on the following industry standards: eXtensible Markup Language (XML); Simple Object Access Protocol (SOAP); Web Services Description Language (WSDL); and Universal Description, Discovery, and Integration (UDDI).

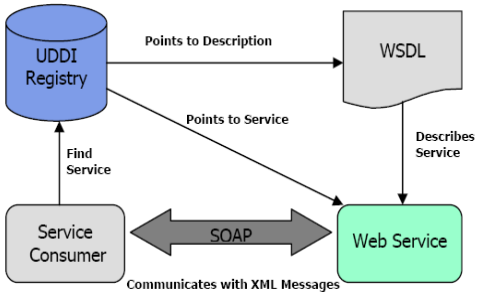


**Fig.1. Web Service Basic Structure**

### 3.1 Web Service Background Theory

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms. The Web Service Architecture (WSA), as defined by the W3C, is intended to provide a common definition of a Web Service. WSA involves many layered and interrelated technologies as shown in Figure 1.

#### 3.1.1 WSDL

WSDL is used to describe a Web service, to specify its location, and to describe the operations the service provides. WSDL-based document provides enough information about how to interact with the target Web service. It contains five elements: three abstract elements (types, messages, and port Type), and two concrete elements (binding, and service address). The abstract elements define the interface: types are embedded XML Schema where data types are defined; messages describe details of methods and their parameters; and portType defines operations in terms of input/output messages. The concrete elements, on the other hand, define physical properties: binding provides protocol information for the operations; service address specifies URI for locating a service. WSDL is used in generating a proxy class - a client-side object that mimics the method calls available on the Web service. Application developers work with the proxy instead of directly writing SOAP messages. Proxy handles message construction, and sending/receiving of SOAP messages.

#### 3.1.2 SOAP

SOAP is an XML-based communication protocol and encoding format for inter-

application communication. SOAP is widely viewed as the backbone to Web services. SOAP message specification has three parts: envelope for data encapsulation; data encoding rules; and RPC conventions. The envelope is the root node of a SOAP message and has two parts: header and body. The envelope header specifies application-level requirements: digital signature for password-protected services, account number for pay-per-use SOAP services, and transaction management. The envelope body describes message contents and processing instructions - application-specific data such as method name, parameters, and return values.

### 3.1.3 UDDI

Universal Description, Discovery, and Integration (UDDI) registry is a collection of information on all the registered Web services. It enables dynamically discovering Web services providers. UDDI is a free public registry - vendors publish their Web services and consumers search for appropriate Web services. It has three components: white pages - contain address, contact details, and known identifiers for Web services providers; yellow pages - have industrial categorization of Web services based on standard taxonomies; and green pages - contain technical information about services. UDDI is relatively light-weight, and contains enough information to direct users to resources hosted outside of it. It uses XML to represent its contents.

## 3.2 JAX-WS Web Service

JAX-WS provides facilities of XML-based message exchanging mechanism over the Web Services. There are two tools are provided by JAX-WS to make Java application developer easier to program Web Services applications and those are such as:

- WSGen for Service Provider
- WSImport for Service Requestor

As shown in the Figure 2 of architecture overview, Service Endpoint (Service Provider) and Proxy Code (Service Requestor) will be generated by those tools to hide complexity of the Web Services programming. The actual Web Services component can be invoked and it is called as Service Endpoint. The Service End Point consists of following Java components: Service Endpoint Interface (SEI) and Service Implementation Class.

The reason of JAX-WS is to refocus the library's usage to a more web service centric model. It provides java applications to communicate with each other using SOAP protocol. The programming model of JAX-WS is unique in that the client program will attempt to bind to a service through the loading of WSDL file or through reflection upon a class that has been annotated with web service descriptor. Upon acquiring a service, the program will request the specific port and call the method desired.

JAX-WS also includes the Java Architecture for XML Binding (JAXB) and SOAP with Attachments API for Java (SAAJ). JAXB enables data-binding capabilities by providing a convenient way to map an XML schema to a representation in Java code. The JAXB shields the conversion of the XML schema messages in SOAP messages to Java code without having to fully understand XML and SOAP parsing. The JAXB specification defines the binding between the Java and XML schemas. SAAJ provides a standard way of dealing with XML attachments contained in a SOAP message.

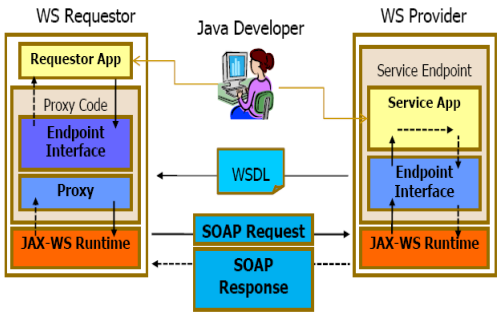


**Fig.2. JAX-WS Architecture**

## 4. Web Service Security

Open standards have gained momentum among enterprises as a mechanism for Web services to communicate with partners, customers, and suppliers. XML, Simple Object Access Protocol (SOAP), and HTTPS are among the technologies for developing interoperable Web services with open, flexible, and adaptive interfaces. That inherent openness, however, poses security risks. Without proper protections, Web services can expose vulnerabilities that could lead to dire consequences. Ensuring that those services and their communications are integral, confidential, and secure is critical for all parties.

To provide proof that the sender of the request is the authenticate user, he/she must also include a digital signature. For all requests except those using SOAP with WS-Security, the signature is calculated using Access Key. For example Amazon Web Service uses the Access Key ID in the request to look up Secret Access Key and then calculates a digital signature with the key. If the signature AWS calculates matches

the signature you sent, the request is considered authentic. Otherwise, the request fails authentication and is not processed.

In this paper for generating signature, we can calculate HMAC hash by using a base64-encoded HMAC_SHA256 signature and construct a request to web service by using SOAP request. A keyed-hash message authentication code (HMAC-SHA) signature can be calculated. After getting the signature and then we send the request to WS with this signature. In web service server, the system generates a signature from the request data and calculates the signature we sent in the request. If the signature generated by WS matches the one we sent in the request, the request is considered authentic. If the comparison fails, the request is discarded, and WS returns an error response.

## 5. Proposed System

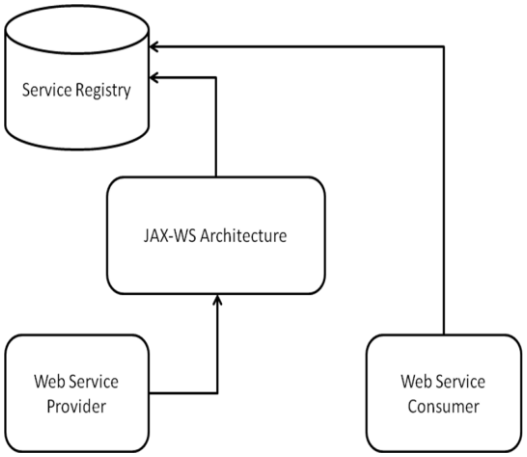


**Fig.3. Architecture for Proposed System using JAX-WS Architecture**

Figure 3 shows a simple architecture for proposed system using JAX-WS architecture. Service Providers advertise their services in service registry. This registry serves as a repository for the service advertisements, against which the service requester's queries are matched. Service provider will create Web services that show item lists, accept order information, shipping information, and ordered items and accept payment. Service requestors input their data and the system will convert them into WSDL. Service requestors use the artifacts generated to invoke the Web service. The Web service clients do not need to deal with any SOAP format, like creating or parsing SOAP messages. Rather, this is handled by the JAX-WS run time, which uses the generated artifact code (the JAXB-generated class). The Web service client in turn deals with the Java object (the JAXB-generated class), which eases the development of Web service clients and invoking operations on the Web service.

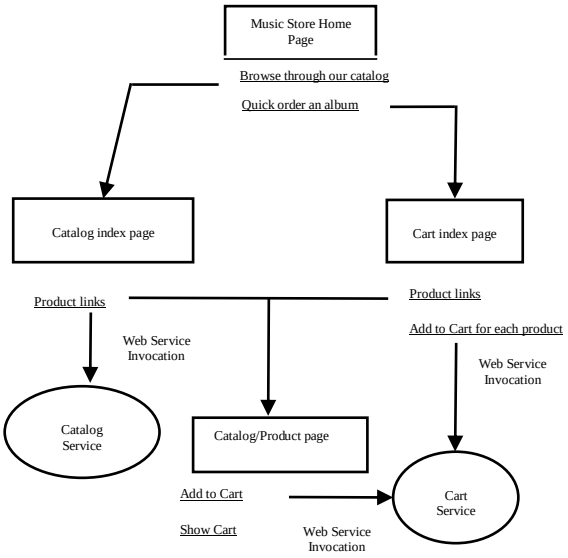


**Fig.4. Proposed Web Services for Music Store Web Application**

This paper will use JAX-WS architecture to implement Catalog Service which provides number of store items, and to expose this Catalog Service as a Web Service. JAX-WS client can use this Catalog Web Service remotely in a Music Store web site. As shown in Figure 4, the system will also use Catalog Service and Cart Service. Catalog Service will have product links and listen to samples links. Service requester can start catalog service by clicking "Browse through our catalog" link. If the user has already registered, the user browsing through the Music Store web site can add any album on the site to shopping cart. The user can use Cart Service to buy the items in the shopping cart. After that, the user can submit order and the system will show another page that informs the user that the order is successful.

## 6. Design and Implementation

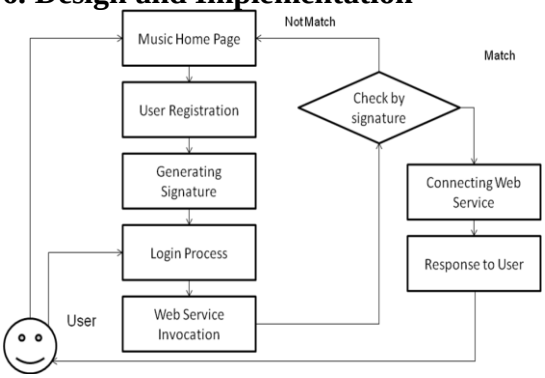


**Fig.5. Flow Chart of the Proposed System**

Figure 5 shows the flow chart of the proposed system. In this system, user must register to use catalog service and cart service of music store application. After registering, the system will generate signature with access key. In login process, users will login with their name and password. The system will give their access key and signature value. The user will enter their access key and the system will compare their signature and generated signature from web service provider. If these signatures match, the user can use catalog service and cart service. If not, the user will be denied. The screen shot of this system are shown in Figure 7, 8 and 9.

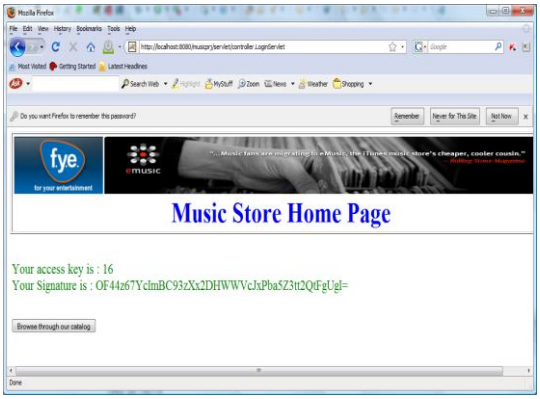


**Fig.6. Generating signature of Music Store Application**

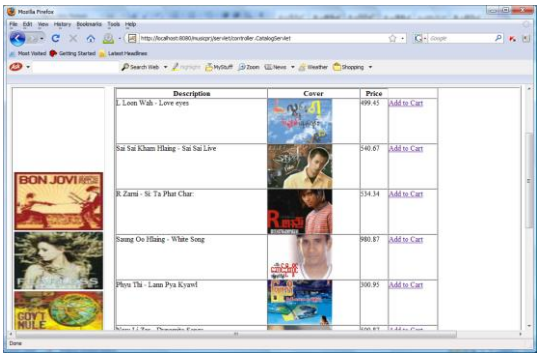

**Fig.7. Catalog Web Service of Music Store Application**

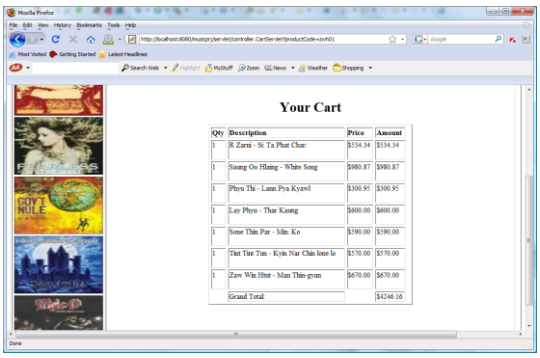


**Fig.8. Shopping Cart Web Service of Music Store Application**

## 7. Conclusion and Future Work

This paper will describe developing web application using JAX-WS implementation. The successful development of web services will greatly depend on the ability to automate as much of the web services possible under interoperable standards. Service discovery is one important concept of web service's e-commerce interactions. JAX-WS is JEE enabling technology and it allows us to forgo the work of navigating the different web service layers and concentrate on accessing the functionality we require.

## References


[1] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, "Web Services Description Language (WSDL) 1.1", W3C Note.

[2] Apache Foundation. Web Services Invocation Framework. http://ws.apache.org/wsif/.

[3] Apache Foundation. Apache Axis 2. http://ws.apache.org/axis2/.

[4] Codehaus XFire. http://xfire.codehaus.org/.

[5] Apache Foundation. Apache CXF: An Open Source Service Framework. http://incubator. apache.org/cxf/.

[6] D. Kohlert and A. Gupta. Java API for XML-Based Web Services, Version 2. http://jcp.org/aboutJava/communityprocess/ mrel/jsr224/index2.html, 2007.

[7] JSR-101 Expert Group. Java API for XMLBased RPC, Version 1.1. http://java.sun.com/xml/downloads/jaxrpc.ht ml# jaxrpcspec10, 2003.

[8] S. Nagano, T. Hasegawa, A. Ohsuga, and S. Honiden. Dynamic Invocation Model of Web Services Using Subsumption Relations. In ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04), 2004.