Experiences with implementing some WS-* specifications

Shrideep Pallickara
Community Grids Lab
Indiana University
spallick@indiana.edu
Outline

- Overview of some Web Service specifications
- Implementation strategy
- Problems encountered
PART – I
WS-Addressing
WS-Addressing

- Web Services Addressing (WSA) provides transport-neutral mechanisms to address Web services and messages.
- WSA provides two very important constructs:
  - endpoint references (EPR) and
  - message information headers (MIH)
- WSA is leveraged by several WS-* specifications

http://www.naradabrokering.org
Endpoint References (EPR)

- Endpoint references are a transport neutral way to identify and describe service instances and endpoints.
  - A typical scenario would involve a node sitting at the edge of an organization, directing traffic to the right instance based on the information maintained in the EPR.
  - EPRs are constructed and specified in the SOAP message by the entity that is initiating the communications.
EPRs – Structure

- An address element which is a URI
- A reference properties element which is a set of properties required to identify a resource
- A reference parameters element which is a set of parameters associated with the endpoint that is necessary for facilitating specific interactions.
Message Information Headers (MIH)

- The MIH enables the identification and location of endpoints pertaining to an interaction.
- The interactions include Request, Reply/Response, and Faults.
Message Information Headers - II

- **To** (mandatory element): This specifies the intended receiver of message.
- **From**: This identifies the originator of a message.
- **ReplyTo**: Specifies where replies to a message will be sent to.
- **FaultTo**: Specifies where faults, as a result of processing the message, should be sent to.
Message Information Headers III

- **Action**: This is a URI that identifies the semantics associated with the message. WSA also specifies rules on the generation of Action elements from the WSDL definition of a service.
  
  - In the WSDL case this is generally a combination of `[target namespace]/[port type name]/[input/output name]`. For e.g. http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe is a valid action element.

- **MessageId**: This is typically a UUID which uniquely identifies a message. This is sometimes also used correlate responses with previously issued requests..

- **RelatesTo**: This identifies how a message relates to a previous message. This field typically contains the messageId of a previously issued message.
WSA Rules

- Identifies how the EPR elements should be added to the Header of the SOAP Message while targeting an endpoint.

- Has rules pertaining to the generation of responses and faults.
  - Contents of the `wsa:RelatesTo` and/or the `wsa:Action` field.

http://www.naradabrokering.org
WSA Rules

- WSA also outlines the rules related to targeting of replies and faults.
  - In the case of faults, it also outlines the content of the `wsa:Action` element.
- It outlines rules related to the generation of the `wsa:RelatesTo` element.
Part -II

WS-Eventing
Overview of Notifications

- Entities communicate through the exchange of messages.

- A **notification** is a message encapsulating an *occurrence of interest* to the entities.

- Notification based systems are an instance of messaging-based systems where entities have two distinct roles viz. source and sink.
Register Interests
(Subscriptions)
Notificaions
Routing Notifications from Source

- A sink first needs to register its interest in a situation, this operation is generally referred to as a subscribe operation.
- The source first wraps occurrences into notification messages.
- Next, the source checks to see if the message satisfies the constraints specified in the previously registered subscriptions.
  - If so, the source routes the message to the sink.
  - This routing of the message from the source to the sink is referred to as a notification.
Loosely-coupled & Tightly-coupled Systems

- Depending on the nature of the underlying frameworks, the coupling between the sources and sinks can vary.

- In loosely-coupled systems, a source need not be aware of the sinks.
  - The source generates events and an intermediary, typically a messaging middleware, is responsible for routing the message to appropriate sinks.

- In tightly-coupled systems, there is no intermediary between the source and the sink.
WS-Eventing

- WS-Eventing is an instance of a tightly-coupled notification system.
  - There is no intermediary between the source and the sink.
    - The source is responsible for the routing of notifications to the registered consumers.

- WS-Eventing, however, introduces another entity — the subscription manager — within the system.
Subscriptions in WS-Eventing

Subscriptions within WS-Eventing have an identifier and expiration times associated with them.

- The identifier uniquely identifies a specific subscription, and is a UUID.
- The expiration time corresponds to the time after which the source will stop routing notifications corresponding to the expired subscription.

Also specifies the dialect (XPath, Regular expressions etc) and the constraint associated with the subscription.
A subscription manager is responsible for operations related to the management of subscriptions.

Every source has a subscription manager associated with it.

The specification does not either prescribe or prescribe the collocation of the source and the subscription manager on the same machine.
Subscription Manager Operations

- It enables sinks to retrieve the status of their subscriptions. These subscriptions are the ones that the sinks had previously registered with the source.
- It manages the renewals of the managed subscriptions.
- It is responsible for processing unsubscribe requests from the sinks.
WS-Eventing Entity Interactions I

Source
Subscription Manager
Sink

getStatus
Renew
Unsubscribe

Subscribe
Subscription End
Notifications

http://www.naradabrokering.org
WS-Eventing Entity Interactions - II

- When the sink subscribes with the source, the source includes information regarding the subscription manager in its response.
- Subsequent operations — such as getting the status of, renewing and unsubscribing — pertaining to previously registered subscriptions are all directed to the subscription manager.
- The source sends both notifications and a message signifying the end of registered subscriptions to the sink.
Part III
WS-ReliableMessaging
WSRM describes a protocol that facilitates the reliable delivery of messages between two web service endpoints in the presence of component, system or network failures.

WSRM facilitates the reliable delivery of messages from the source (or originator) of messages to the sink (or destination) of messages.

The delivery (and ordering) guarantees are valid over a group of messages, which is referred to as a sequence.
Creation of Sequences

- In WSRM prior to ensuring reliable delivery of messages between the endpoints, the source initiates an exchange with the sink pertaining to the creation of a Sequence.
- This Sequence is intended to facilitate the grouping of a set of related messages.
- This Sequence is identified by an identifier, typically a UUID. Other information associated with the Sequence include information regarding:
  - The source and the sink
  - Policy information related to protocol constants such as acknowledgement and retransmission intervals.
  - Security related information if needed.
WSRM Sequences

- In WSRM all messages issued by a source exist within the context of a Sequence that was established prior to communications.
- Once a source has determined that all messages within a Sequence have been received at the sink, the source initiates an exchange to terminate this sequence.
- The specification allows for a maximum of $2^{64} - 1$ messages within a Sequence.
- The specification places no limits on the number of Sequences between a specific source and sink.
  - However, it is expected that at any given time there is NO more than 1 active Sequence between 2 specific endpoints.
Publishing Messages in WSRM

- Every message from the source contains two pieces of information —
  - The Sequence that this message is a part of and
  - A monotonically increasing Message Number within this Sequence.

- These Message Numbers enable the tracking of problems, if any, in the intended message delivery at a sink.
  - Message Numbers enable the determination of out of order receipt of messages as well as message losses.
Issuing Acknowledgments

- In WSRM a sink is expected to issue acknowledgements back to the source upon receipt of messages.
- This acknowledgement contains information about
  - The Sequence and
  - The Message Numbers within this Sequence.
- An acknowledgement must be issued only after a certain time — the acknowledgement interval — has elapsed since the receipt of the first unacknowledged message.
- This acknowledgement may cover a single message or a group of messages within a Sequence.
Processing Acknowledgments

- Upon receipt of this acknowledgement a source can determine which messages might have been lost in transit and proceed to retransmit the *missed* messages.

- Thus if a sink has acknowledged the receipt of messages 1 – 10 and 13 – 18.
  - The source can conclude that messages with Message Numbers 11 and 12 were lost en route to the sink and proceed to retransmit these messages.
Retransmissions and Error Corrections

A source may also pro-actively initiate the retransmission of a message for which that an acknowledgement has not been received within a specified time — the retransmission interval — after which it was issued.

In WSRM error corrections can also be initiated at the sink; this is done through the use of negative acknowledgements.

Negative acknowledgments identify the message numbers that have not been received at a sink.

Message Numbers increase monotonically. If Message Numbers 1, 2, 3, 4 and 8 within a specific Sequence have been received at a sink.

This sink can easily conclude that it has not received messages with message numbers 5, 6 and 7 from the source.
Notification of Errors

- WSRM provides for notification of errors in processing between the endpoints involved in reliable delivery.
  - These are routed back as SOAP Faults.
- The range of errors can vary from an inability to decipher a message’s content to complex errors pertaining to violations in implied agreements between the interacting source and sink.
- All errors are reported as faults with the appropriate `wsa:Action` attribute, and encapsulated in WSRM fault elements.
Part IV
Reflections on Implementing WS-* specifications
Some quick observations about WS-*

- Typically addresses core areas or those where the demand is substantial enough to eschew proprietary ad hoc solutions.

- In some cases common issues across various WS-* specifications mandate additional WS-* specifications.
  - Exemplars include WS-Addressing, WS-Policy.

- Various specifications are intended to incrementally augment capabilities at an endpoint.
  - For e.g. if you need reliable messaging capabilities simply plug in a WSRM module. If you need notification capability plug in WS-Eventing or WS-Notification.
Some quick observations about WS-*

- Functionality of specifications encapsulated within stand-alone SOAP messages.
  - They typically also include a WSDL definitions of operations, but all functionality is encapsulated in SOAP messages.

- Primary interaction model is one-way, asynchronous SOAP messaging.

- Lot of these specifications are also intended to be stackable.
WS-*
WS-Eventing
WS-RM
WS-Security
XML Schema
SOAP Rules
WS-Addressing
WS-Policy
Typical implementation strategy

- Develop strategy for processing the XML Schema associated with the specification.
  - This would allow you to process the XML messages from Java (or language of choice).
  - XML generated over the wire should be conformant to the relevant schemas.
    - This allows one to interact with other implementations.

- Develop Processor to enforce rules and processing associated with the specification.
  - This would include performing actions, issuing requests/responses and faults.

- Ensure that rules and processing related to leveraged specifications are enforced.
We were looking for a solution that allowed us to process XML from within the Java domain.

There are 4 choices

- Develop Java classes ourselves
- Use wsdl2java to do this
- Use the JAXB Data Binding Framework
- Use a schema compiler such as Castor or XMLBeans
Writing one’s own classes

- Approach used by Apache’s Sandesha project.
  - Implementation of WSRM

- Error prone and quite difficult
  - Increasingly the developer has to deal with several other specifications.

- Another approach is to just process messages based on DOM.
  - Quite difficult to do. No examples that we are currently aware of.

http://www.naradabrokering.org
WSDL2Java Problems

- Issues (in version 1.2) related to this tool’s support for schemas have been documented in http://www-106.ibm.com/developerworks/webservices/library/ws-castor/.

- Specifically, the problems relate to insufficient (and in some cases incorrect) support for complex schema types, XML validation and serialization issues.
JAXB Issues

- JAXB is a specification from Sun to deal with XML and Java data-bindings.
- JAXB though better than what is generated using Axis’ `wsdl2java` still does not provide complete support for the XML Schema.
  - JSR 31 expert group decided NOT to attempt full compatibility with the XML Schema standard.
  - You may run into situations where you may find an inaccessible data inside your schema.
- We looked at both the JAXB reference implementation from Sun and JaxMe from Apache (which is an open source implementation of the JAXB Framework).

http://www.naradabrokering.org
Rationale for the choice of XMLBeans

- We settled on XMLBeans because of two reasons.
  - It is an open source effort. Originally developed by BEA it was contributed by BEA to the Apache Software Foundation.
  - In our opinion, it provides the best and most complete support for the XML schema of all the tools currently available.
- XMLBeans allows us to validate instance documents and also facilitates simple but sophisticated navigation through XML documents.
- The XML generated by the corresponding Java classes is true XML which conforms to (and can be validated against) the original schema.

http://www.naradabrokering.org
Developing WS-* Processors

- In some cases there would be more than one role associated with a specification. Ensure that processing related to each role is done.
  - E.g. WSRM and WSE.

- Processing the SOAP Messages
  - Direction of the message is important.
    - Messages processed differently depending on whether it was received over the network or from application.

- When problems are encountered the processor needs to throw exceptions and/or issue faults.
  - Faults need to conform to rules outlined in both SOAP and WS-Addressing.

http://www.naradabrokering.org
Common Problems

- Schemas seem to change quite often.
- WS-* specifications typically leverage other specifications.
  - Changes in the schemas of these specifications can affect the one being implemented.
- Container problems
WS-* specs have compatibility issues

- Every new version has a new target namespace.
- This ensures that an implementation of a specific version of a specification will ONLY work with other implementations of the same version.
- This is the equivalent of having a new package name for every class every time you release a new version of your software.
  - Applications developed using the old class names will not work without major updates.
- You can generate classes for every version of the spec.
  - Lots and lots of duplicate classes. Code re-use and manageability is sacrificed.

http://www.naradabrokering.org
A General comment on Web Service Containers

- It is based on supporting the RPC model which is out-of-sync with several new WS specifications.
  - Every message needs to be a request (invocation of a remote method) or the corresponding response.
  - Focal point is WSDL not SOAP.
    - Similar to IDL-centric CORBA approach.
  - SOAP 1.2 clearly states primary purpose is one-way messaging NOT the carrying of RPC invocations.

- Problems
  - It is difficult to fit the RPC model for say WS-Eventing
    - Where would a RPC request (notification) be made? There are multiple destinations that the message needs to be sent to.
  - Forcing every exchange, ACKs/Retransmissions to be based on the RPC request/response model is very limiting.
The Handler Approach

- A handler is a module that can reside in the processing path of SOAP messages as they traverse between service endpoints.
- A handler may be configured to reside in the paths associated with the message exchanges such as requests, responses or faults.
- Handlers facilitate the incremental addition of capabilities to services.
  - No need to change service implementations themselves.
- Handlers are autonomous entities
  - A given handler has complete access to the entire SOAP message. Read, Write, and Replace.
- Handlers can be cascaded to form Handler chains
Handlers: Problems

- Handlers are statically pre-configured.
  - No dynamic re-configuration of the handler-chain.

- Handlers cannot pro-actively inject messages into the processing path between the service endpoints.
  - In WSRM, a node needs to issue acknowledgements or initiate retransmissions at regular intervals.
  - Sometimes, a given SOAP message may result in multiple SOAP messages being forked off.
    - In WS-Eventing a message may need to be routed to multiple interested consumers for that message.
  - The current handler model precludes us from easily supporting these scenarios.

- The handler model in the proposed JAX RPC 2.0 specification does not address these issues.
  - We hope Axis will incorporate support for richer interactions.

http://www.naradabrokering.org
Handlers: What they should support

- Injecting messages
  - A given notification message may spawn multiple copies of the message being routed to consumers.
  - In WSRM, this would facilitate retransmissions and responses.

- Ability to generate responses and stop message processing
  - Some messages need not be propagated to application at all. For e.g. an application need not know about WSRM acks or naks.
In WS-Eventing: You do not want a notification message to traverse H1, H2, H3. In WSRM you do not wish for retransmissions to be processed by H1, H2, H3.

But messages need to traverse H5 and H6.
Final Comments

- WS-* specifications still have some way to go.
  - Stability and compatibility
- WS Containers need some changes
  - Better support for the one-way messaging model
- Handler/Filter model needs to be richer.