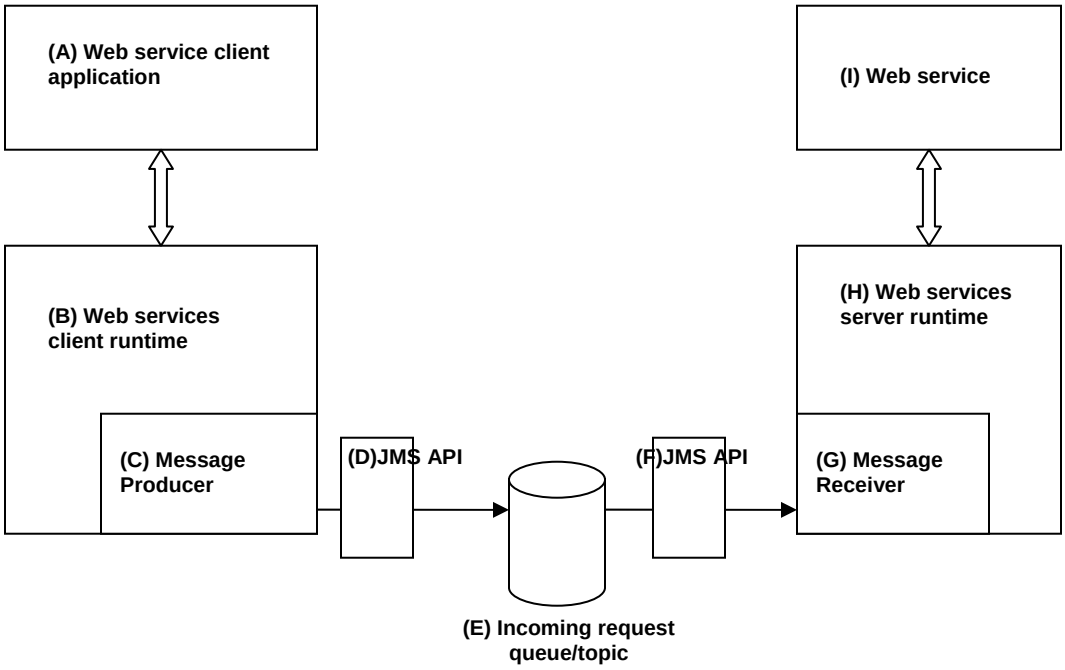
Proposal for a SOAP/JMS testing strategy

Here is a high-level description of a possible approach for testing SOAP/JMS conformance. There are lots of details yet to be determined, but the intent is to stimulate some discussions.

First, let's define (at a high level) the environment that will be tested. I tried to model this after Eric's picture in his email to the soapjms mailing list.

A SOAP/JMS implementation will likely consist of both a message producer and a message receiver. Here's a picture that describes the environment:



We have a web services client application (A) that is supported by the vendor's client runtime (B), which contains the message producer (C). The message producer (C) uses the JMS API (D) to create and deliver request messages to the request queue or topic (E). On the server side, the message receiver (G) is contained in the vendor's server runtime (H) and receives the request message from the request queue/topic (E) and processes it by using the JMS API (F).

The main purpose of the conformance test would be to test components (C) and (G) to verify that they are in compliance with the SOAP/JMS spec. Namely, that the message producer (C) conforms to the behavior of a requesting SOAP node and that the message receiver (G) conforms to the behavior of a responding SOAP node. By testing the conformance of the message producer (C), one can ensure that it will interoperate with a

conformant message receiver (G). Likewise, by testing the conformance of the message receiver (G), one can ensure that it will interoperate with a conformant message producer (C).

Test Suite Components

The test suite would consist of the following components:

- A set of web service client applications (A) that sufficiently exercise the vendor's client runtime (B) (including the message producer (C)) to cause various messages to be sent to the request queue.
- A set of web service implementations (endpoints) that will allow the vendor's server runtime (H) (including the message receiver (G)) to be sufficiently exercised.
- A mocked up version of the message producer (C) to be used for testing the vendor's message receiver (G).
- A mocked up version of the message receiver (G) to be used for testing the vendor's message producer (C).

The tests which make up the test suite would consist of a set that focuses on the message producer (C) and a set that focuses on the message receiver (G). If a particular vendor implementation includes only one of these components, then only the set of tests corresponding to that component would need to be executed. Note: this last statement might need to be thrown out if we will not allow a "partial" implementation to be declared "compliant".

Scenario 1 – Testing the vendor's message producer

To test the vendor's message producer (C), various client applications are executed (this could consist of only a single client application which simply invokes all the necessary web service requests). The client application will invoke web service operations which will exercise the client runtime (B) which will, in turn, cause the message producer (C) to send various messages to the request queue. A mocked up version of the message receiver (G) is configured to listen on the queue. Each request message is delivered to the message receiver (G) and it validates each message to ensure that it contains the correct JMS message properties as outlined in the SOAP/JMS spec. For tests involving the request-response MEP, it would also send a reply message back to the destination specified by the request message's JMSReplyTo header (in compliance with the SOAP/JMS spec).

Note that I haven't specified any details of how the JMS message is validated. Some mechanism for examining and validating the messages that does not involve Java code would be preferable. Perhaps we could convert the JMS message into some sort of canonical form (this idea was taken from Amy's proposal) and then compare it to a result which is known to be correct.

Scenario 2 – Testing the vendor's message receiver

To test the vendor's message receiver (G), a mocked up version of the message producer (C) is executed that will send a variety of request messages to the request queue. The

vendor's message receiver (G) is configured to listen on the request queue. It receives each request message and processes it by dispatching to the server runtime (H), which in turn dispatches the request to the web service endpoint (I). For tests involving the request-response MEP, the message receiver (G) would send a reply message to the destination specified in the request message's JMSReplyTo header. The message producer (C) would then receive the reply message and validate it to ensure that it contains the correct JMS message properties as outlined in the SOAP/JMS spec. As in Scenario 1 above, the reply message could be converted by (C) into a canonical form so that it can be compared to a known good result.

Conclusion

Basically, the idea is to separately test the message producer (C) and message receiver (G) by driving the vendor's runtime implementation with various web service invocations, while using a mocked up version of the opposing component. The mocked up version of the opposing component would use the JMS APIs to retrieve the various properties from the message and convert the JMS message into a canonical form that can then be used to validate the message contents.