Deployment Problems in Axis, Suggestions and Workarounds

Community Grids Lab, Indiana University

In this document we provide an overview of the problems that we encountered while deploying Web Service specifications within Axis. There are several important capabilities, pertaining to support for Web specifications, that are currently missing in Axis (1.2 RC3). We are hoping that these problems will be resolved in Axis so that Web Service specifications can be deployed in a less cumbersome fashion than at present.

We first enumerate problems that we encountered; subsequent sections elaborate the need for specific features and how (if possible) the lack thereof was circumvented.

1. Injection of messages: Within Axis currently only the Clients are allowed to inject messages.
2. Based on the request-response paradigm: In Axis every message is considered a request which should have its accompanying response within a pre-defined period of time.
3. No ability to gracefully terminate processing related to a message within the Handler chain associated with a service.
4. Handlers cannot initiate messages on their own.
5. Static configuration of the handler chain.
6. Problems with initiating responses.
7. Problems with corruption of SOAPMessages; specifically the deviation from the original intended schema-validated XML document encapsulated within the SOAPMessage and the SOAPMessage itself.

1. Message Initiation

Message initiation is a difficult task. Clients are the only entities that are able to initiate messages as requests. Server-side components either a handler in the handler-chain or the target service do not have this capability. The only message initiation is via a request/response mechanism where requests can be initiated only by the clients. There are several scenarios that indicate the need for message initiation in the server part.

Consider the case of Acknowledgements in WS-ReliableMessaging (hereafter WSRM) which requires message initiation from the WSRM Sink to the Source. The WSRM specification requires endpoints to comply with the acknowledgement and retransmission intervals that are exchanged prior to the creation of a Sequence. This implies that in acknowledgements may be issued several seconds after the receipt of a message. Furthermore, a single acknowledgement may encapsulate information pertaining to the receipt of several thousand messages. The limitations within the request-response paradigm are clear in such scenarios. It is clear that one-way messaging and message-initiation would resolve this particular problem. The problems outlined here also arise during retransmissions initiated by a WSRM source.

WS-Eventing (hereafter WSE) is another scenarios where message initiation is needed. A message may need to be reproduced to send the copies to multiple subscribing end points.

2. Other request-response based problems

Sometimes an entity may need to inform another entity without the need to receive a response. For example, when we issue a WSRM acknowledgement we are not looking for an acknowledgement to the WSRM acknowledgement.

3. Ability to terminate processing related to Message within a Handler chain

Currently there is no ability to gracefully terminate processing related to a message within the Handler chain associated with a service. Once a message has been received within a handler chain there is no graceful way to prevent this message from reaching the Service. A good example of the need for this feature would be the case of acknowledgments in WSRM. Only the WSRM handler needs to know whether the WSRM sink has received the message that was previously sent. Also, since most WS specifications are aimed at incrementally adding functionality to a service, situations may arise where similar such handshakes/control-messages should be stopped from reaching the actual service. In most cases such messages result in problems at the service.

Currently the only way to do so is to throw an Exception. Furthermore, there is no way to correctly access or interact with the handler-chain managing a specific handler. Although access to the AxisEngine instance is available, we cannot stop the message propagation.

We may need to retransmit the message as it is required in reliable messaging. If an acknowledgment has not been received from an end point, the message must be resent in order to ensure reliable messaging. As we said, although it is easy in client part, this task requires several tricks in server part.

4. Handlers cannot inject messages

This issue was mentioned earlier too, but needs to be clarified in the case of handlers too. Most WS specifications are naturally implemented as handlers that can augment a service's capabilities. However, such handlers need to be able to initiate control-messages on their own accord. While there is access to the Handler Chain there is no access to message propagation features. This feature may be made available through the AxisEngine class or through the handler chain itself.

5. Static Handler chains

In Axis handlers are currently statically configured – the configuration being made when a service is being deployed. A handler can not be added or removed from a service. Current axis architecture allows cloning the handler chain. The cloned chain can replace a current one after required changes are applied. But, this is not sufficient. Dynamic configuration of handler chains will be very useful. A deployment may have lots of handlers but a user/handler should be able to select a group of handlers that a message would need to go through.

This is especially true in cases of retransmissions where the handler may have already processed the message resulting in duplicate processing which may or may not lead to errors. Similarly security requirements may result in a message be passed through a different set of handler chains – here handlers related to message digest, encryption and signing may be added to the outgoing path in the handler chain.

6. Problems with initiating responses and Corruption of SOAP Messages in Axis

We encountered several problems during our implementation. One of them affected our implementation strategy and we believe it should be fixed in newer axis versions; we used a message-style Web Service. When we tried to send a response back from the server, the original SOAPEnvelope is being modified. There exist four valid signatures for message-style service methods. Since we need the SOAP header, we are using following method:

```
public void method(SOAPEnvelope request, SOAPEnvelope response);
```

The problem starts when we try to inject my `org.apache.axis.message.SOAPEnvelope` by using the `getBody()` `getHeader()` `setBody()` and `setHeader()` methods of the SOAPEnvelope. After injection of the response the SOAPBody and SOAPHeader include different prefixes than those which were originally supplied. While change of prefixes is in itself not a problem, there are problems with validation since the Axis processing eliminates prefixes for the SOAP Body but not for the SOAP Envelope or the Header. Please note that this is no longer a valid SOAP message since there is no SOAP Body! Another interesting side effect is that the processing removes prefixes from elements contained in the Body of the SOAPMessage, for e.g. a WSRM Acknowledgement response will have some wsrm: prefixes removed and some left intact. Needless to say such a message cannot be processed at another Axis client not to mention another Container since the over-the-wire XML is invalid.

This results in exceptions at the receiving side.

The scenario is as follows;
Firstly, we create a valid SOAPEnvelope:
org.apache.axis.message.SOAPEnvelope soapEnvelope;

We get the SOAPBody and SOAPHeader from envelope:

```
org.apache.axis.message.SOAPBody soapBody = (org.apache.axis.message.
          SOAPBody) soapEnvelope.getBody();
org.apache.axis.message.SOAPHeader soapHeader = (org.apache.axis.message.
          SOAPHeader) soapEnvelope.getHeader();
```

Next, we insert them into response:
```
response.setBody(soapBody);
response.setHeader(soapHeader);
```

At this point, client throws exception.

```
response.setEnvelope(soapEnvelope);
```

We tried setSOAPEnvelope method of SOAPEnvelope class. Somehow, it does not work. The response message reaches to the client as an empty message.

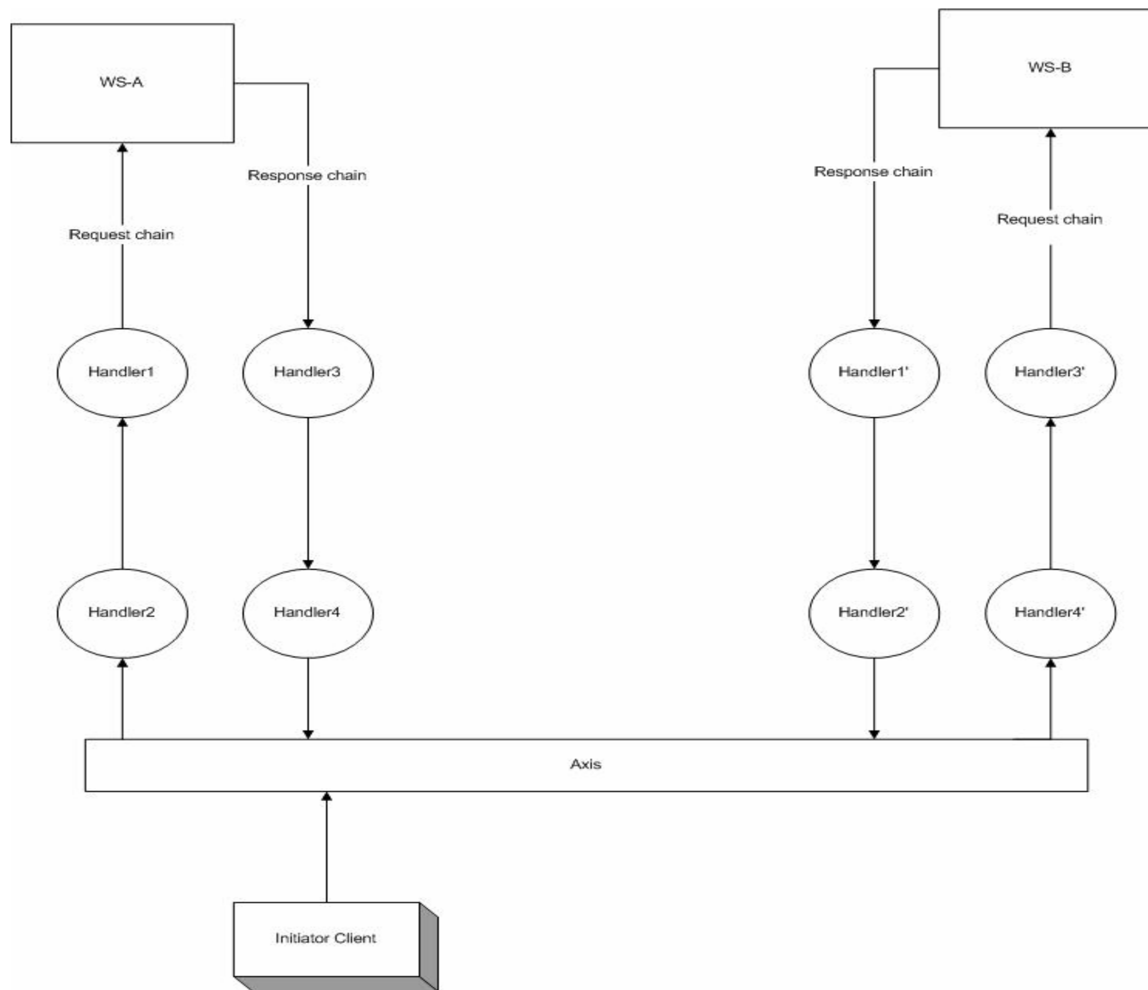
7. Solutions to some of the problems

We have temporary solutions for some issues above; instead of blocking a message, a message can be set to a dummy task. We had to choose this method because we wanted to stop the propagation of the message without getting an exception. By letting the message arrive at the end point we have prevented an exception being thrown. On the other hand, there exists a downside to this solution. This adds to performance costs because of the processing of the dummy task. At present we take this as an acceptable performance degradation within the current axis architecture.

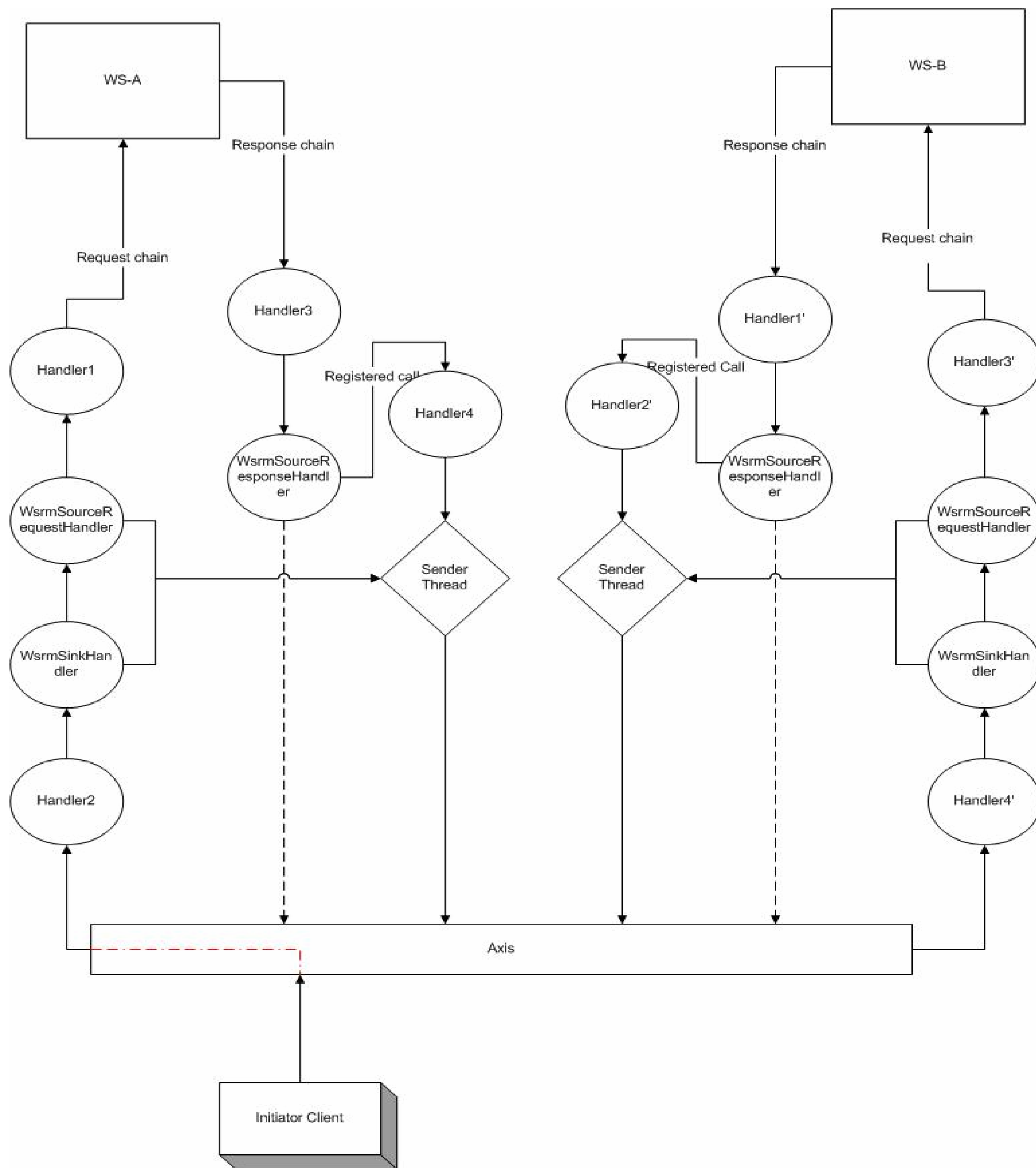
Although messages can not be initiated by the server part, we can bypass this restriction by using an Axis client wherever a message initiation is required. In this architecture, both source and destination will have client and server capability. Current Axis architecture requires the server part to reside in a web container; we use Apache Tomcat as the web container. As a result, both source and destination should be in the web container.

Next, we developed a sender thread which is responsible for sending any message to the other node. Since we are using one-way messaging, we gave the responsibility of sending messages to this thread instead of dealing with the Axis message responding mechanism (because of problems related to message corruption). The thread enables non-blocking sending of messages.
We did not use a response chain because we came up with the idea of building our own chain structure. After some point in the response handler chain, we diverted the massage path to the sender thread and added our own handler chain. It allowed us one-way messaging without breaking any of the already deployed handler structure at a service. In addition, this can easily support dynamic handler chains.

For example, let’s assume we have following two web services that send messages to each other.



We can add a reliable messaging into the handler chain as shown in the following picture. For reliable messaging, while the request handler chain should include two handlers, sink and source, the response handler chain consists only of source handler. The messages are sent by "Sender Thread". In order to prevent the loss of Handler4 or Handler2' functionality, the chain structure, we mentioned above, is added to the system.


WS-A
Response chain
Request chain
Handler3
Handler1
Registered call
Handler4
WsrmSourceResponseHandler
WsrmSourceRequestHandler
Sender Thread
WsrmSinkHandler
Handler2
WS-B
Response chain
Request chain
Handler1'
Registered Call
Handler2'
Handler3'
WsrmSourceResponseHandler
WsrmSourceRequestHandler
Sender Thread
WsrmSinkHandler
Handler4'
Axis
Initiator Client