

Common Architecture for Functional Extensions on Top of Apache Axis 2

Jaliya Ekanayake and Dennis Gannon
Department of Computer Science
Indiana University Bloomington, Indiana
email{jekanaya, gannon}@cs.indiana.edu

ABSTRACT

Quality-of-services in the domain of Web services plays a major role when utilizing Web services to various business scenarios. Reliable Messaging and secure conversation, governed by WS-ReliableMessaging [3] and WS-SecureConversation [4] specifications respectively, and many other WS- specifications are such quality-of-services that most of the business integrations expect from Web service engines. Apache Axis2, the new implementation of the Apache's web services engine, was designed with the intention of supporting these quality-of-services and as extensions to the Axis2 engine. This contribution introduces a common architecture to implement various quality-of-services on top of Apache Axis2 avoiding future incompatibilities between these implementations.*

INTRODUCTION

Apache Axis2 architecture supports various quality-of-services such as reliable messaging and secure conversation as functional extensions that are implemented by utilizing the functionalities of the handler framework. Axis2 provides a mechanism for implementing these extensions using the concept of handlers. Handlers are software components which are invoked by Axis2 engine on the reception and sending of SOAP messages passing the current SOAP message into them. Handlers can interpret and modify the passed SOAP message and the engine will keep on invoking the handlers that are deployed for a particular message path, until either the SOAP message is handed over to the final recipient or it is transmitted to the transmission medium.

Axis2's handler framework consists of different set of levels and rules using which the handlers can be configured and ordered. Axis2 uses the concept of flows and phases [1] to arrange the handlers, where flow represents a message path inside the engine such as InFlow, OutFlow and FaultFlow [1] and the phases represent a set of levels for that handlers can be configured. Axis2 use predefined phases as well as custom defined phases and this makes the configuration of handlers inside the engine more flexible. Axis2 engine provides the necessary implementation for deploying and configuring these handlers and also the mechanism to route a given SOAP message through the handlers, while handlers are used to support various custom processing for the SOAP message and also to support quality-of-services.

Different quality-of-services imposes different constraints to the message flow and may even introduce new messages to the communication. Business scenarios require combinations of these quality-of-services to achieve a required level of quality-of-services such as reliable and secure. Although the improved handler framework in Axis2 makes the implementation of these quality-of-services easy it is important to follow a common architecture that prevents incompatibilities between these implementations. This paper discusses a common architecture that can be used to implement these functional extensions on top of Apache Axis2 with respect to the ongoing work on implementing WS-ReliableMessaging[3] and WS-SecureConversation[4] support for Apache Axis2 and present a generalized the architecture to support different other functional extensions.

RELATED WORK

Axis2 Architecture

Axis2 architecture is a modular architecture that can be broken into seven different modules as shown in the following diagram.

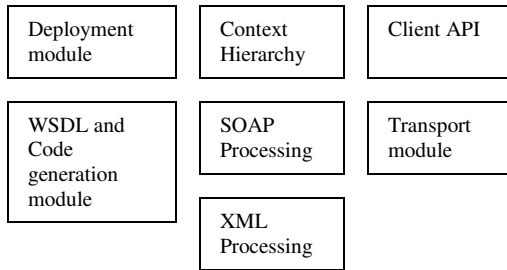


Figure 1: Axis2 Architecture

Context hierarchy is the information repository for Axis2 engine and can be accessible by handlers and services. Deployment module and WSDL[10] and code generation module update the context hierarchy. Deployment module process deployment specific information and provides a mechanism to deploy services and handlers both statically and dynamically. XML processing module in Axis2 is based on XML pull parsing and provides an object model (AXIOM- Axis Object Model) for the SOAP[11] processing module.

SOAP[11] processing module contains the Axis2 engine and the handler framework that can be used to implement different functional extensions. Client API(Application Program Interface) provides users with a flexible API to invoke various Web services scenarios. Transport module handles various transport specific functionalities such as sending and receiving messages using different transport protocols.

Axis2 Handler Framework

Axis2's handler frame work consists of flows phases, handlers and Axis2 engine. In addition Axis2 introduce a new concept called Message Receiver [1] which is the last component that receives a particular SOAP message from the point of view of the Axis2 engine. Message

receivers can be implemented to support Web service invocations or to do any other processing by the user. By default Axis2 provides two message receivers to handle In-Only and In-Out Web services invocations, namely InOnlyMessageReceiver[1] and InOutMessageReceiver [1].

Axis2 engine is based on a one way message processing model where the engine either perform send or receive functions with respect to a particular SOAP message. When ever a SOAP message is received by an Axis2 engine (either in the server side or in the client side) engine will perform receive operation and when a message is going out from the engine then the engine perform send operation. Send and receive either involve routing of SOAP message through a set of handlers till it reach either final receiver (Message Receiver) or till it reach the transport medium as shown in the following diagram.

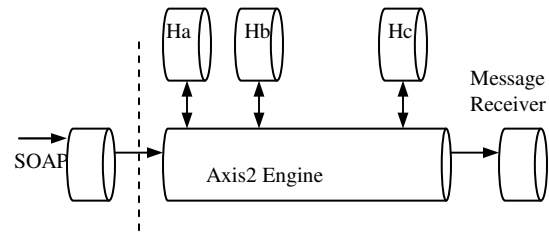


Figure2: Axis2 engine's receive() operation

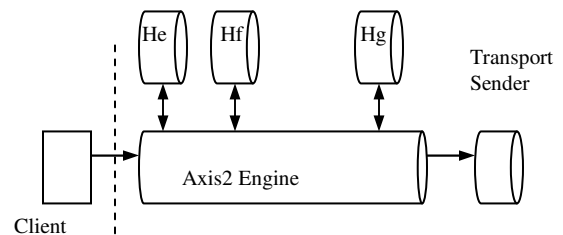


Figure3: Axis2 engine's send() operation

Axis2 provides a mechanism called module to group set of handlers that are used to provide some unique functionality. Phases and Flows are used to arrange modules for a particular message path. Flows represent possible message paths inside the engine and can take values such as In-Flow, Out-Flow, In-Fault-Flow and Out-Fault-Flow. Phase represents various levels of a

message flow and modules can be deployed in different phases. Following diagram illustrates a possible flow arrangement.

In-Flow

| | | | |
|-----------|--------------|----------|---------------|
| Transport | Pre-Dispatch | Dispatch | Post-dispatch |
|-----------|--------------|----------|---------------|

| | | | |
|--------|------------|--------------------|--------------------|
| Policy | User phase | Message validation | Message processing |
|--------|------------|--------------------|--------------------|

Out-Flow

| | | | |
|--------------------|--------|------------|-----------|
| Message Initialize | Policy | User phase | Transport |
|--------------------|--------|------------|-----------|

Figure4: Axis2 Phases

User can specify any of these phases as the phase for a particular handler in a module that he wants to deploy and the handlers in a single module may expand into several phases in a flow. Predefine phases are used to perform generic functionalities such as dispatching and transport whereas user defined phases are used to deploy custom modules. Phases play a crucial role in handler arrangement as various scenarios expect the SOAP message to be routed through an ordered set of handlers.

Axis2 also provides the capability of pausing a message in a given handler and resuming it back from that point. This feature is very useful in supporting scenarios such as reliable messaging where messages should be processed in order of their respective message numbers. In addition it will also support dynamically adding operations to a given service at the deployment time. This feature is useful when dispatching various quality of service specific SOAP messages that has SOAP body [&&] element but not a part of the deployed web service but targeted to it.

Axis2 has improved its runtime information model using a context hierarchy, consisting of message context, operation context, service context, service group context and configuration context, that allows complete access to the underlying information model to the handlers and services. This feature allows the handlers to

share data across various messages and provide different functionalities not only based on a single message but also on several sequences of messages.

Axis2 has the complete support for asynchronous messaging ranging from API level asynchronous support, where the client is not blocked during a service invocation but the transport connection is blocked, to complete asynchronous support, where both client and the transport connection are not blocked. Later allows the client to expect a Web service response in a different transport connection and Axis2 provides the necessary listeners for the above configuration.

REQUIREMENTS FOR WS-*

Various WS-* specifications imposes different changes to the message flow in a normal web service’s communication. These changes can be categorized into two main types. Some specifications require only modification of incoming and outgoing messages mainly at the SOAP header level. WS-Addressing and WS-Security are such specifications where WS-Addressing add and interpret addressing related SOAP headers and WS-Security encrypts or decrypts the entire message or part of the message. Second category imposes additional messages to the communication such as CreateSequenceRequest[3] in WS-ReliableMessaging and RequestSecurityToken [4] in WS-SecureConversation[4]. Although they are semantically similar messages that handle protocol initialization the ways that they are used may be different. RequestSecurityToken[4] message is targeted to a web service that generates and manages security tokens, whereas CreateSequenceMessage[3] in WS-ReliableMessaging[3] is sent to the same web service that has the business logics. These requirements impose additional support from a soap engine.

AXIS2’S SUPPORT

Implementing the first type of WS-* can be done by using a set of handlers that interpret or

modify the messages in the communication path. Supporting second type of WS-* where they add additional messages to the communication requires a feature to pause (temporary stop from processing) [1] messages till some precondition is established. Axis2 provides a mechanism for pausing messages at the handler and resume them back whenever the handler decides to do so. Once resumed Axis2 engine will route the message through the remaining handlers in the message flow from the handler that paused its execution. In addition Axis2 provide a mechanism for modules (module is a collection of handlers typically implementing some functional extension) to add operations to an already deployed Web service for which the module provide functional extension support, although the implementation of these additional operations is handled by the module itself. This feature allows Axis2 to dispatch messages that are targeted to a particular Web service yet the interpretation and the implementation is up to the functional extension. Following diagram illustrates this feature.

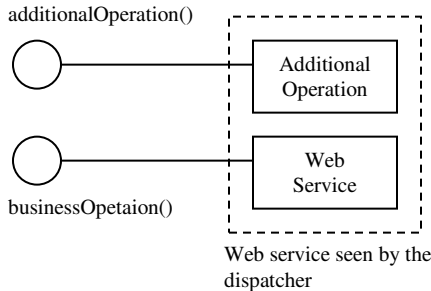


Figure5: Addition of Operations

This is highly useful feature in a scenario such as WS-ReliableMessaging[3] where all the protocol specific messages such as CreateSequenceRequest[3], TerminateSequenceRequest[3] messages are sent to the actual Web service itself. The dispatcher can bind each of these messages to respective operations that are now visible in the pseudo Web service interface as seen by the dispatcher.

COMMON ARCHITECTURE FOR WS-*

WS-ReliableMessaging[3] and WS-SecureConversation[4] both have the requirement of sending protocol specific messages in addition to the Web service requests and responses. Also in a typical usage scenario both of them will be used together to provide secure reliable communication channel to the Web services. WS-* such as WS-Addressing[5] and WS-Security[6] require only modifications to the messages and the implementations of these can also be done adhering to the following architecture based on the handler framework of Axis2 as shown in the following diagram.

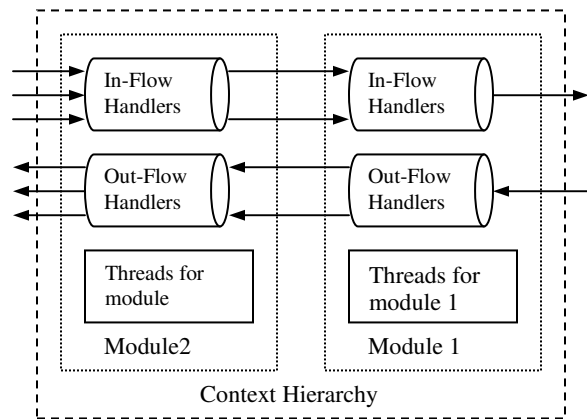


Figure6: Common Architecture

Each module comprises of several handlers that may operate in different phases of different flows. Handlers can modify, pause, resume or add new messages to the message path. To resume messages and also to send a given message multiple times (e.g. in the case of retransmissions in WS-ReliableMessaging[3]) handlers use additional threads. The context hierarchy that contains all the information about the handler configuration as well as engine and service specific configuration information is available to modules.

When a message arrives to a particular handler it can decide to process the message and forward it to the next set of handlers or it can pause sending the message. Either the context hierarchy or some other persistent mechanism can be used to store the paused status of the

message. If the handler needs new messages to be sent before sending the message that it has been paused, it can do so by creating and sending new messages using the same axis engine that can be obtained using the configuration context. If there is a requirement to send these messages multiple times or expect asynchronous responses to these new messages then the handler can incorporate a thread to handle sending. Once the necessary conditions for sending the message that has been paused are met the thread can send the paused message by calling resume mechanism provided by the Axis2 architecture. Any message going out from the module one's handlers that operate in phase before module two's handlers are in will reach the module two's handlers if they are deployed as shown in the figure [&&]. Module two can again use the same approach to handle those messages and again it can send new messages in addition to the messages that it receive from the module one.

Sandesha2 Implementation and the support for WS-SecureConversation

Above architecture was used to implement Apache Sandesha2 that is an implementation of WS-ReliableMessaging[5] on top of Apache Axis2.

Following diagram illustrate the core components of Apache Sandesha2.

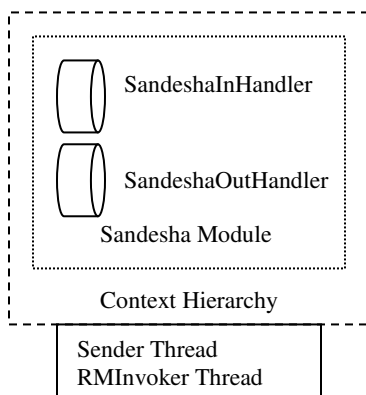


Figure6: Sandesha2 Module

Two handlers namely SandeshaInHandler[2] and SandeshaOutHandler[2] are used in both client and the server sides. Any outgoing message is

handled by the SandeshaInHandler[2] and any incoming message is handled by SandeshaInHandler in both client and server sides. Any message received by the SandeshaInHandler[2] will be first check for the validity and the message is identified by interpreting the headers. If the message is any of the WS-ReliableMessaging[3] specific message such as CreateSequenceRequest[3], TerminateSequence[3] etc... then the required book keeping is performed and the message is paused and ignored. Messages that are paused and ignored will not be resumed later and just discarded by the axis2 engine. If the message has an associative response then, it will be generated by the handler and saved in a queue. (Context hierarchy or any persistent mechanism can be used to save the messages) Sender thread that continuously monitors this queue will pick up the message to be sent and use Axis2 engine to send the message to the required destination.

If the message is an application (request or response of a web service) then it will be paused and put into a queue after the initial book keeping. A separate thread (InOrderInvoker[2]) is used to resume those messages in the correct order to invoke web services. Resume operation provided by Axis2 will continue processing of the message from the point where the message has been paused.

Any message coming to RMOutHandler[3] will undergo a similar processing, except that it needs to send WS-ReliableMessaging[3] specific messages such as CreateSequenceRequest[3] TerminateSequenceRequest[3] etc.. Application messages are paused and queued, till the necessary sequences been established. Whenever the necessary conditions are established, the sender is used to send the messages that are in stored in the queue.

Apache Sandesha was implemented using the above architecture and since WS-SecureConversation[4] is under development using the same architecture. Sandesha module will inject additional messages to the message path and both application messages as well as these additional messages will be received by the WS-SecureConversation[4] module once we

placed it in a higher phase than Sandesha module. All these messages are application messages for WS-SecureConversation[4] module and it can proceed with its operation and may send new messages as well. Each module filters these additional messages at their respective phases and modules operating in lower levels will not see these additional messages as illustrated by the arrows in figure [Figure 6]

CONCLUSIONS AND FUTURE WORK

Apache Axis2 has introduced new concepts and features for implementing various Web services support. WS-* represent a set of functional extensions that is based on Web services and all SOAP engines are eventually required to provide the necessary support for these specifications. Axis2 has developed to cater all these new requirements that the Web Services domain is requesting for a SOAP engine and it has also provided new set of capabilities such as asynchronous support, message pausing and resuming. Although these improvements has made the WS-* implementation on top of Axis2 fairly easy, a common architecture for these implementations is a must in order to avoid future inconsistencies and integration problems.

The architecture presented in this paper has been used to implement Sandesha 2 and it has been tested for the interoperability with Microsoft Indigo [9] framework and it has passed all the tests. An implementation of WS-SecureConversation[4] on top Apache Axis2 is under construction using the above architecture and no inconsistencies have been identified, showing that the common architecture presented in this paper works well with WS-* implementations on top of Apache Axis2. Future work involve in implanting other WS-* support on top of Axis2 and since most of the other WS-* have the same or subset of requirements as WS-ReliableMessaging [3] and WS-SecureConversation [4] so the above architecture can be used to implement functional extensions such as WS-Coordination [7], WS-AtomicTransaction [8] as well.

REFERENCE:

- [1]. Apache Axis2, Architecture Guide
<http://ws.apache.org/axis2/Axis2ArchitectureGuide.html>
- [2]. Apache Sandesha2,
<http://ws.apache.org/sandesha/sandesha2/index.html>
- [3]. WS-ReliableMessaging specification, February 2005,
<http://xml.coverpages.org/WS-ReliableMessaging200502.pdf>
- [4]. WS-SecureConversation Language specification, February 2005,
<http://specs.xmlsoap.org/ws/2005/02/sc/WS-SecureConversation.pdf>
- [5]. WS-Addressing specification, W3C Member Submission 10 August 2004,
<http://www.w3.org/Submission/ws-addressing/>
- [6]. Web Services Security (WS-Security) Version 1.0 05 April 2002, <http://www-128.ibm.com/developerworks/webservices/library/ws-secure/>
- [7]. WS-Coordination specification, Version 1.0 August 2005,
<ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf>
- [8]. WS-AtomicTransaction specification Version 1.0 August 2005,
<ftp://www6.software.ibm.com/software/developer/library/WS-AtomicTransaction.pdf>
- [9]. Microsoft Interoperability Test,
<http://131.107.72.15/ilab/wcinteroplab.htm>
- [10] Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001,
<http://www.w3.org/TR/wSDL>
- [11] SOAP Version 1.2,
<http://www.w3.org/TR/soap/>