

# Reliable Messaging Specs for Web Services: Analysis and Comparison \*

Alejandro Houspanossian, Cristian Fiorentino, Juan Pablo Carlino  
UNICEN, Faculty of Sciences  
Campus Universitario  
{ahouspan, cfioent, jcarlino}@exa.unicen.edu.ar

May 2006

## Abstract

Currently there is a growing trend towards using Web services to integrate heterogeneous systems. However, in order to use web services in real world applications, a key aspect must be addressed: *reliable messaging*. Two similar specifications, WS-Reliability and WS-ReliableMessaging, have emerged to undertake this issue, but confusing the community at the same time. In this work we provide an analysis and comparison of these two competing specifications, and related information in the context of web services, message-oriented middleware and enterprise integration.

---

\*Course: Digital Interaction Among Applications. Prof. Mariano Cilia. Feb/2006. Tandil, Argentina.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Reliable Messaging Requirements . . . . .	4
2.2	State of the Art . . . . .	5
2.3	Message-Oriented Middleware, Enterprise Service Bus, and WS-Rel* Specifications . . . . .	5
<b>3</b>	<b>WS-Reliability (WS-R)</b>	<b>7</b>
3.1	Scope of WS-Reliability . . . . .	7
3.2	Relation with Other Specs . . . . .	7
3.3	The WS-R Model . . . . .	7
3.4	SOAP Message Exchange Patterns (MEP) . . . . .	8
3.5	Message Reply Patterns . . . . .	9
3.6	Reliability Agreements . . . . .	10
3.7	Additional Considerations . . . . .	10
3.8	Wrap-up Example . . . . .	11
<b>4</b>	<b>WS-ReliableMessaging (WS-RM)</b>	<b>13</b>
4.1	Scope . . . . .	13
4.2	Relation with Other Specs . . . . .	13
4.3	Reliable Messaging Model . . . . .	14
4.4	Protocol Preconditions . . . . .	14
4.5	Protocol Elements . . . . .	15
4.6	Faults . . . . .	15
4.7	Security . . . . .	16
4.8	Final Remarks . . . . .	17
4.9	Wrap-up Example . . . . .	18
<b>5</b>	<b>WS-Reliability vs. WS-ReliableMessaging</b>	<b>21</b>
<b>6</b>	<b>Implementations</b>	<b>24</b>
6.1	WS-ReliableMessaging . . . . .	24
6.2	WS-Reliability . . . . .	25
6.3	Other Initiatives . . . . .	25
<b>7</b>	<b>Reliability in Notification Services (NS)</b>	<b>27</b>
7.1	WS-Notification (WS-N) . . . . .	27
7.2	Java Message Service (JMS) . . . . .	28
7.3	NS Reliability Conclusions . . . . .	29
<b>8</b>	<b>Conclusions</b>	<b>30</b>
8.1	Looking for <i>the</i> Standard . . . . .	30
8.2	What's next? . . . . .	31

# 1 Introduction

SOA (Service Oriented Architecture) provides an standard-based interoperable and loose-coupled integration model, and is seen as the next evolutionary step for software development [10]. In SOA-based solutions, discoverable, interchangeable and composable units of functionality are exposed as *services*<sup>1</sup>. In SOA, three main roles are recognized: *service providers*, *service requestors* and *service registries*. Basically service providers use the registry to publish or advertise their services. Service requestors access the registry to get information about services and providers (this includes binding information). SOA is important because it addresses one of the major challenges on the current IT-World: heterogeneous systems integration.

Web services (WSs) technology is SOA's best known infrastructure. It is currently in the way of consolidation, and lies on three main pillars: WSDL, SOAP and UDDI specifications. WSDL specifies an XML-based language for describing service interfaces. SOAP is a platform-and-language-agnostic messaging protocol. The UDDI specification defines a set of services and a data model to represent, publish and locate information about services and services providers.

The SOAP standard is independent of the transport protocol. However, SOAP messages are usually sent over HTTP (SOAP/HTTP). In fact, the only binding defined in the SOAP specification is to HTTP. HTTP is a widely supported transport protocol. In this way, Web services gain in terms of pervasiveness and interoperability. However, HTTP is not reliable.

In order to use WSs in real-world business applications, *reliability* must be provided. Different approaches for addressing these issues have been proposed and developed. In this work we provide an overview of the actual state of art, focusing on two proposed specifications: WS-Reliability (WS-R) and WS-ReliableMessaging (WS-RM). The ultimate goal of this work is to analyze and compare both initiatives.

To do so, first we set the stage by presenting background information (Section 2). Later we present relevant information about both WS-R (Section 3) and WS-RM (Section 4) specifications. Afterwards, we are able to give our analysis, presenting similarities and differences found among these two specifications (Section 5). We provide an overview of existent implementations and other related initiatives in Section 6. In section 7 we relate the specifications presented in sections 3 and 4 (referred in combination as WS-Rel\* specifications) with Notification Services specifications in the context of MOM. Finally, we present our conclusion in Section 8.

---

<sup>1</sup>Implementation and platform details are hidden behind (service) interfaces, which are defined using standard description languages. The same interface can actually be implemented by many systems, with different technologies and in the context of different organizations and execution platforms.

## 2 Background

In this section we introduce the problem of unreliable messaging in the context of WSs technology.

One of the main features of web services is that they are based on globally accepted (web) standards. It accounts for interoperability, and is the key to integrate heterogeneous systems in both Enterprise Application Integration (EAI) and Business-to-Business (B2B) contexts. In this way, most of web service implementations work on top of HTTP. That is, they send SOAP messages over HTTP.

HTTP is a globally supported transport protocol but it is not reliable. HTTP does not guarantee that messages are going to be delivered *once* and *without duplicates*. One of the options is to support reliable messaging at the application level. It means that the *logic* that provides for reliability is embedded in the applications. The problem with this approach is that it leads to non-interoperable solutions. In the context of WSs, this solution must be avoided. Other approaches propose, for instance, the extension of HTTP to make it reliable. Once again, this solution would work only between systems that understand the extended HTTP protocol.

These and other solutions will be analyzed further in this work. However, what should be recognized now is that the solution should be based on accepted standards.

*Without a Web services standard for providing reliable message delivery, applications will implement the necessary function in their business logic. This requirement places a burden on developers of business logic, but more importantly impedes interoperability because of inconsistent, differing solutions to a common problem. [3]*

Coming back to SOAP, we have already mentioned that it is transport-protocol agnostic. In principle, SOAP messages can be sent over JMS (or other communication technology) as much as over HTTP. However, the SOAP specification defines one single binding: SOAP/HTTP. Because of this, the broad adoption of this *platform*. SOAP over JMS (SOAP/JMS) has also advantages as well as disadvantages. Basically, JMS provides a reliable messaging model, but its use is far from being so widespread as HTTP.

In the rest of this section we introduce the following topics:

1. the requirements on reliable messaging
2. the state of the art on reliable messaging
3. reliable messaging in the context of MOM

### 2.1 Reliable Messaging Requirements

Several approaches to reliable messaging have been proposed. In [8], the author mentions two of these approaches, which attempt to provide for reliability by establishing a reliable transport protocol, over which SOAP messages are exchanged. In the first case, reliability is achieved by applying queuing technologies. That is to say, applications send (and read) messages to (and from) queues. In the second case, the attempt is based on adding reliability to fundamentally unreliable transport protocols (like HTTP). HTTPR (Reliable HTTP) adds a layer to HTTP that assures that a message will be delivered to its destination application *exactly once*, or it will be reliably reported as *undelivered*. As the author in [8] claims, the problem of both approaches is that they jeopardize or compromise the interoperability features expected from Web services. In both cases, sender and receiver applications must support the same transport protocol. WSs should be transport-protocol agnostic. In support of the Enterprise Service Bus, for instance, where messages can be potentially

sent through multiple hops before they reach destination, and where the intermediate hops are not necessarily known *a priori*, independence of the transport protocol is key.

In the following, a list of the key requirements on reliable messaging is presented. This list is extracted from [8]:

1. Support carrying message traffic reliably in support of business processes whose lifetimes commonly exceed the up times of the components on which these processes are realized.
2. Support quality-of-service assertions such as:
  - Each message sent be received exactly once (once and only once), at most once, at least once, and so on.
  - Messages be received in the same order in which they were sent
  - Failure to deliver a message be made known to both the sender and receiver
3. Accommodate mobility of a reliable business process to different channels or physical machines.
4. Support message transfer via intermediaries.
5. Leverage the SOAP extensibility mechanism to achieve reliable messaging.
6. Enable reliable messaging bindings to a variety of underlying reliable and unreliable transport protocols together with the Message Routing Protocol.
7. Compose with other protocols to support security and other message delivery services.

## 2.2 State of the Art

Currently there are two specifications that are fighting to become *the* standard for reliable messaging. These are: WS-Reliability and WS-Reliable Messaging. Both initiatives address the issue of reliable web services.

### Recent Versions of WS-Reliability and WS-ReliableMessaging

- Twenty-five months after the first public announcement for the Web Services Reliable Messaging Protocol (WS-ReliableMessaging), the co-authors BEA Systems, IBM, Microsoft, and TIBCO have announced that the WS-ReliableMessaging and WS-RM Policy specifications will be submitted to OASIS for further refinement and finalization as a Web services standard (April 2005).
- WS-Reliability v1.1 was approved as an OASIS Standard on November 15, 2004. Main supporters of this spec are Fujitsu Limited, Hitachi, Ltd., NEC Corp, Oracle Corp., Sonic Software, and Sun Microsystems.

## 2.3 Message-Oriented Middleware, Enterprise Service Bus, and WS-Rel\* Specifications

One of the main approaches to systems integration is based on messaging<sup>2</sup> infrastructures, which are in charge of managing the flow of messages among disparate applications. Basically, it works

---

<sup>2</sup>Messaging is a technology that enables high-speed, asynchronous, program-to-program communication with reliable delivery[12].

by decoupling producer and consumer applications. To do so, messages are published to and consumed from a common channel (mediator). The applications must agree on a channel as well as the format of the message. Message delivery is addressed by Messaging Systems or Message-Oriented Middleware (MOM) in an environment as unreliable as computer networks. Support for reliability is one of the requirements on a messaging system.

Of course, when adopting a MOM solution, the same (messaging) protocol must be understood by all the participating applications. The use of proprietary approaches may be reasonable in the context of a single enterprise (EAI), but it is not adequate in a more heterogenous and autonomous environment like the one involving digital interaction among partner companies (e.g. B2B). The Enterprise Service Bus (ESB) is a concept (or model) developed to address this last issue. ESB combines messaging, Web services, data transformation and intelligent routing. ESB can be seen as an interconnected grid of messaging systems [1].

In the context of ESB, reliable messaging must be assured in particular in those cases where the two endpoints (of a point-to-point communication) cross the boundaries of an independent organization. It implies that proprietary approaches to assuring reliable messaging (which are part of -proprietary- MOMs already) should be somehow complemented with canonical or standard protocols. Figure 1 illustrates the cases of internal and external traffic. Proprietary approaches could be well suited for the internal traffic (due to performance aspects). However, for the external traffic, an interoperable (i.e. standard-based) open protocol for reliable messaging is required. In this work we present two competing Web services specifications that are proposed to fill this gap<sup>3</sup>.

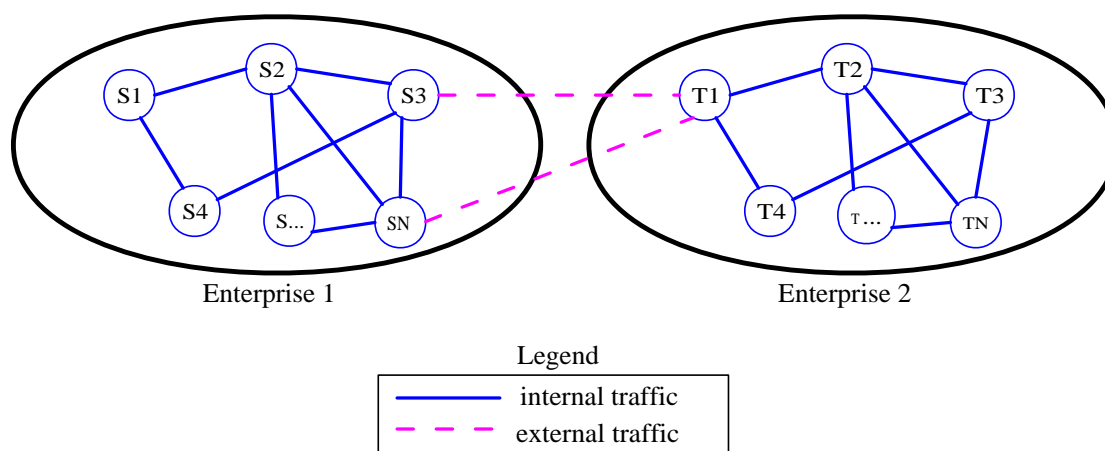


Figure 1: MOM and Reliable Messaging

Specialists agree [1] on the *80/20 rule*, which stands for 80% of internal traffic against 20% of external. Because of performance reasons, the use of native approaches are preferred to standard solutions whenever organizational boundaries are not crossed.

<sup>3</sup>Of course, both WS-Reliability and WS-ReliableMessaging can be used to add reliability to any application of WSs technology.

### 3 WS-Reliability (WS-R)

WS-Reliability is supported by Fujitsu Limited, Hitachi, Ltd., NEC Corp, Oracle Corp., Sonic Software, and Sun Microsystems. The purpose of the specification is to define *reliability* in the context of current Web Services standards. In WS-Reliability [13], *Reliable Messaging (RM)* is defined as:

*The execution of a transport-agnostic, SOAP-based protocol providing quality of service in the reliable delivery of messages.*

To this regard, two aspects (to be addressed) are recognized. On the one hand, the *wire aspect*, which is used to communicate messaging service's message processor nodes (sending and receiving RMPs<sup>4</sup>). It encompasses message headers and interchange protocols at the wire level. On the other hand, the *Quality of Messaging Service Aspect*, defines a protocol between the user of the messaging service and the messaging service. It represents the relationships of the application layer with the reliable messaging middleware. This protocol is defined by a set of *abstract operations*: *Submit, Deliver, Notify, Respond*.

#### 3.1 Scope of WS-Reliability

WS-Reliability defines a protocol for exchanging SOAP messages reliably. Reliability features are specified in SOAP headers independently of the underlying protocol. The specification also contains a binding to HTTP.

Some messaging features are seen as orthogonal to the specification and are out of scope for WS-Reliability: *Routing* - the specification addresses end-to-end reliability (RMP-to-RMP) and is not concerned with intermediaries) - and *transactions*.

#### 3.2 Relation with Other Specs

**W3C SOAP1.1/1.2** WS-Reliability defines reliable messaging protocol embedded in the SOAP Header. SOAP1.1 and SOAP1.2 are the base protocols.

**OASIS ebXML Message Service Specification 2.0** WS-Reliability borrows the reliable message mechanism defined in the ebXML Message Service Specification 2.0<sup>5</sup>.

**OASIS WS-Security** WS-Reliability defines reliability independently from security. WS-Security, which also defines a mapping to SOAP headers, can be used.

**WS-I Basic Profile 1.0** This specification is compliant with WS-I Basic Profile 1.0a<sup>6</sup> for use of other technologies including SOAP, WSDL, and XML schema.

#### 3.3 The WS-R Model

The WS-Reliability specification considers a set of abstract operations that are used to model reliability contracts between the (reliable) messaging middleware (i.e. *Sending* and *Receiving RMPs*) and its users (i.e. Producers and Consumers of messages). The operations are listed bellow (see also Fig. 2):

---

<sup>4</sup>In WS-Reliability, RMP stands for Reliable Messaging Processor. RMPs are modules that enforce *reliable messaging*. RMPs constitute the reliable messaging middleware.

<sup>5</sup>OASIS ebXML Messaging Services T. C. 4/2002. Available at: <http://www.ebxml.org/specs/ebMS2.pdf>

<sup>6</sup>Basic Profile Version 1.1. 8/2004. Available at: <http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-07-21.html>

**Submit** used by the application layer to send a message reliably

**Deliver** used to deliver the payload to the (receiving) application layer

**Respond** used by the (receiving) application layer to route a response

**Notify** used to notify to a producer of an error, or to transferring a response.

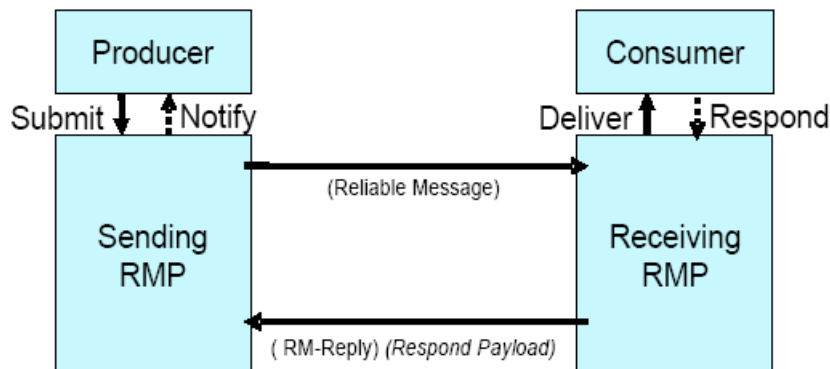


Figure 2: Abstract Operations

Acknowledgment indications (ACKs) are signaled when messages are successfully delivered (it implies the successful execution of the *Deliver* operation).

Currently the specification defines the following reliability features:

- Guaranteed message delivery (At-Least-Once) delivery semantics
- Guaranteed message duplicate elimination (At-Most-Once)
- Guaranteed message delivery and duplicate elimination (Exactly-Once)
- Guaranteed message ordering for delivery within a group of messages

### 3.4 SOAP Message Exchange Patterns (MEP)

WS-Reliability defines two exchange patterns: One-way and Request-response. These patterns must be supported by the reliable messaging middleware. In other words, Sending and Receiving reliable message processors (RMPs)<sup>7</sup>. Support for these patterns is translated into a list of assumed abilities.

#### 1. SOAP One-way MEP:

- (a) the *sender* can initiate a communication (and send SOAP messages).
- (b) no response containing a SOAP envelope is sent back although a non-SOAP response (e.g., an HTTP error code) may be returned.

#### 2. SOAP Request-response MEP:

<sup>7</sup>In the rest of this section the terms *sender* and *receiver* are used to refer respectively to *Sending* and *Receiving* RMPs as defined in [13] (and not to application level entities)



- (a) the *sender* is able to send a SOAP message.
- (b) the *receiver* can send back a message with a SOAP envelope (response).

### 3.5 Message Reply Patterns

In order to provide for reliability, whenever appropriate, acknowledgments and fault indication must be routed back to the message sender. In WS-Reliability it is translated to the publishing of message replies (RM-Reply). To this regard the specification considers three patterns:

#### 1. Response RM-Reply Pattern:

- (a) the *sender* sends the request using the request-response pattern.
- (b) the *receiver* sends the reply (RM-Reply) as part of the response (*piggybacking*)

#### 2. Callback RM-Reply Pattern:

- (a) the *sender* sends the message according to the pattern required at the application level (Request-response or One-way).
- (b) the *receiver* MUST send the RM-Reply using a *SOAP One-way MEP*.

#### 3. Poll RM-Reply Pattern:

- (a) the *sender* sends the message (the pattern may be either Request-response or One-way).
- (b) the *sender* asks for Acknowledgment (Polling). The request (<PollRequest> element) can be done synchronously (the request is sent in a SOAP Request-response pattern). Or asynchronously: the request must be sent as a SOAP One-way pattern.
- (c) the *receiver* sends the RM-Reply either (if synchronous polling) in the response message of the same SOAP instance that carried the PollRequest or (if asynchronous polling) in a message from a SOAP One-way MEP instance. This message MUST NOT contain a payload.

Listing 1: Specifying Reply Patterns

```

1 <soap:Envelope ... >
  <soap:Header>
3     <Request ... >
4         ...
5         <ReplyPattern>
6             <Value> Callback </Value>
7             <ReplyTo>
8                 <BareURI> http://wsr-sender.org/abc/wsrmlistener </BareURI>
9             </ReplyTo>
10        </ReplyPattern>
11        ...
12    </Request>
13 </soap:Header>
14 <soap:Body>
15     ...
16 </soap:Body>
17 </soap:Envelope>

```

Reply patterns are specified with the <ReplyPattern> construct, inside <Request> tags (see listing 1). Possible values for <ReplyPattern> are: *Response*, *Callback* and *Poll*. In the case of the

*Callback pattern*, the element `Request/ReplyPattern/ReplyTo` must be also defined. The usual content of this element is an URI (by default the type is `xs:anyURI`). By setting the property `ReplyTo@reference-scheme`, other reference schemes (supported by the receiving RMP) can be used.

### 3.6 Reliability Agreements

The specification defines the term *reliability agreement*. An *agreement* describes the reliability features that are to be used during the exchange of reliable messages. It is abstractly represented as a list of *agreement items*. An agreement embraces the two axes of contracts referred before in page 7: wire protocol and quality of messaging service.

The *agreement items* considered in the specification (and their possible values) are presented next.

- **GuaranteedDelivery** (enabled/disabled): for setting *Guaranteed Delivery*.
- **NoDuplicateDelivery** (enabled/disabled): for setting message delivery without duplicates, or *Duplicate Elimination*.
- **OrderedDelivery** (enabled/disabled): for setting *Guaranteed Message Ordering*.
- **GroupMaxIdleDuration** (number of seconds): For setting the elapsed time limit from the last message sent or received in a group, after which the group can be terminated. More about message grouping in the coming sub-section.
- **GroupExpiryTime** (number of seconds): For setting the date and time after which the group can be terminated.
- **ExpiryTime** (number of seconds): For setting the date and time after which a message must not be delivered to the receiving application.
- **RetryMaxTimes** (integer number): For setting the maximum number of times a message must be resent if not acknowledged.
- **RetryTimeInterval** (number of seconds): For setting the minimal elapsed time between two re-sending of the same message.
- **ReplyPattern** (Response, Callback, Poll): For setting the mode of response for acknowledgments and faults.

The specification defines that, in order to enforce the reliability requirements, the only input the *receiver* will need, will be obtained from the header elements of received messages. Only the *sender* needs to have knowledge of the *RM Agreement* initially (whether by configuration, an API call, a message, the result of an algorithm or any other means). It is important to mention that the concrete representation of the agreements is currently out of the scope of the specification (will be addressed in future versions).

### 3.7 Additional Considerations

**Message Identification and Grouping** Messages always have a group. Each group has a global identifier. Unique sequence numbers are used to identify messages inside a group. In other words, message identifiers are a combination of group identifier and sequence number. Actually, the sequence number is optional. It can be omitted when there is only one message in the group.

**Fault Codes** The specification defines two kinds of faults: *message format* and *message processing* faults. *Message Format Faults* are: `InvalidRequest`, `InvalidPollRequest`, `InvalidMessageId`, `InvalidMessageParameters`, `InvalidReplyPattern` and `InvalidExpiryTime`. The reader is referred to the specification [13] for a deeper description (and causes) of these faults. *Message Processing Faults* are: `FeatureNotSupported` (e.g. an message with a `<MessageOrder>` element sent to a Receiving RMP that does not support *Guaranteed Message Ordering*), `PermanentProcessingFailure` (which indicates that the failure is fatal and subsequent retries of the same message will also fail), `MessageProcessingFailure` (the one that indicates a transient fault; the message may succeed after a subsequent retry) and `GroupAborted` (all processing for the group associated with the reliable message request has been aborted by the Receiving RMP).

### 3.8 Wrap-up Example

The HTTP message below (listing 2) uses the `<Request>` to specify the reliability features (lines 6-20). The required features (lines 17-19) are: *GuaranteedDelivery* (`<AckRequested>` element), *NoDuplicateDelivery* (`<DuplicateElimination>` element), and *OrderedDelivery* (`<MessageOrder>` element). The reply pattern (lines 14-16) is *Poll*, meaning that no *Acknowledgment* or *Fault* will be sent back unless explicitly requested by another message containing a `<PollRequest>` header.

Listing 2: Example of a Reliable Message over HTTP

```

1 POST /abc/servlet/wsrEndpoint HTTP/1.0
2 Content-Type: text/xml; charset=utf-8
3 Host: 192.168.183.100
4 SOAPAction: ""
5 Content-Length: 736
6 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
7   <soap:Header>
8     <Request xmlns="http://docs.oasis-open.org/wsr/2004/06/ws-reliability-1.1.xsd"
9       soap:mustUnderstand="1">
10       <MessageId groupId="mid://20040202.103832@wsr-sender.org">
11         <SequenceNum number="0" groupExpiryTime="2005-02-02T03:00:33-31:00" />
12       </MessageId>
13       <ExpiryTime>2004-09-07T03:01:03-03:50</ExpiryTime>
14       <ReplyPattern>
15         <Value>Poll</Value>
16       </ReplyPattern>
17       <AckRequested/>
18       <DuplicateElimination/>
19       <MessageOrder/>
20     </Request>
21   </soap:Header>
22   <soap:Body>
23     <Request xmlns="http://example.org/wsr">Request Message</Request>
24   </soap:Body>
25 </soap:Envelope>

```

Eventually, the *sender* will poll the receiver for acknowledgements or fault indications. It is done by sending a `<PollRequest>`. This request can reference different groups of messages and also message sequences in a group. Listing 3 provides an example of this.

Finally, the replies (acknowledgments or fault indications) are sent. In the case that the messages of a group have a sequence number, the response must contain the `<SequenceReplies>` tag, and the corresponding reply ranges (`<ReplyRange>` tags). The `<NonSequenceReply>` element is used to send replies for those messages that do not have a sequence number. If the `<NonSequenceReply>` tag does not include a fault indication, it is considered an acknowledgment indication. An example is presented in listing 4.

### Listing 3: Poll Request Example

```
1 POST /abc/servlet/wsrEndpoint HTTP/1.0
3 Content-Type: text/xml; charset=utf-8
Host: 192.168.183.100
5 SOAPAction: ""
Content-Length: 432
7 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
9     <PollRequest
        xmlns="http://docs.oasis-open.org/wsr/2004/06/ws-reliability-1.1.xsd"
11     soap:mustUnderstand="1">
        <RefToMessageIds groupId="mid://20040202.103832@wsr-sender.org">
13             <SequenceNumRange from="0" to="20"/>
        </RefToMessageIds>
15     <RefToMessageIds groupId="mid://20040202.103811@wsr-sender.org" />
        </PollRequest>
17     </soap:Header>
        <soap:Body />
19 </soap:Envelope>
```

### Listing 4: Response Example

```
HTTP/1.0 200 OK
2 Server: WS-ReliabilityServer
Date: Mon, 02 Feb 2004 10:38:32 GMT
4 Content-Language: en
Content-Type: text/xml; charset=utf-8
6 Content-Length: 593
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
8   <soap:Header>
        <Response soap:mustUnderstand="1"
10     xmlns="http://docs.oasis-open.org/wsr/2004/06/ws-reliability-1.1.xsd">
        <NonSequenceReply groupId="mid://20040202.103811@wsr-sender.org"/>
12     <SequenceReplies groupId="mid://20040202.103832@wsr-sender.org">
        <ReplyRange from="0" to="14"/>
14     <ReplyRange from="15" to="15" fault="InvalidRequest"/>
        <ReplyRange from="16" to="20"/>
16     </SequenceReplies>
        </Response>
18     </soap:Header>
        <soap:Body />
20 </soap:Envelope>
```

## 4 WS-ReliableMessaging (WS-RM)

WS-ReliableMessaging is a protocol standard that addresses reliable messaging. It emerged from the joint efforts of IBM, BEA Systems, Microsoft and TIBCO Software representatives.

The primary goal of WS-RM specification is to create a modular mechanism for reliable message delivery. It defines a messaging protocol to identify, track, and manage the reliable delivery of messages between exactly two parties, a source and a destination. It also defines a SOAP binding that is required for interoperability. Additional bindings may be defined. This mechanism is extensible allowing additional functionality, such as security, to be tightly integrated. WS-RM integrates with and complements the WS-Security, WS-Policy, and other Web services specifications. Combined, these allow for a broad range of reliable, secure messaging options.

### 4.1 Scope

WS-ReliableMessaging provides an interoperable protocol that a Reliable Messaging (RM) Source and Reliable Messaging (RM) Destination use to provide Application Source and Destination a guarantee that a message that is sent will be delivered. It is the responsibility of the RM Source and RM Destination to fulfill these guarantees, or raise an error. The protocol allows endpoints to meet this guarantee for the delivery assurances explained on subsection 4.3. Persistence considerations related to an endpoint's ability to satisfy these delivery assurances are the responsibility of the implementation and do not affect the wire protocol. As such, they are out of scope of WS-RM.

WS-RM is described in a transport-independent manner allowing it to be implemented using different network technologies. To support interoperable Web services, a SOAP binding is defined within the specification.

The protocol depends upon other Web services specifications for the identification of service endpoint addresses and policies. How these are identified and retrieved are detailed within those specifications and are out of WS-RM scope [4].

### 4.2 Relation with Other Specs

The architecture of WS-ReliableMessaging supports composition with other messaging and Web service specifications and standards. This architecture consists of the following specifications [20]:

**WS-ReliableMessaging** A protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures.

**WS-Addressing** A framework to identify Web service endpoints and to ensure end-to-end endpoint identification in messages.

**WSDL** A set of constructs to specify Web service interfaces and bindings for endpoints.

**WS-Policy** A base set of constructs that can be used and extended by other Web services specifications to describe a broad range of requirements, preferences, and capabilities of service interfaces.

**WS-Transactions and WS-Coordination** A set of Web service interface definitions and protocols that support participant control and agreement on the outcome of distributed, multi-party interactions.

**WS-EndpointResolution** A set of Web service mechanisms that support selecting a specific endpoint for an operation or message from a set of allowed candidates. This is particularly useful in server farms and mobile environments.

**WS-MetadataExchange** A set of Web service mechanisms to exchange policies, WSDL and potentially other metadata between two or more parties.

**The WS-Security Roadmap protocols** In April 2002, IBM and Microsoft published a roadmap for Web service security (WS-Security Roadmap) that supports, integrates and unifies several popular security models, mechanisms, and technologies.

**WS-TransmissionControl** A set of constructs for controlling the exchange of messages between services to improve reliability by preventing message loss due to service unavailability, overloading queues and other causes.

### 4.3 Reliable Messaging Model

WS-ReliableMessaging provides an interoperable protocol that a Reliable Messaging (RM) Source and Reliable Messaging (RM) Destination use to provide Application Source and Destination a guarantee that a message that is sent will be delivered. The guarantee is specified as a delivery assurance. The protocol supports the endpoints in providing these delivery assurances. It is the responsibility of the RM Source and RM Destination to fulfill the delivery assurances, or raise an error.

There are four basic delivery assurances (Fig. 3) that endpoints can provide:

**AtMostOnce** Messages will be delivered at most once without duplication or an error will be raised on at least one endpoint. It is possible that some messages in a sequence may not be delivered<sup>8</sup>.

**AtLeastOnce** Every message sent will be delivered or an error will be raised on at least one endpoint. Some messages may be delivered more than once

**ExactlyOnce** Every message sent will be delivered without duplication or an error will be raised on at least one endpoint. This delivery assurance is the logical "and" of the two prior delivery assurances.

**InOrder** Messages will be delivered in the order that they were sent. This delivery assurance may be combined with any of the above delivery assurances. It requires that the sequence observed by the ultimate receiver be non-decreasing. It says nothing about duplications or omissions.<sup>9</sup>

### 4.4 Protocol Preconditions

The correct operation of the protocol requires that a number of preconditions must be established prior to the processing of the initial sequenced message:

The RM Source must have an endpoint reference that uniquely identifies the RM Destination endpoint; correlations across messages addressed to the unique endpoint must be meaningful.

---

<sup>8</sup>As informed on Oasis WS Reliable Messaging Working Issues List[29], issue 020, there are at least three possible semantics associated with *At-Most-Once*: (1) It means that the sender will never retransmit a message, regardless of whether it is acknowledged by the destination. (2) The sender may retransmit messages, but is not required to do so, however the destination will not deliver duplicates. (3) The sender must retransmit messages, however the destination may drop messages in times of resource saturation, but will never deliver a duplicate.

<sup>9</sup>As informed on Oasis WS Reliable Messaging Working Issues List[29], issue 027, the *InOrder* delivery assurance can only be enforced for messages within one sequence. If a new sequence has to be created, for example due to a MessageNumber rollover, the ordering of the messages can not be enforced unless there is a way to link the sequences together. If this is the intention it should be clarified in the specification.

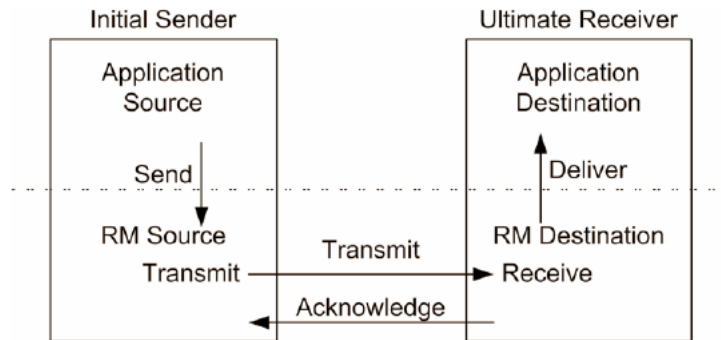


Figure 3: WS-Reliable messaging Model

The RM Source must have knowledge of the destination’s policies, if any, and the RM Source must be capable of formulating messages that adhere to this policy.

If a secure exchange of messages is required, then the RM Source and RM Destination must have a security context.

#### 4.5 Protocol Elements

The protocol elements are:

**Sequences** The RM protocol uses a `<Sequence>` header block to track and manage the reliable delivery of messages. Messages for which the delivery assurance applies MUST contain a `<Sequence>` header block.

**Sequence Acknowledgments** The RM Destination informs the RM Source of successful message receipt using a `<SequenceAcknowledgement>` header block.

**Request Acknowledgement** The purpose of the `<AckRequested>` header block is to signal to the RM Destination that the RM Source is requesting that a `<SequenceAcknowledgement>` be returned.

**Sequence Creation** The RM Source MUST request creation of an outbound Sequence by sending a `<CreateSequence>` element in the body of a message to the RM Destination.

**Sequence Termination** After an RM Source receives the `<SequenceAcknowledgement>` acknowledging the complete range of messages in a Sequence, it sends a `<TerminateSequence>` element, in the body of a message to the RM Destination to indicate that the Sequence is complete, and that it will not be sending any further messages related to the Sequence.

An example scenario involving these elements is shown in figure 4

#### 4.6 Faults

WS-RM defines the following fault codes, which can be detailed within the `<SequenceFault>` element.

**Sequence Terminated** This fault is sent by either the RM Source or the RM Destination to indicate that the endpoint that generates the fault has either encountered an unrecoverable condition, or has detected a violation of the protocol and as a consequence, has chosen to terminate the sequence.

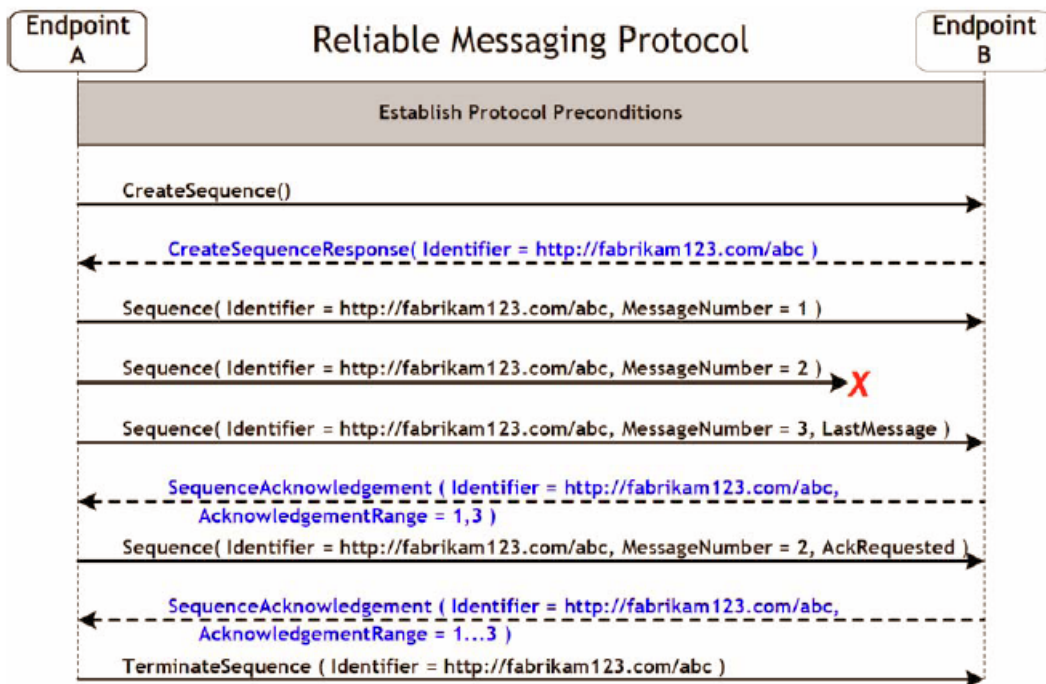


Figure 4: WS-RM normal scenario Example

**Unknown Sequence** This fault is sent by either the RM Source or the RM Destination in response to a message containing an unknown sequence identifier.

**Invalid Acknowledgement** This fault is sent by the RM Source in response to a `<SequenceAcknowledgement>` that violates the cumulative acknowledgement invariant.

**Message Number Rollover** This fault is sent to indicate that message numbers for a sequence have been exhausted.

**Last Message Number Exceeded** This fault is sent by an RM Destination to indicate that it has received a message that has a `<MessageNumber>` within a Sequence that exceeds the value of the `<MessageNumber>` element that accompanied a `<LastMessage>` element for the Sequence.

**Create Sequence Refused** This fault is sent in response to a create sequence request that cannot be satisfied.

Sequence creation uses a `CreateSequence`, `CreateSequenceResponse` request reply. Faults for this operation are treated as defined in WS-Addressing. `CreateSequenceRefused` is a possible fault reply for this operation. `UnknownSequence` is a fault generated by endpoints when messages carrying RM header blocks targeted at unrecognized sequences are detected, these faults are also treated as defined in WS-Addressing.

## 4.7 Security

Security is a complementary aspect of WS-RM, so this specification only summarizes recommendations to secure a reliable conversation because is out of its scope. All security implementations should follow the mechanisms defined in WS-Security, WS-Trust and WS-SecureConversation.



## 4.8 Final Remarks

Some discussion points can be highlighted about the protocol decisions taken by the standard promoters. As mentioned early in Section 4.1, WS-ReliableMessaging states that persistence considerations related to an endpoint's ability to satisfy the delivery assurances are the responsibility of the implementation and do not affect the wire protocol. This decision started some discussion [21] [22] [23] [24] about the need of some persistence constraint to guarantee that once the RM Destination has delivered the message to the Application Destination, it would not be lost by some unexpected failure. The question should be formulated as: How much reliability gives WS-RM to the endpoints if as such, it sends an ACK message prior the the Application processing?

Another interesting point about WS-ReliableMessaging is the way in which it treats the four delivery assurances. In early versions [28], these delivery assurance types were part of the WS-RM itself. A source specify the assurance type within the WS-RM header (by means of a policy assertion for WS-RM [26] attached using WS-PolicyAttachment [25]). These elements could be `<wsrm:AtMostOnce>`, `<wsrm:AtLeastOnce>`, `<wsrm:ExactlyOnce>` and `<wsrm:InOrder>`.

The last version, on the other hand, omits intentionally any explicit way to specify the delivery assurance type to the source point. As noted on the WS Reliable Messaging Working Issues List [29], issue 009:

In the specification, the delivery assurances are part of a private contract between the RM destination and the application destination. They are not published and they are not visible to the "outside" world - i.e. to the source.

The main concept is that WS-RM outlines the high level mechanisms and tools for implementing the four delivery assurance types mentioned early, and because these types were thought as a contract between the RM destination and the application destination, there is no need to specify the QoS to the RM source. The mechanisms (in conjunction with the protocol specified in the WS-ReliableMessaging standard) are accorded by source and sender based on WS-RM Policy Assertion [26] and defines:

1. an inactivity timeout parameter that either the RM Source or RM Destination MAY include. If during this duration, an endpoint has received no application or control messages, the endpoint MAY consider the RM Sequence to have been terminated due to inactivity.
2. a base retransmission interval parameter that the RM Source MAY include. If no acknowledgement has been received for a given message within the interval, the RM Source will retransmit the message. The retransmission interval MAY be modified at the Source's discretion during the lifetime of the Sequence.
3. a backoff parameter that the RM Source MAY include to indicate the retransmission interval will be adjusted using the commonly known exponential backoff algorithm.
4. an acknowledgement interval parameter that the RM Destination MAY include. Per WS-ReliableMessaging, acknowledgements are sent on return messages or sent stand-alone. If a return message is not available to send an acknowledgement, an RM Destination MAY wait for up to the acknowledgement interval before sending a stand-alone acknowledgement. If there are no unacknowledged messages, the RM Destination MAY choose not to send an acknowledgement. This parameter does not alter the formulation of messages or acknowledgements as transmitted; it does not alter the meaning of the `wsrm:AckRequested` directive. Its purpose is to communicate the timing of acknowledgements so that the RM Source may tune appropriately.

Another interesting element found in this specification is the inclusion of a NACK message to allow RM Destination to confirm the absence of a message in some sequence. Although its optional character, this mechanism helps to enhance the communication performance, avoiding to wait until RM Source timeout to detect the loss of a message.

WS-ReliableMessaging provides mechanisms to deal with sources behind a NAT or Firewall. In this case, there are two possibilities: send the confirmation (ACK) to a message within a response (commonly known as “piggy-backing”) or send the confirmation to another specified address based on WS-Addressing capabilities. While the `<AckRequested>` header can be viewed as a type of polling mechanism there is a bit of a difference, pointed out by Doug Davis[30], one of the contributors of WS-RM:

in WS-RM the `<AckRequested>` header doesn't require the server to send back an ACK in response to the request. The `<AckRequested>` is just a hint to the RM Destination that it needs to send an ACK back but it doesn't mandate when. Normally, it would return an ACK back as result of an `<AckRequested>` but its not as firm of a requirement to do it immediately.

This lack of constraints over the semantics of `<AckRequested>` contributed to the presentation of a new standard within the WS-\* family of specifications: Web Services Polling (WS-Polling) [31].

Finally, we can highlight a subtle point of confusion regarding policies and scopes, detailed in the open issue 021 in [29]:

in WS-RM Policy, and per WS-PolicyAttachment, RM Policy assertions may be attached to ports; this implies that an RM Policy applies to two Destinations: the destination of the inbound messages, and the destination of the outbound messages (meaning for example to both a WS endpoint and its clients). But differences in technical capabilities between Source and Destination, and differences in business and QoS requirements, suggest that a policy (delivery assurance) defined for messages going one direction, may not be appropriate for messages going the other way.

Some paragraphs before, we cited the concept of independence in the delivery assurances between two endpoints, so even if the scope of an RM Policy remains at port level there could be an additional scoping attribute stating inbound vs outbound.

## 4.9 Wrap-up Example

In this section we will show the SOAP message interchanges for the example illustrated in figure 4. To initiate a reliable message communication between two endpoints, an RM-Source need to create a sequence of messages (listing 5).

Then, WS-Destination confirms the sequence creation sending a create sequence response. Now suppose rm-source sends three messages in the sequence (only shown the first message on listing 6). The third is the last, so it contains the `<LastMessage>` element.

Message number 2 has not been received by the RM Destination due to some transmission error so it responds with an acknowledgement for messages 1 and 3 (listing 7)

The sending endpoint discovers that message number 2 was not received so it resends the message and requests an acknowledgement.

The RM Destination now responds with an acknowledgement for the complete sequence which can then be terminated (listings 8)

### Listing 5: Create Sequence Example

```

1 <?xml version="1.0" encoding="UTF-8"?> <S:Envelope
  xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3  xmlns:wsm="http://schemas.xmlsoap.org/ws/2005/02/rm"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
5  <S:Header>
    <wsa:MessageID>
7      http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546817
    </wsa:MessageID>
9    <wsa:To>http://fabrikam123.com/serviceB/123</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequence</wsa:Action>
11   <wsa:ReplyTo>
    <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
13   </wsa:ReplyTo>
  </S:Header>
15 <S:Body>
    <wsm:CreateSequence>
17      <wsm:AcksTo>
    <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
19      </wsm:AcksTo>
    </wsm:CreateSequence>
21 </S:Body>
</S:Envelope>

```

### Listing 6: Message 1 Example

```

<?xml version="1.0" encoding="UTF-8"?>
2 <S:Envelope
  xmlns:S="http://www.w3.org/2003/05/soap-envelope"
4  xmlns:wsm="http://schemas.xmlsoap.org/ws/2005/02/rm"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
6   <S:Header>
    <wsa:MessageID>
8      http://Business456.com/guid/71e0654e-5ce8-477b-bb9d-34f05cfc9e
    </wsa:MessageID>
10   <wsa:To>http://fabrikam123.com/serviceB/123</wsa:To>
    <wsa:From>
12     <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
    </wsa:From>
14   <wsa:Action>http://fabrikam123.com/serviceB/123/request</wsa:Action>
    <wsm:Sequence>
16     <wsm:Identifier>http://Business456.com/RM/ABC</wsm:Identifier>
    <wsm:MessageNumber>1</wsm:MessageNumber>
18   </wsm:Sequence>
  </S:Header>
20 <S:Body>
    <!-- Some Application Data -->
22 </S:Body>
</S:Envelope>

```

### Listing 7: First Acknowledgement

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3 xmlns:wsm="http://schemas.xmlsoap.org/ws/2005/02/rm"
4 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
5   <S:Header>
6     <wsa:MessageID>
7       http://fabrikam123.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546810
8     </wsa:MessageID>
9     <wsa:To>http://Business456.com/serviceA/789</wsa:To>
10    <wsa:From>
11      <wsa:Address>http://fabrikam123.com/serviceB/123</wsa:Address>
12    </wsa:From>
13    <wsa:Action>
14      http://schemas.xmlsoap.org/ws/2005/02/rm/SequenceAcknowledgement
15    </wsa:Action>
16    <wsm:SequenceAcknowledgement>
17      <wsm:Identifier>http://Business456.com/RM/ABC</wsm:Identifier>
18      <wsm:AcknowledgementRange Upper="1" Lower="1"/>
19      <wsm:AcknowledgementRange Upper="3" Lower="3"/>
20    </wsm:SequenceAcknowledgement>
21  </S:Header>
22  <S:Body/>
23 </S:Envelope>

```

### Listing 8: Sequence Acknowledgement

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <S:Envelope
3 xmlns:S="http://www.w3.org/2003/05/soap-envelope"
4 xmlns:wsm="http://schemas.xmlsoap.org/ws/2005/02/rm"
5 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
6   <S:Header>
7     <wsa:MessageID>
8       http://fabrikam123.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546811
9     </wsa:MessageID>
10    <wsa:To>http://Business456.com/serviceA/789</wsa:To>
11    <wsa:From>
12      <wsa:Address>http://fabrikam123.com/serviceB/123</wsa:Address>
13    </wsa:From>
14    <wsa:Action>
15      http://schemas.xmlsoap.org/ws/2005/02/rm/SequenceAcknowledgement
16    </wsa:Action>
17    <wsm:SequenceAcknowledgement>
18      <wsm:Identifier>http://Business456.com/RM/ABC</wsm:Identifier>
19      <wsm:AcknowledgementRange Upper="3" Lower="1"/>
20    </wsm:SequenceAcknowledgement>
21  </S:Header>
22  <S:Body/>
23 </S:Envelope>

```

## 5 WS-Reliability vs. WS-ReliableMessaging

In previous sections, WS-Reliability (Sect. 3) and WS-ReliableMessaging (Sect. 4) specifications have been presented in detail. We compare in this section these two competing initiatives. The analysis is organized as a list of items, that reflects different aspects of reliable messaging.

**Application Level Reliability Semantics** The first concern to compare is the semantics of the acknowledgments indications (acks).

- In WS-R, acks are sent only after the messages (payload) have been successfully delivered to the (receiving, consuming) application layer. In other words, the receiver (messaging middleware) assures that message is processed by the application before sending its ack to the sender. In terms of WS-R specification, acks are only issued after the successful completion of a *Deliver* operation.
- In WS-RM, acks are sent just after the RM-Destination passes the message to the Application Destination, without waiting for the completion of application processing. In this way, the application is partially in charge of assuring message reliability.

**Delivery Assurance Policies** Both specifications support different levels of reliable messaging service (contracts). However, the way these contracts are specified and represented varies.

- In WS-R, contracts are called *Agreements* (list of *agreement items*). The specification does not include a representation for agreements. Agreements are only defined abstractly. Once *somehow* an agreement is communicated to the sending endpoint (of the reliable middleware), it is represented in terms of the WS-R (wire) protocol and mapped to message headers. The receiving endpoint is in charge of interpreting message headers and supporting the semantics defined in the specification.
- In WS-RM, Delivery Assurance semantics are defined but not represented within WS-RM schemas or policies because a Delivery Assurance is considered an internal contract between Application-Destination and RM-Destination. Instead, WS-RM defined (by means of WS-PolicyAssertion) a set of primitive parameters that must be negotiated and adhered by both RM-Source and RM-Destination. These parameters are fundamental to implement the internal Delivery Assurance types.

**Message Reply Patterns** This aspect is related to the way acknowledgment (or fault) indications are sent from the receiving to the sending entities of the messaging middleware.

- WS-R offers three *message reply patterns* (see sub-section 3.5): Response, Callback and the Polling.
- WS-RM does not explicitly define message reply patterns because of its own independent-implementation philosophy. Despite of this omission, a close look at its policies and message structures allow us to take a crucial conclusion: polling is not guaranteed, because the `AckRequested` semantic does not prescribes to the RM-Destination to answer immediately (i.e. in the same underlying HTTP connection) which is a problem for a Source behind a Firewall or NAT pretending to communicate asynchronously. Except for this problem, which is being solved by WS-Polling and WS-Addressing standards, asynchronous communication is possible adjusting correctly ack timeouts and using the `AckRequested` message.

## Asynchronous Replies (ACKs and fault indications)

- WS-R supports asynchronous indications to be sent through two methods (reply patterns): callback and polling. As explained before, the callback variant implies that the destination endpoint of the reliable middleware has the ability to establish a communication with the sending endpoint. When connections cannot be opened (e.g. presence of firewalls or NAT) the *polling* variant can be used to tackle the problem. Basically, it is the sender who polls the receiver for ack (and fault) indications.
- In WS-RM, acks can go back (synchronously) on the HTTP response flow (piggybacking) or they can be transmitted (asynchronously) to an endpoint as a callback because WS-RM is based in WS-Addressing.

**Addressing** this aspect is about the mechanisms used to communicate addressing information.

- In WS-R, the <ReplyTo> tag is used to define the address of the service where acks must be sent for processing (target). The address is usually represented as a plain URL. It means that a direct connection will be required in order to send acks (as discussed before, in some situations this approach is not feasible).
- WS-RM is based on WS-Addressing, which differentiates the concepts (and addresses) of *destination*, *source endpoint* and *reply endpoint*. This scheme allows a response (ack) to be processed by an endpoint different than the source.

An alternative approach to the specification of a target endpoint (in the <ReplyTo> tag) would be the consideration of multi-hops and reverse paths. It implies a multi-hop addressing model for representing the path of messages involved in an end-to-end communication.

**Negative ACKs Support** It allows the receiver to signal negative acknowledgments (NACKs) for data not received. This mechanism is beneficial in terms of performance (avoiding unnecessary waiting time).

- WS-R do not support negative acks.
- WS-RM do support this mechanism (optionally), but this difference in functionality represents an additional complication to combine both specifications in the future.

## Message Identification

- WS-R defines group of messages. A message is identified by a group identifier and a message number. However if there is a single message on a group it is not mandatory to have a message number.
- WS-RM defines sequences of messages. A message is identified by a sequence identifier and a message number (compound key).

## Dependency of Technologies

- The obvious dependence of the WS-R specification on SOAP. Reliability-related parameters in WS-R is included into the SOAP header.
- WS-RM gives a better abstraction of this issue, supporting bindings to different protocols. Anyway, only the SOAP binding is available, yet.

## Security issues

- The WS-R specification can be combined with the OASIS Web Services Security: SOAP Message Security 1.0 specification. Despite of this, the WS-R specification does not attempt to profile their combination.
- WS-RM integrates with and recommends the use of WS-Security specification, so it's out of its scope how to address these security issues.

## Other aspects to consider:

- Message Order Support
  - Both the WS-R and the WS-RM specifications offer the Message order Quality of Service requirements, assuring messages to be received in order into the same group or sequence respectively. WS-RM enforces this ordering to messages within one sequence. This is a problem if a new related sequence has to be created as an extension of the former.
- Messages delivery agrupation/sequence
  - WS-R *Groups* refers to WS-RM *Sequences*
  - WS-R define the overall group life cycle.
  - WS-RM define sequence creation and sequence termination messages to generate sequences.
- WS-RM Supports Explicit Sequence Creation: The destination of a reliable exchange may allocate the identifier used for the exchange. This added degree of control allows the destination more control over resources. Without this optimization, the destination must maintain state for all delivered messages for the maximum possible transmission delay. For transports such as SMTP, this can be quite long.

## 6 Implementations

This section presents a list of existing WS-RM and WS-R implementations, including a brief report of each of them. We also present information about other initiatives on reliability solutions for WS's.

### 6.1 WS-ReliableMessaging

#### 6.1.1 Sandesha

Sandesha gives WS-ReliableMessaging implementations for the Apache Web Services project. Currently Sandesha has two implementations for the two widely used Web Service frameworks of Apache.

- **Sandesha1 [33] (for Apache Axis 1.x):** This implementation provides a complete support for WS-ReliableMessaging with the support to WS-Policy and WS-Addressing.

**Project Status:** A single distribution is available.

- **Sandesha2 [34] (for Apache Axis2):** using Sandesha2 you can add reliable messaging capability to the web services hosted using Axis2. Sandesha2 can also be used with Axis2 client to interact with already hosted web services in a reliable manner.

**Project Status:** No distribution found.

#### 6.1.2 Plumbwork Orange [35]

Plumbwork Orange is a project to augment the Web Services Enhancements for Microsoft .NET (WSE) by implementing various WS-\* specifications and support libraries. There exists a WS-ReliableMessaging implementation for Plumbwork Orange, so included also in WSE.

**Project Status:** Alpha 3 distribution can be downloaded.

#### 6.1.3 WCF/Indigo [36]

Windows Communication Foundation (WCF) (formerly code-named “Indigo”) is a set of *.NET* technologies for building and running connected systems. It is a new breed of communications infrastructure built around the Web services architecture. It provides secure, reliable, and transacted messaging along with interoperability. WCF unifies a broad array of distributed systems capabilities in a composable and extensible architecture, spanning transports, security systems, messaging patterns, encodings, network topologies, and hosting models.

WCF is Microsoft's implementation of all these WS-\* standards that Microsoft has developed in cooperation with industry partners, including WS-Reliable Messaging.

**Project Status:** A WCF Beta 1 distribution “not available” link exists.

#### 6.1.4 Diablo [37]

BEA WebLogic Server 9.0 code name “Diablo” supports the latest industry standards such as J2EE 1.4 and the latest Web Services standards, helping to allow developers to more quickly create portable and interoperable applications. Diablo is designed to provide enterprise-class messaging, which is a critical part of the infrastructure for building out a SOA.



Diablo implements the WS-ReliableMessaging standard which is designed to make different systems compatible for reliable exchange of business messages.

**Project Status:** WebLogic Server version 9.1 is now available.

### 6.1.5 Xfire [38]

XFire is a next-generation java SOAP framework. XFire makes service oriented development approachable through its easy to use API and support for standards. It is built on a low memory StAX based model.

Although it does not explicitly inform about implementing the WS-RM one, its site argues to implement various WS-R\* specifications.

**Project Status:** Various 1.0\* Releases Available.

## 6.2 WS-Reliability

### 6.2.1 RM4GS [39]

Three IT industry heavyweights, Fujitsu Limited, Hitachi, and NEC Corp., have releasing Reliable Messaging for Grid Services (RM4GS), an open-source implementation of the Web Services Reliability (WS-Reliability) standard 1.1.

It has been made publicly available with the purpose to *help speed the widespread adoption of software products that incorporate the WS-Reliability function.*

**Project Status:** Java source code available for free download.

## 6.3 Other Initiatives

### 6.3.1 NaradaBrokering [40]

The NaradaBrokering project at the Community Grids Lab is an open source project that researches fundamental issues pertaining to distributed middleware systems. These include, among others, issues of efficient routing, support for complex interactions, robustness, resilience, ordering, security and trust.

NaradaBrokering aims to provide a unified messaging environment that integrates grid services, web services, peer-to-peer interactions and traditional middleware operations. The NaradaBrokering project has been in the Open Source domain for the past 3 years.

They argue to not to have a preference between either WS-RM or WS-Reliability and that they plan to provide support for both.

**Project Status:** v1.0 of the WS-RM and WS-R combined release available.

### 6.3.2 ActiveMQ [42]

ActiveMQ is an open source JMS 1.1 provider and Message Fabric supporting clustering, peer networks, discovery, TCP, SSL, multicast, persistence, XA and integrates seamlessly into J2EE 1.4 containers, light weight containers and any Java application. ActiveMQ is released under the Apache 2.0 License

**Project Status:** ActiveMQ supports the WS-Notification specification, and within the official mailing list it was announced the future inclusion of the WS-RM support.

### **6.3.3 WSE [43]**

Web Services Enhancements for Microsoft .NET (WSE) is a supported add-on to Microsoft Visual Studio .NET and the Microsoft .NET Framework providing developers the latest advanced Web services capabilities to keep pace with the evolving Web services protocol specifications.

### **6.3.4 WS-I [44]**

WS-I Set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability.

## 7 Reliability in Notification Services (NS)

As it is obvious, WS-R and WS-RM specifications focus on reliable messaging for Web Services. But the exchange of messages is limited to a pair of actors: consumer and producer (a simple MOM case). What if we want to transfer messages reliably, among N consumers and M producers? These specifications do not assume any underlying mechanism, nor middleware behavior. So what about then if we want to transmit messages following a pub/sub paradigm reliably: then, what about reliability in NS's?

As an alternative, we could use any WS-R and WS-RM implementation to connect clients to NS local brokers, and still to connect the whole broker network. But this will not assure reliability in the overall (probably multi-hop) NS. Let's consider a scenario in which a node A sends a message reliably to another node B, and B tries to do the same to node C, but this last attempt fails. With this, no one assure to A that message reached all destinations. In other words, with WS-R and WS-RM alone, there is a lack of reliability in the overall middleware behavior.

In the following, we shortly introduce two NS specifications, focusing on the way they achieve reliability: *WS-Notification* and *JMS*.

### 7.1 WS-Notification (WS-N)

The Web Services Notification (WS-Notification) specification has been elaborated in terms of a Web Services Notification Framework which supports Publish-Subscribe Notification for Web services. This framework has been developed by Akamai Technologies, Computer Associates International, Fujitsu Laboratories of Europe, Globus, Hewlett-Packard, IBM, SAP AG, Sonic Software, and TIBCO Software.

These specifications implement the Notification pattern in which a service provider, or other entity, initiates messages based on a subscription or registration of interest from a service requestor. The Web Services Notification Framework defines how the publish/subscribe (pub/sub) pattern commonly used in Message-Oriented middleware products can be realized using Web services. This includes brokered as well as direct pub/sub which allows the publisher/subscribers to be decoupled and provides greater scalability. In the notification pattern a Web service (or other entity) disseminates information to a set of other Web services without requiring prior knowledge of them.

The WS-Notification family of related white papers and specifications [16, 17, 18, 19] that define a standard Web services approach to notification using a topic-based publish/subscribe pattern define:

1. Standard message exchanges to be implemented by service providers that wish to participate in Notifications
2. Standard message exchanges for a notification broker service provider allowing publication of messages from entities that are not themselves service providers;
3. Operational requirements expected of service providers and requestors that participate in notifications;
4. An XML model that describes *topics*. The Web Services Notification Framework currently includes three normative specifications.

Details about reliability have not been found in this specification. The only words about reliable messaging, explain that WS-Notification must be composable with other Web services specifications like the WS-RM one. The specification also describes that subscriptions messages can introduce an

attached policy that could be used to govern reliability requirements. Then, reliability issues are set to other specifications (actually the WS-RM one) and there is no middleware-aware reliability defined in WS-N.

## 7.2 Java Message Service (JMS)

JMS offers a Message Oriented Middleware (MOM) API specification for Java. This specification offers two messaging models: Point-to-point which refers on working with queues of messages, and the Publish/Subscribe (Pub/Sub) model [32] based on topics as destinations. Due to this later model, it is called to be a topic-based Notification Service.

Prior to detailing about reliability in JMS, we shortly present a WS related example. In order to offer reliability, WS community has thought of various alternatives until reliable messaging specifications were widely available. One interesting approach was to use implementations of SOAP over more reliable communication infrastructures. And one of these infrastructures was JMS. So, how much reliable is JMS?

A queue in JMS is defined to be always available to hold messages that it has received, whether or not the client that consumes its messages is active. For this reason in the Point-to-Point model, a client does not have to take any special precautions to insure that it does not miss messages. To insure reliability in the the Pub/Sub model is a little more complex, and we refer to it in the rest of this section.

JMS documentation [14, 15] refers on various mechanisms to achieve reliability. After a JMS specification analysis (and as these specifications do), we can say that JMS can be reliable. But we have found various drawbacks to be taken into account. First, the level of reliability depends on the way these mechanisms are used and how are they combined. Second, JMS specification lets some open reliability issues to be solved by implementors. In the following, JMS reliability mechanisms are presented, and conclusions about them are given.

- **Controlling message acknowledgment:** messages can be set to be auto-acknowledged (AUTO\_ACKNOWLEDGE) on being received, acknowledged by the client (CLIENT\_ACKNOWLEDGE) -possibly after message processing- or to be lazily acknowledgement (DUPS\_OK\_ACKNOWLEDGE) where duplication of messages might exist.
- **Specifying message persistence:** You can specify that messages are persistent, meaning that they must not be lost in case of a provider failure.
- **Setting message priority levels:** You can set various priority levels for messages, which can affect the order in which the messages are delivered.
- **Allowing messages to expire:** You can specify an expiration time for messages, so that they will not be delivered if they are obsolete.
- **Creating durable subscriptions:** You can create durable topic subscriptions, which receive messages published while the subscriber is not active.
- Although not relevant in this analysis it could be taken into account as reliability features to create temporary destinations and the use of local transactions.

Table 1 [14] shows some conclusions from all this. As we can see, in theory we can obtain at-most-once and once-and-only-once delivery assurances.

How Published	Nondurable Subscriber	Durable Subscriber
NON_PERSISTENT	at-most-once (missed if inactive)	at-most-once
PERSISTENT	once-and-only-once (missed if inactive)	once-and-only-once

Table 1: JMS Reliability

**But due to our analysis we reached to conclude that Table 1 is true if:**

- We take into account that we have no lost messages (messages are acked after being processed by the application):
  - Using one of the two scenarios to ensure that a message will not be acknowledged until processing of it is complete:
    - \* Using an asynchronous receiver -a message listener- in an `AUTO_ACKNOWLEDGE`. This configuration acks a message after the receiving method has finished.
    - \* Using a synchronous receiver in a `CLIENT_ACKNOWLEDGE` session (assuming programming error risks).
  - Assuring the persistence in *each* message producer or message.
- No duplicates are generated:
  - Not using `DUPS_OK_ACKNOWLEDGE` acks. If you do so, messages could be replicated due to provider fails.
  - If the specific JMS implementation (provider) takes into account the ambiguity produced when a failure is produced before acks arrive (specification leaves this issue open to the implementation).
- Taking care that desired reliable messages have not been set to expire in time.

We conclude then that we can obtain various levels of reliability from JMS. But it much depends on the correct use of the offered strategies. This is aimed to provide reliability, a JMS user must set all these strategies programmatically (and not descriptively), being prone to errors. The other point to guarantee reliability is to investigate that the specific JMS solution to be used adequately solves the explained ambiguity. JMS does not offer an organized set of QoS requirements to descriptively offer reliability.

### 7.3 NS Reliability Conclusions

As previously mentioned, WS-RM and WS-R specifications are not enough in detail to cover reliability in NS's. This is due to these specifications cover just peer-to-peer protocols. Although the sum of WS-N plus WS-RM sounds nice, there is lack of information to correctly join the NS and reliability worlds. The JMS specification actually offers a NS reliability approach. But as mentioned, there are various drawbacks and open issues. Still more, both NS specifications cover only topic-based addressing. A more general “Reliable NS” specification would be desirable to be applicable in more expressive mechanisms.

## 8 Conclusions

WS-Reliability and WS-ReliableMessaging are two initiatives that pursue the same goal: to support point-to-point reliable messaging for web services. At first glance both specifications tackle the same problem following similar approaches<sup>10</sup>. Both support the same set of delivery assurances (at-most-once, exactly-once, at-least-once, in-order). Both protocols are mapped onto SOAP headers. Both specifications rely on the same ideas of acknowledgment signaling, message grouping and identification. Regarding this last item, both specifications use own schemas for message identification, instead of relying on already existing ones (for instance, rather than using the `<MessageID>` element from WS-Addressing, WS-RM defines its own `<MessageNumber>` element for message identification). Both specifications consider the use of policies for specifying quality of services. Both protocols are point-to-point, in the sense that they must be implemented only by sending and receiving modules (intermediaries are not required to support these protocols).

After a detailed analysis, differences among these specifications came to light. Most of them are related to origin, supporters, evolution, scope and levels of abstraction of the specs. On the one hand WS-R was developed as an open specification (and subject to standardization at OASIS) and is supported by Sun, Sonic, among others. On the other hand, WS-RM was born as a proprietary protocol, developed principally by IBM and Microsoft. It has been made public recently (and submitted to OASIS for standardization). There is a key difference related to the semantics and scope of *reliable messaging*. WS-R considers that a message has been successfully delivered after it has been processed by the receiving application. On the other side WS-RM sends acknowledgements indications once messages are successfully delivered to the receiving endpoint of the messaging middleware. The different moments in which message acknowledgements are signaled (pre and post application message processing in WS-RM and WS-R respectively) is an important issue to be taken into account by applications and still by specification implementors.

WS-R is a concrete specification intended to be used in the context of SOAP over HTTP. For instance, WS-R explicitly considers two interaction patterns (One-way and Request-response) and explain them in the context of the specification. Additionally, WS-R defines 3 reply patterns (for signaling ACKs or faults) which are illustrated considering the two interaction patterns mentioned before.

On the other side WS-RM proposes to be independent of the underlying transport protocol and is designed to work over different transmission protocols (not just SOAP).

### 8.1 Looking for *the* Standard

Up to this point we have discussed differences and similarities of two efforts intended to address the same problem. However, the concurrent existence of these two solutions generates a new problem. Web services technology attacks the big problem of systems heterogeneity and interoperability by offering a set of open and agreed-upon specifications. However, the absence of a *unique* standard for reliability, put the users in the situation of choosing among two incompatible approaches. This constitutes a weak link in the WS-chain.

WS-R was born as a community effort. WS-RM, in turn, was disclosed recently and appeared as a strong rival of the community's effort. Indeed, there are voices coming from the WS-\* community criticizing the attitude of WS-RM promoters, qualifying it as disruptive [45].

---

<sup>10</sup>Although, in many points, WS-R and WS-RM use different names to designate the same underlying idea.

## 8.2 What's next?

Without a doubt, we think that the intention of standardizing different aspects of Web Services (as reliability) is an important and necessary work to do. Indeed the convergence of both initiatives that have been analyzed in this work would be, unquestionably, a good news for the WS community.

On April 19, 2005, WS-RM promoters announced a proposed *OASIS technical committee* and *TC Charter* to continue WS-RM initiative. A *Call for Participation* in the new OASIS Web Services Reliable Exchange (WS-RX) Technical Committee was issued on May 03, 2005 [41]. Among WS-RX supporters are included all those companies actively engaged with WS-R specification. It looks like the conflict moves towards a happy end [5].

## References

- [1] Chappell, David. Enterprise Service Bus. O'Reilly. 2004.
- [2] Web Services Reliability (WS-Reliability) Version 1.0: Frequently Asked Questions (FAQ). January 9, 2003. Sun.
- [3] Reliable Message Delivery in a Web Services World: A Proposed Architecture and Roadmap. IBM and Microsoft. March 2003. <ftp://www6.software.ibm.com/software/developer/library/ws-rm-exec-summary.pdf>
- [4] Web Services Reliable Messaging Protocol (WS-ReliableMessaging) February 2005
- [5] Eric Newcomer's Weblog. WS-RM Goes to OASIS. April 2005. <http://www.iona.com/blogs/newcomer/archives/000163.html>
- [6] IBM, Microsoft. Secure, Reliable, Transacted Web Services. October 2003. <http://www.ibm.com/developerworks/webservices/>
- [7] Doug Davis, IBM. WS-RM and WS-R: Can SOAP be reliably delivered from confusion?. Feb/2005.
- [8] Prasad Yendluri (webMethods). Web Services Reliable Messaging. In WebProNews Online (August 08, 2003). <http://xml.coverpages.org/reliableMessaging.html#Yendluri>
- [9] Robin Cover (xml.coverpages.org). Reliable Messaging Technical Report. December 2005. <http://xml.coverpages.org/reliableMessaging.html>
- [10] SOA Whitepaper. Adobe, Inc., 2005. [http://www.adobe.com.tr/enterprise/pdfs/Services-Oriented\\_Architecture\\_from\\_Adobe.pdf](http://www.adobe.com.tr/enterprise/pdfs/Services-Oriented_Architecture_from_Adobe.pdf)
- [11] Patterns: Service-Oriented Architecture and Web Services. IBM Redbook, 2004.
- [12] G. Hohpe, B. Woolf. Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions. Addison Wesley. 2004.
- [13] WS-Reliability Spec. Nov/2004.
- [14] Java Message Service (Specification) Version 1.1 April 12, 2002.
- [15] Java Message Service API Tutorial. Kim Haase (2002).
- [16] Publish-Subscribe Notification for Web services Version 1.0 May 2004.
- [17] Web Services Base Notification (WS-Base Notification) Version 1.0 May 2004.
- [18] Web Services Brokered Notification (WS-BrokeredNotification) Version 1.0 May 2004.
- [19] Web Services Topics (WS-Topics) Version 1.0 May 2004.
- [20] Reliable Message Delivery in a Webservices World: A Proposed Architecture and Roadmap. <http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-rm-exec-summary.asp>
- [21] WS-WhatThe?. March 2006. <http://www.kleineikenscheidt.de/stefan/archives/2006/03/ws-whatthe.html>



- [22] The case of the missing durable messaging feature. February 2006. <http://friends.newtelligence.net/clemensv/PermaLink,guid,e75402c6-bdd6-438c-9bf2-31f64b8e0557.aspx>
- [23] James Strachan's Weblog. Limitations of WS-\* and the need for Queued Messaging. February 28, 2006. <http://radio.weblogs.com/0112098/2006/02/28.html#a554>
- [24] Netzoid Blog. Do we need WS-RM and WS-QueuedMessaging?. March 1st, 2006 <http://netzoid.com/blog/2006/03/01/durabilityws/>
- [25] D. Box, et al, Web Services Policy Attachment (WS-PolicyAttachment). September 2004.
- [26] Web Services Reliable Messaging Policy Assertion (WS-RM Policy). February 2005.
- [27] D. Box, et al, Web Services Policy Framework (WS-Policy). September 2004.
- [28] D. Box, et al, Web Services Reliable Messaging Protocol (WS-ReliableMessaging). March 13, 2003.
- [29] Oasis WS Reliable Messaging Working Issues List. August 2005. <http://www.oasis-open.org/committees/download.php/14329/ReliableMessagin>
- [30] Doug Davis - Raleigh / IBM [dug@us.ibm.com](mailto:dug@us.ibm.com)
- [31] Kyle Brown, Doug Davis, Christopher Ferris and Anthony Nadalin, Web Services Polling (WS-Polling). Initial public draft release. W3C Member Submission 26 October 2005. <http://www.w3.org/Submission/ws-polling/>
- [32] Eugster, Felber, Guerraoui, Kermarrec. The many faces of publish/subscribe. 2003. ACM Computing Surveys, 35(2):114-131.
- [33] Sandesha1 oficial web page. <http://ws.apache.org/sandesha/sandesha1.html>
- [34] Sandesha2 oficial web page. <http://ws.apache.org/sandesha/sandesha2/index.html>
- [35] Plumbwork Orange oficial web page. <http://sourceforge.net/projects/plumbworkorange/>
- [36] WCF/Indigo oficial web page. <http://msdn.microsoft.com/webservices/indigo/default.aspx>
- [37] Diablo oficial web page. <http://www.bea.com/diablo>
- [38] Xfire oficial web page. <http://xfire.codehaus.org/>
- [39] RM4GS oficial web page. <http://businessgrid.ipa.go.jp/rm4gs/index-en.html>
- [40] NaradaBrokering oficial web page. <http://www.naradabrokering.org/>
- [41] WS-RX oficial web page. [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=ws-rx](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-rx)
- [42] ActiveMQ oficial web page. <http://www.activemq.org/>
- [43] WSE oficial web page. <http://msdn.microsoft.com/webservices/webservices/building/-wse/default.aspx>
- [44] WS-I oficial web page. <http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>
- [45] Clint Boulton. Sun Lashes Out at Microsoft, Others Over Spec. <http://siliconvalley.internet.com/news/article.php/2109831.htm>. March 14, 2003.