Web Services Reliable Messaging TC WS-Reliability

Working Draft 0.90, 26 January 2004

Document identifier:

wd-web services reliable messaging tc-ws-reliability-0.90

Location:

http://www.oasis-open.org/committees/wsrm/documents/specs/1.1/ws-reliability1.1.pdf

Editor:

Kazunori Iwasa, Fujitsu Limited <kiwasa@jp.fujitsu.com>

Abstract:

Web Services Reliability (WS-Reliability) is a SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicates, and guaranteed message ordering. WS-Reliability is defined as SOAP header extensions, and is independent of the underlying protocol. This specification contains a binding to HTTP.

Status:

This document is updated aperiodically on no particular schedule.

Committee members should send comments on this specification to the wsrm@lists.oasis-open.org list. Others should subscribe to and send comments to the wsrm-comment@lists.oasis-open.org list. To subscribe, send an email message to wsrm-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Web Services Reliable Messaging TC web page (http://www.oasis-open.org/committees/wsrm/).

The errata page for this specification is at http://www.oasis-open.org/committees/wsrm/documents/errata/1.1/index.html.

Table of Contents

1 Introduction

1.1 Purpose of WS-Reliability

The purpose of WS-Reliability is to address reliable messaging requirements, which become critical, for example, when using Web Services in B2B applications. SOAP [SOAP1.1] or [SOAP1.2] over HTTP [RFC2616] is not sufficient when an application-level messaging protocol must also address reliability and security. While security is getting traction in the development of Web Services standards, reliability is not. This specification is intended as an initial proposal for defining reliability in the context of current Web Services standards. The specification borrows from previous work in messaging and transport protocols, e.g., SOAP, and the ebXML Message Service [ebMS].

1.2 Scope and Definition of Reliable Messaging

The focus of this specification is on the SOAP layer and envelope. The OASIS WS-RM TC does not presume to cover all aspects of Reliable Messaging. Several fundamental questions on reliability need to be addressed in subsequent work, and are only partially addressed in this specification:

- Assuming that reliability objectives cannot always be guaranteed or attainable, should a reliability contract include advanced quality of service elements (which may translate into specifying quantitative thresholds, e.g. how large a message archive or time period a duplicate check should cover)?
- Beyond the specified qualities of message delivery (guaranteed delivery, duplicate elimination, and message ordering), should reliability also define the degree of synchronization between sender and receiver applications (i.e. the degree to which both sender and receiver parties will have same understanding of whether a request was properly received or not)?

Within the scope of this specification, the following features are investigated:

- Asynchronous messaging at the application level.
- Three reliability features: Guaranteed Delivery, Duplicate Elimination, and Guaranteed Message Ordering.

In the current specification, we will define reliable messaging as the mechanism supporting the following requirements at the application level:

- Guaranteed message delivery, or At-Least-Once delivery semantics.
- Guaranteed message duplicate elimination, or At-Most-Once delivery semantics.
- Guaranteed message delivery and duplicate elimination, or Exactly-Once delivery semantics.
- Guaranteed message ordering for delivery, within a context delimited using a group ID.

1.3 Limits of Scope

Some messaging features are not mentioned in this specification. They are considered out of scope, yet the design of this specification is preserving compatibility with some of them. They are:

- Application level synchronous messaging. Synchronous messaging applications that require immediate knowledge of the error status instead of waiting for the messaging layer to resend the message when an error is returned.
- Routing features. They can be used in conjunction with an implementation of this specification.
- Security Features. They can be used in conjunction with an implementation of this specification.

1.4 Goal of This Specification

The goal of this specification is to define:

- A mechanism to guarantee message delivery and its expression in SOAP messages.
- A mechanism to eliminate duplicate messages and its expression in SOAP messages.
- A mechanism to guarantee received message order (within a context) and its expression in SOAP messages.

1.5 Notational Conventions

This document occasionally uses terms that appear in capital letters. When the terms "MUST", “REQUIRED”, “SHALL”, "SHOULD", "RECOMMENDED", “MAY”, “OPTIONAL”, "MUST NOT", “NOT REQUIRED”, “SHALL NOT”, and "SHOULD NOT" appear capitalized, they are being used to indicate particular requirements of this specification. An interpretation of the meanings of these terms appears in [RFC2119].

1.6 Relation to Other Specifications

(1) W3C SOAP1.1/1.2:

SOAP1.1 [SOAP1.1] and SOAP1.2 [SOAP1.2] are the base protocol for this specification. This specification defines extensions to SOAP Header and Body elements.

(2) OASIS ebXML Message Service Specification 2.0:

The reliable message mechanism defined in the ebXML Message Service Specification 2.0 [ebMS] is implemented in a number of products and open source efforts, many of which have undergone interoperability testing. WS-Reliability borrows from this technology.

(3) OASIS WS-Security:

This specification can be used with WS-Security [WSS] when that effort is completed in OASIS.

(4) WS-I Basic Profile 1.0

This specification is compliant with WS-I Basic Profile 1.0 for use of other technologies including SOAP, WSDL, and XML schema.

1.7 Examples of Messages Compliant with WS-Reliability

(To be added later.)

1.8 Terminology

Reliable Messaging:

The set of mechanisms and procedures required to send messages reliably. This includes the processing of Acknowledgment messages, re-sending of messages, duplicate message elimination, and message ordering.

Reliable Messaging Processor (RMP):

A module capable of processing and enforcing Reliable Messaging as described in this specification. With regard to the transmission of a message from one RMP to another, the former will be act in the role of “sender” and the latter in the role of “receiver”.

Deliver:

An abstract operation the Receiving RMP may invoke per Reliable Message (e.g, a request to the application layer to take responsibility for the Reliable message).

Submit:

An abstract operation the Sending RMP supports, invoked per Reliable message (e.g., a request to the sending RMP to take responsibility for the reliable message. The time at which this operation is invoked must be clearly identifiable so that the RMP can always establish in which order two submissions are made.

Notify:

An abstract operation the Sending RMP may invoke per Reliable Message (e.g, a notification that the Sending RMP cannot insure that the Requested Reliability feature were realized)

Message Delivery:

Message delivery is the action of invoking the deliver operation for a Reliable Message. This action marks the end of the RMP processing for this message. The time at which this action occurs must be clearly identifiable so that the next message processor (application) can always establish in which order two deliveries are made.

Examples of message delivery are:

- pushing the message in a queue accessible by an application,
- calling back an application component,
- storing the message in a database where it is accessible by the next processor.

Reliable Message:

A message for which the sender requires some level of reliable delivery, typically requiring acknowledgment for notification of delivery.

PollRequest Message:

(TBD)

Acknowledgment Message:

An Acknowledgment Message contains an RM:Response element referring to at least one previous message (and containing no RM:Fault element). An Acknowledgment Message signals that the acknowledged message has been successfully delivered, meaning that it has satisfied all the reliability requirements placed on it for delivery.

Reliable Messaging Fault Message:

Such a message contains both an RM:Response element referring to at least one previous message, and an RM:Fault element. It signals to the sender of the referred message that there was a failure to receive or process the message.

Reliable Messaging Reply (or RM-Reply):

A message which is either an Acknowledge Message or Reliable Messaging Fault message.

Response RM-Reply Pattern:

We say that the Response RM-Reply pattern is used if the outbound Reliable Message is sent in a request of the underlying protocol and the RM-Reply is sent in the response message of the underlying protocol that corresponds to the request.

Callback RM-Reply Pattern:

We say that the Callback RM-Reply pattern is used if the RM-Reply of a previous message is contained in an underlying protocol request of a second request/response exchange (or a second one-way message).

Polling RM-Reply Pattern:

We say that the Polling RM-Reply pattern is used if a second underlying protocol request is issued to the receiver of a previous message, in order to obtain an RM-Reply. The RM-Reply is contained in the underlying protocol response to this request. This polling pattern is expected to be used in situations where it is inappropriate for the sender of reliable messages to receive underlying protocol requests.

1.9 The Reliability agreement

1.9.1 Definition

A Reliability agreement, or RM Agreement describes an agreed contract between a sender RMP and a receiver RMP, regarding:

- the nature, content and occurrence of exchanged messages,
- the timing, content and occurrence of the submit, deliver, notify operations on these RMPs.

In so far as the submit, notify and deliver operations are interpreted as implementing communication between an RMP and an application, the above contract can be seen as a contract between the application layer, the sender and receiver RMPs.

The way such a contract is established or communicated to each party is out of scope, although the assumption is that no prior communication of the contract to the receiving parties (RMP and its application) is required: all the needed synchronization is achieved through the message protocol, i.e. the Receiver RMP does not need other input than the message headers.

The highest-level items of the RM Agreement are the main RM features themselves:

- Guaranteed Delivery (or at-least-once delivery): When a Sender application submits a well-formed business payload to the RMP, the agreement requires that either: (1) the payload is delivered to the Receiver application, or (2) the Sender application is notified in case of failure.
- Duplicate Elimination (or at-most-once delivery): When an RMP delivers a business payload to a Receiver application, the agreement requires that no future business payload from a message with same identity as the message (MessageId) containing the first payload will ever be delivered to the Receiver application.
- Guaranteed Message Ordering delivery: When a Sender application submits an ordered sequence of business payloads to a RMP, the agreement requires that when delivering a business payload to the Receiver application, all previous payloads in the sequence have already been delivered.

1.9.2 RM Agreement Items

An RM Agreement is a list of Agreement Items. An RMP implementation MUST be capable of:

(1) taking knowledge (e.g. either via configuration, or via an API call, or via a message, or via the result of an algorithm) of a set of values that represent the RM Agreement Items described in this specification,

(2) processing them according to the semantics described in this specification.

Some of these items will map to some message header field, some will not. They are:

- GuaranteedDelivery (enabled/disabled): for setting guaranteed delivery.
- GuaranteedOrdering (enabled/disabled): for setting guaranteed message ordering.
- AtMostOnceDelivery (enabled/disabled): for setting message delivery without duplicates.
- 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.
- 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 how many 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 Acks or Faults.

1.9.3 Messaging Scope of Agreement Items

The messaging scope of these agreement items may be:

- (s1) All messages sent over a connection between a Sender RMP and a Receiver RMP (default).
- (s2) All messages sent within a Group.
- (s3) A single message, within a group of several messages (non-singleton group).

Some agreement items obviously relate to a particular scope, e.g. ExpiryTime is affecting each message separately, while GroupExpiryTime is an agreement about groups. Such scopes are "required" scopes that must be supported.

The smallest required scope for each RM agreement item is:

Message scope (s3):

- ExpiryTime
- RetryMaxTimes

- RetryTimeInterval
- ReplyPattern
- GuaranteedDelivery
- AtMostOnceDelivery

Group scope (s2):

- GuaranteedOrdering
- GroupExpiryTime
- GroupMaxIdleDuration

NOTE: although an RMP must support each agreement item at the scope level above, the RMP implementation may also provide a way to specify these values for a broader scope.

Example: an RMP implementation may decide to provide a way to specify the ExpiryTime value for all messages of a group.

1.9.4 Rules about Agreement Items

This section describes about rules for Agreement Items, when used in an RM Agreement.

- If GuaranteedOrdering is enabled for a messaging scope, then GuaranteedDelivery and AtMostOnceDelivery MUST also be enabled for that messaging scope.
- If GuaranteedDelivery is enabled for a messaging scope, then the items (RetryMaxTimes, RetryTimeInterval) MUST also be defined for that scope.
- If GroupExpiryTime is enabled for a messaging scope, then the item GroupMaxIdleTime MUST NOT be enabled, and vice versa.

2 Messaging Model

The following sections provide an overview of the WS-Reliability Messaging Model.

2.1 Overview of Messaging Model

The Reliable Messaging Model described in this document makes the following assumptions:

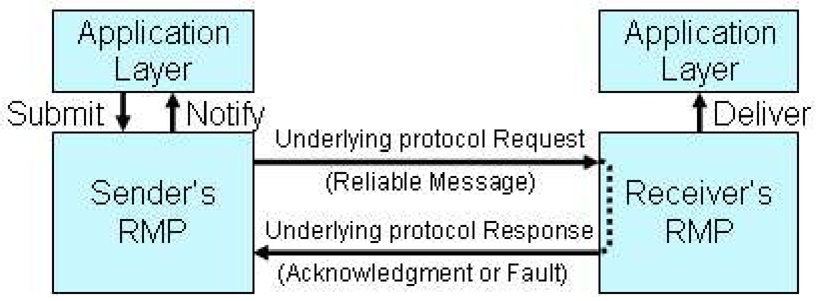
- Only two messaging nodes are considered, with respective roles of sender and receiver: (1) the sender RMP on which the submit message operation is invoked, and (2) the receiver RMP which invokes the deliver message operation. Intermediaries are transparent to this specification. Signal messages such as Acknowledgment message or Reliable Messaging Fault message are sent from the receiving RMP to the sender RMP.
- The underlying protocol is a request-response protocol. In other words, this specification assumes the underlying protocol distinguishes two kinds of messages: requests and responses. Under normal conditions, a response is always sent back for each request. This assumption is not essential to the reliable features described here: these could be reformulated without this assumption.

There are three ways to send back an Acknowledgment message or a Fault message as described as follows:

(1) Request/Response Messaging Model

With this model, the outbound Reliable Message is sent in the underlying protocol request and the Acknowledgment Message is contained in the underlying protocol response message corresponding to the original request. The figure 1 shows this model.

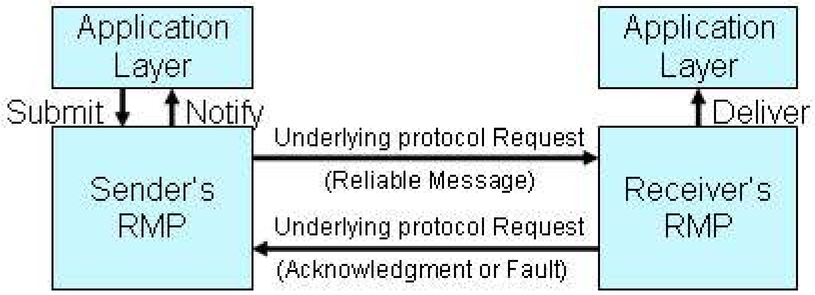
Figure 1 Request/Response Messaging Model



(2) Callback Messaging Model

With model, Acknowledgment Message is contained in an underlying protocol request of a second request/response exchange (or a second one-way message), operating in the opposite direction to the message containing the outbound Reliable Message. The figure 2 shows this model.

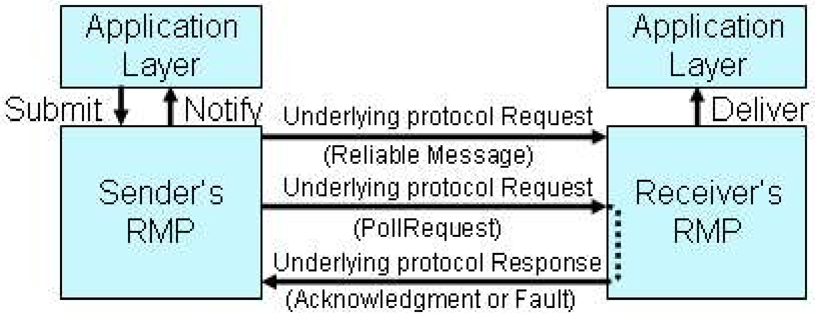
Figure 2 Callback Messaging Model



(3) Poll Messaging Model

With this messaging model, a second underlying protocol request is issued in the same direction as the one containing the outbound Reliable Message to act as a request for acknowledgment. The Acknowledgment Message is contained in the underlying protocol response to this request. This messaging model may be used in situations where it is inappropriate for the sender of reliable messages to receive underlying protocol requests. The figure 3 shows this model.

Figure 3 Poll Messaging Model



2.2 Groups of Messages and Message Identity

Every Reliable Message MUST contain a globally unique Message Identifier. A message always belongs to a group. A group of messages is sent from the sender RMP to the receiver RMP as a sequence of individual messages. The Message Identifier is a combination of a group ID and of an optional sequence number which is an integer, and which is unique within a group. More precisely, a message is identified as follows:

(1) In case there is only one message in the group (singleton): the group ID, which is a globally unique group identifier, may be used alone as Message Identifier. No sequence number is required, although allowed.

(2) In case the message belongs to a group of several messages: the message is identified by the group ID and a sequence number. A the group is submitted to the sender RMP as a sequence of messages, each sequence number value MUST be numbered with consecutive values starting with 0, in the submission order, and MUST be sent in the same order.

2.3 Guaranteed Delivery

A message submitted to the sending RMP with guaranteed delivery requested, will either be delivered by the receiving RMP, or the sending RMP will notify the submitter of failure. The guaranteed delivery mechanism will however do its best to get the message delivered, e.g.

resend a message in case of previous failure. In order for the mechanism described here to operate reliably, it is assumed that the underlying transport protocol prevents message corruption.

If the RMP sending a Reliable Message does not receive an Acknowledgment for a sent message that has not yet expired, it MUST resend the same message with same Message Identifier to the receiver RMP until the sender gets an Acknowledgment message from the receiver, or until the number of resend attempts specified by the RetryMaxTimes agreement item is exhausted, whichever occurs first. The time interval between two retries is specified by the RetryTimeInterval agreement item. If the sender RMP fails to send the message (i.e., no Acknowledgment is received), the sender RMP MUST notify a delivery error.

Guaranteed Delivery assumes also that the RMP functions are operational.

Example 1). A PC Server may use a HDD for it's persistent Storage, and those messages persisted in the HDD are reliably maintained even if the the system software crashes and the system is rebooted. However, if the HDD itself crashes, it is neither possible to deliver the message on the receiver side, nor to notify failure on the sender side.

Example 2) . A message persisted in a sending mobile phone may be lost when it's battery is detached. In this case, neither successful message transmission and delivery will be possible, nor failure notification.

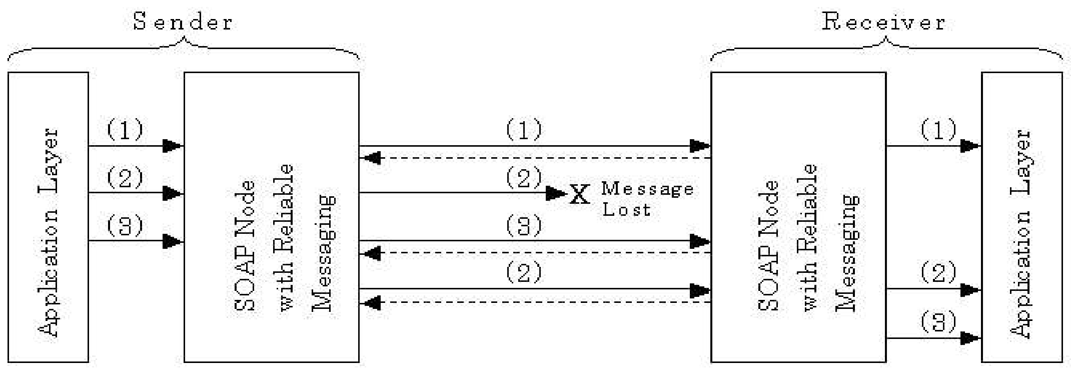
2.4 Duplicate Elimination

A number of conditions may result in reception of duplicate message(s), e.g., temporary downtime of the sender or receiver, a routing problem between the sender and receiver, etc. In order to provide At-Most-Once semantics, the receiver RMP MUST NOT deliver a message that is a duplicate of a previously delivered message. In case Duplicate Elimination is required, an RMP MUST persist the message ID (group ID and optionally the sequence number) of a delivered message at least until the message expires.

2.5 Guaranteed Message Ordering

Some applications will expect to receive a sequence of messages from the same sender in the same order these messages were sent. Although there are often means to enforce this at the application layer, this is not always possible or practical. In such cases, the messaging layer is required to guarantee the message order. Guaranteed Message Ordering provides this function. Figure 3 shows an example how Guaranteed Message Ordering works. When the sender application sent three messages (1), (2), and (3) with Guaranteed Message Ordering, Receiver's RMP MUST guarantee the message order when it delivers these messages . With the case of Figure 3, the receiver's RMP received message (1) and (3), the receiver's RMP delivers the message (1), but it persists message (3) until message (2) is received. When receiver's RMP received message (2), it delivers message (2) and (3) in order.

Figure 3 Guaranteed Message Ordering



2.6 Sequence Number

A sequence number mechanism is used to track and enforce the order of a sequence of messages having a common grouping identifier value. Such a mechanism has been widely used in the past. In the Figure 3 above, assume the sender application layer submits three messages in order of (1), (2), and (3). The sender RMP, with the message ordering function enabled, sends those messages in order of (1), (2), and (3), sequentially and asynchronously, with respective sequence numbers 1, 2, and 3. If the message (2) was not properly received for any reason, the sender will resend the (2) message after a timeout has occurred. The receiver RMP will finally receive these messages as a sequence: (1), (3), and (2). The receiver RMP, with the message ordering function enabled, will delivery message (1). But it MUST NOT deliver (3) before (2) has been delivered. Sequence numbering allows the receiver RMP to easily detect a missing message in a sequence, that is (2), as soon as receiving (3). This condition is recognized by the receiver when the sequence numbers of the messages it receives are not contiguous (e.g., 1, 3, 2). The receiver RMP will wait for a message with sequence number 2, and then deliver message (2) followed by message (3) in order. This behavior can be subject to variants and additional rules to deal with specific failure use cases, such as when a node cannot deliver the proper-sequence of messages due to a message being lost or expired.

2.7 Message Persistence

With Reliable Messaging, the sender is REQUIRED to persist the message until one of the following conditions are met:

- Receipt of an Acknowledgment message from receiver, indicating the message has been successfully delivered.
- All retry attempts have failed, and a delivery failure is reported to the application layer.
- The span of time indicated by the ExpiryTime element has expired.

The receiver MUST persist out of order messages to support Guaranteed Message Ordering. The receiver MUST persist the Message Identifier to support Duplicate Elimination. Both sender and receiver MUST behave as if there was no system failure or system down after recovery.

2.8 Group Termination and State Removal Criteria

NOTE: Being able to know when a group may be terminated, is essential for efficient management of the persistent store of an RMP. As groups may last a long time and their state requires persistence, it is important to know when their persistent image can be reclaimed. The termination rules described in this section may seem multiple and complex. This plurality results from the flexibility given to users in specifying various ways a group can be terminated, which in turn depends on application needs. However, in spite of this plurality, the termination logic is straightforward to implement and shares the same basic mechanisms across termination rules.

Termination of a group in the sender RMP and in the receiver RMP are two distinct events not synchronized by any special message, but instead occurring as the result of rules applying separately to the sender and to the receiver. As a consequence, the termination of a group may occur at quite different times on the sender and receiver RMPs. However, this lack of synchronization allowed by these termination rules is not consequential.

More precisely, there are two distinct operations that an RMP must perform when terminating a group, and these may also occur at different times, especially on the receiver side:

- **Group Closing:** When a group is closed in the Sender RMP, no new message is expected to be sent by the RMP for this group. When a group closes in the receiver RMP, no new message is expected to be received for this group anymore. After a group is closed, all subsequent messages sent with same group ID would be handled as belonging to a new group, unless they are duplicates of previous messages in the group, in which case they are treated as duplicates within this group. If a message is received after the closing of a group, with same group ID as the closed group, it may be considered by the Receiver as belonging to a new group (the Receiver is not required to verify that a new group ID value has not already been used in a previous group, as this test is impractical).
- **Group State Removal:** The state of a group includes group-specific attributes such as group status (e.g. “active”, “closed”), group ID, current sequence number, as well as all received Message Identifiers for the group (e.g. sequence number intervals). The state of a group also includes the persistent image of messages of this group being currently processed, although the removal of the persistence of messages follows its own rules. E.g. The resending mechanism for guaranteed delivery will take care of removal of messages on the sender side, once they are acknowledged. State removal occurs at the time or after the group is closed. When the state of a group is removed, all group attributes are removed, including the past message Ids on receiver side. Therefore not duplicate check may be done over past messages of this group.

In all termination cases (t1, t2, t3, t4, t5) described in this section, it is not necessary to remember the ExpiryTime of all messages of a group, as only the max(ExpiryTime) of messages received for the group is needed. These termination rules apply to both ordered and unordered groups. However, these rules do NOT apply to singleton groups, which contain a single message with no sequence number.

Assuming the last message of a group is marked with “end” status value, a group is defined as being “complete” on the receiver RMP when all the messages from ‘start’ to ‘end’ are received.

2.5.1 Group Termination

There are two parameters - groupExpiryTime and groupMaxIdleDuration to determine when a group can be terminated. The following assumptions pertain to these two group persistence control parameters:

a) the First message in a group (the one with status=start) MUST be used by the sender to indicate that time based group persistence control is in use for the group.

- If the first message in the sequence of a group has neither group persistence parameter present, the group will be terminated according to condition t4 or t5.
- If the first message has either of the two group persistence parameter present (either groupExpiryTime, or groupMaxIdleDuration) then that parameter will be used in the criteria used for terminating the group.
- A fault MUST be returned if both group persistence parameters are present in any request. An InvalidGroupParameter fault shall be sent in this case..
- If groupExpiryTime is in use, the sender must not send a message in that group with an ExpiryTime greater than the groupExpiryTime.

b) The group termination parameter which was sent on the first message in the group MUST be used on all subsequent messages in that group, and MUST be assigned a value.

c) The receiver MUST use the value from message with the highest sequence number received for the group.

d) In any subsequent message the parameter which was sent in the first message can be changed by sending a new value. A new value for groupMaxIdleDuration can either be increased or decreased. The protocol allows change (up or down) of groupExpiryTime, as long as it is never less than max(ExpiryTime) of messages received so far for the group.

An InvalidMessageParameters Fault MUST be returned if the value of groupExpiryTime is decreased to be less than the max(ExpiryTime) of messages received for the group.

2.5.2 Termination (t1):

The group had groupExpiryTime specified.

Receiver side:

Triggering event: groupExpiryTime is over.

The RMP MUST NOT accept any new message for this group and MUST close the group. It is RECOMMENDED that its state be removed as soon as possible after this. No duplicate check needs to be done against that group ever. If a "late duplicate" arrives, it would never be delivered to the next layer, as its ExpiryTime, which is always earlier than groupExpiryTime, would have expired.

Sender side:

Triggering event: groupExpiryTime is over.

The group MUST be closed, and its state removed from the RMP.

2.5.3 Termination (t2):

The group had groupMaxIdleDuration specified.

Receiver side:

Triggering event: groupMaxIdleDuration is over.

The group MUST be closed. But unlike (t1), some of its past messages may not have expired yet, and therefore their ID still be needed for duplicate checks. If we define max(ExpiryTime) as the max of all ExpiryTimes of messages received for a group, an RMP MUST persist the state of a group even after closing of the group, at least until max(ExpiryTime) is reached, in case Duplicate Elimination is required.

Sender side:

Triggering event: groupMaxIdleDuration is over.

The group MUST be closed, and its state removed from the RMP when the time elapsed since the last sent message (including retries) exceeds groupMaxIdleDuration.

2.5.4 Termination (t3):

The group had either groupExpiryTime or groupMaxIdleDuration specified.

Subcase t3.1: The group is complete on receiver side.

Receiver side:

Triggering event: The RMP receives a status="end" message.

The group MUST be terminated. However, its state is removed according to (t1) or (t2), depending which termination criterion was specified for the group.

Sender side:

Triggering event: The RMP sends a status="end" message.

All messages of the group have been sent. If Guaranteed Delivery was required, the group MUST be closed and state is removed once all sent messages have either been acknowledged, or their delivery failure notified. If no Guaranteed Delivery was required, the group MUST be closed and its state may be removed immediately.

Subcase t3.2: The group is not complete on receiver side.

Receiver Side:

Triggering event: the RMP receives a status="end" message.

In this case, the group is not yet closed. Indeed, an "end" status only tells that "no greater sequence number will ever be received after", but late messages may still arrive for this group. Then the Receiver RMP MUST apply termination rules of (t1) or (t2), depending which one of the two group termination parameters (I.e. groupExpiryTime or groupMaxIdleDuration) was specified.

Sender Side:

Triggering event: the RMP sends a status="end" message.

As all messages for the group have been sent, same rules apply as in t3.1.

2.5.5 Termination (t4):

The group had neither groupExpiryTime nor groupMaxIdleDuration specified.

Subcase t4.1: The group was complete on receiver side.

Receiver side:

Triggering event: the RMP receives a status="end" message.

The group MUST be closed. The time of removal of its state is determined by the max (ExpiryTime) received of the Group.

Sender side:

Triggering event: the RMP sends a status="end" message.

Same rule applies as in t3.1.

Subcase t4.2: The group was not complete on receiver side.

Receiver side:

Triggering event: The RMP receives a status=”end” message.

In this subcase, the RMP should keep the group processing active: this event, by itself, does not cause the closing of the group.

The group can close in two ways:

a) When the group becomes complete, or

b) When the max(ExpiryTime) received of the Group is reached.

In either case a) or b), the time of group state removal is based on the max(ExpiryTime) received of the group.

Sender side:

Triggering event: the RMP sends a status="end" message.

Same rule applies as in t3.1.

2.5.6 Termination (t5):

Receiving side:

Triggering event: In an ordered group, a message expires before delivery in out-of-order sequence.

The group MUST be closed.

State is removed:

- If the group does not have termination parameter, then it will be removed when the max(ExpiryTime) received of the group passes.
- If the group uses groupExpiryTime, then removal criteria as defined in t1 will apply.
- If the group uses groupMaxIdleDuration, then removal criteria as defined in t2 will apply.

Sender Side:

Triggering event: In an ordered group, a non-acknowledged message expires.

State is removed:

- If the group does not have termination parameter, then it will be removed when the max(ExpiryTime) sent for the group passes.

- If the group uses groupExpiryTime, then removal criteria as defined in t1 will apply.
- If the group uses groupMaxIdleDuration, then removal criteria as defined in t2 will apply.

2.9 WSDL operation type

The following table shows the correspondence of WSDL operation types to RM-Reply patterns.

Chart 1 WSDL operation types

WSDL operation type	***Response RM-Reply pattern***	***Callback RM-Reply pattern***	***Poll RM-Reply pattern***
Request/Response WSDL operation type*	Supported	Supported	Supported
One-way WSDL operation type	Disallowed **	Supported	Supported

* The WS-Reliability protocol does not support treating a WSDL response as a reliable message. It only supports delivery of the WSDL request as a Reliable message.

** WS-I BP 1.0 disallows sending soap envelope in HTTP response. However, the protocol does not required Receiving RMP to enforce this restriction. The receiver can do whatever the header asks for.

2.10 Attachments

When this spec is used with W3C note SOAP messages with Attachments specification, the following rules MUST be met:

1) The first MIME part MUST include whole SOAP envelope with WS-Reliability header elements.

2) The charset of the Content-Header of the first MIME part MUST be either UTF-8 or UTF-16.

3) Zero or more additional MIME parts MAY be included in a reliable message.

4) The receiver RMP MUST deliver all MIME parts in a Reliable Message to the receiving application.

3 Message Format

Figure 4 shows the structure of WS-Reliability elements embedded in the SOAP Envelope.

Figure 4 Structure of WS-Reliability elements

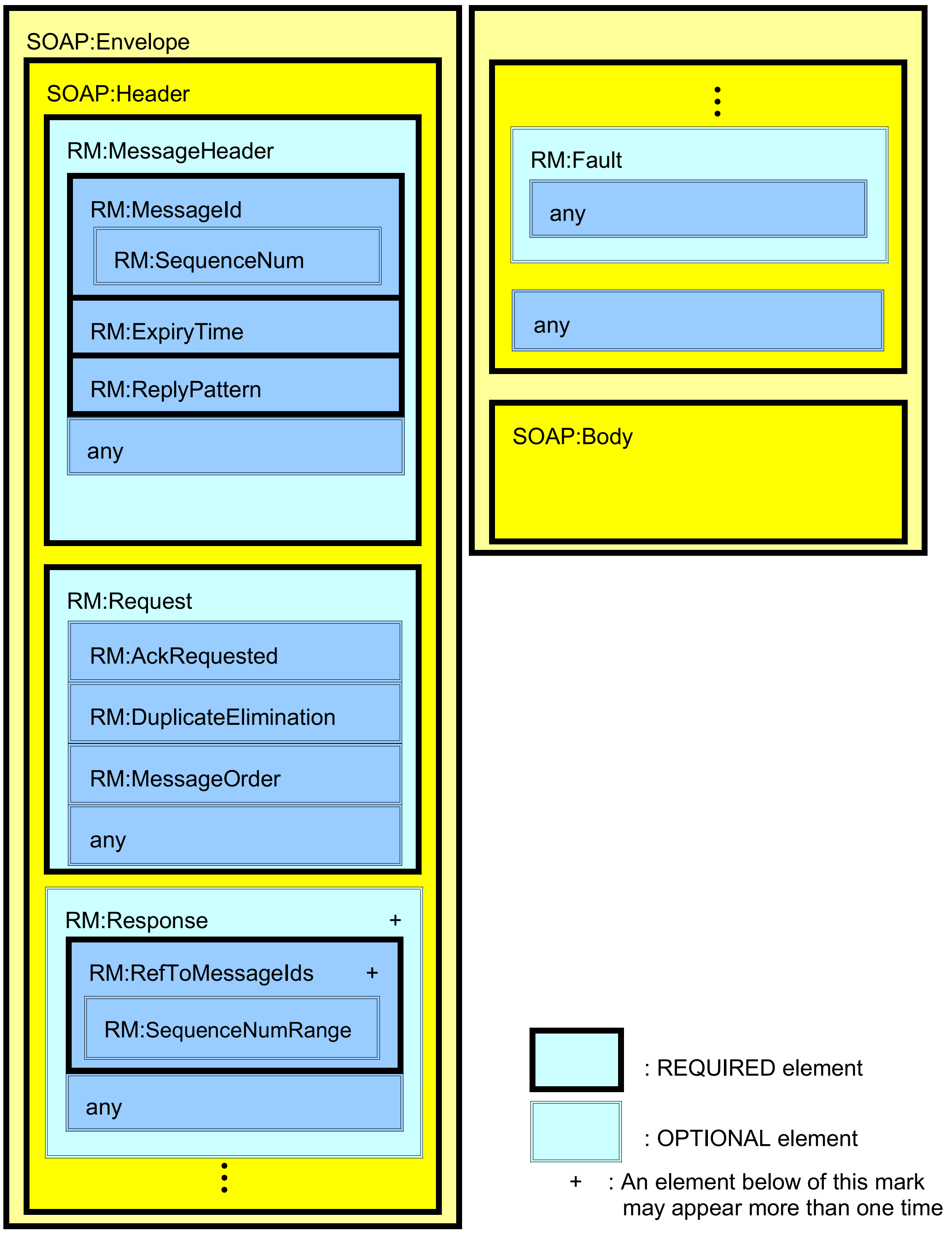
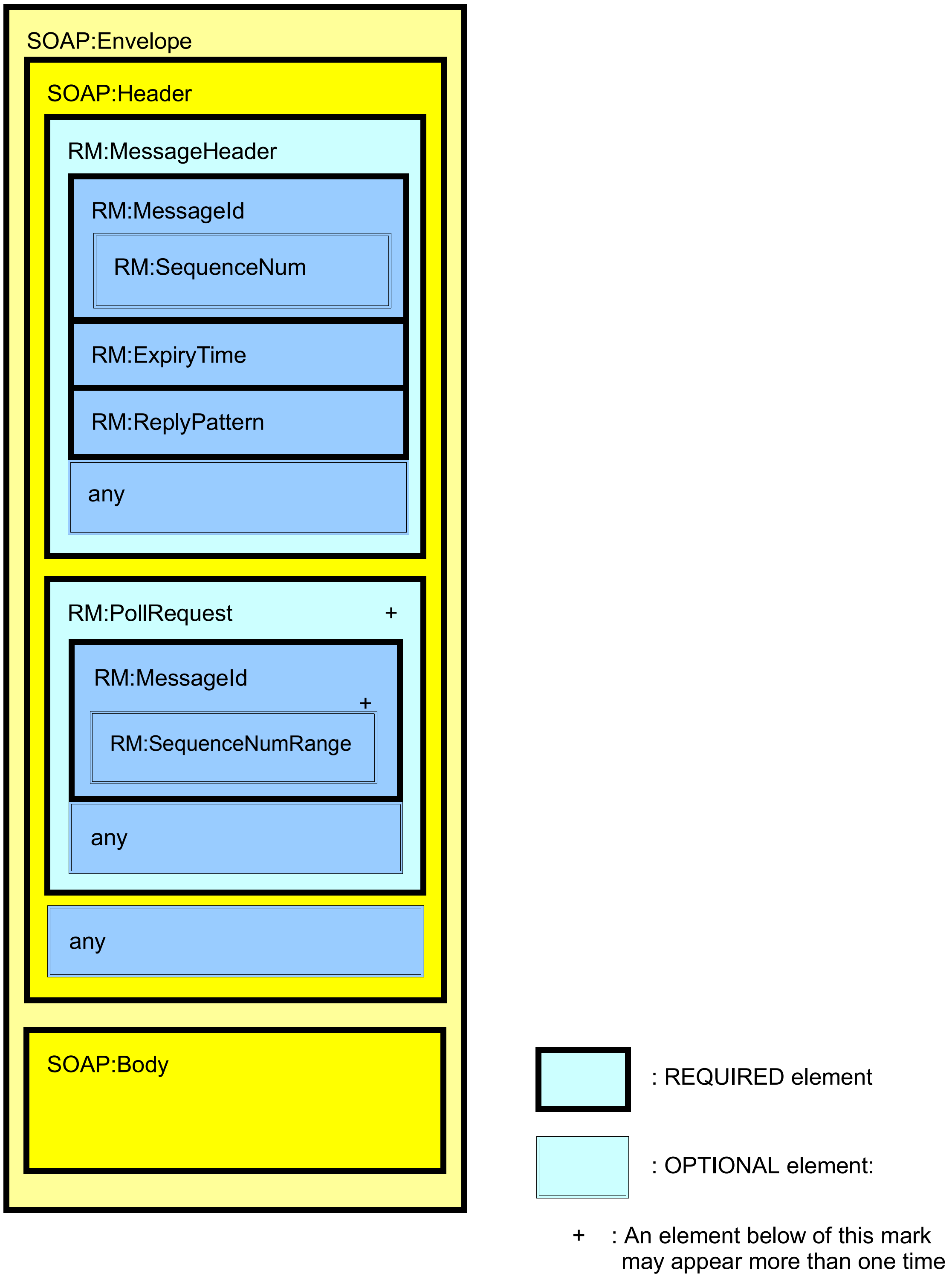


Figure 5 shows the structure of PollRequest message embedded in the SOAP Envelope.

Figure 5 Structure of PollRequest message elements



The namespaces [XML namespaces] for reliable messaging defined in this specification are:

http://www.oasis-open.org/committees/wsrm/schema/1.1/SOAP1.1 for SOAP1.1 and

http://www.oasis-open.org/committees/wsrm/schema/1.1/SOAP1.2 for SOAP1.2

If there are additional elements that are not described in this specification present in a message, the Reliable Messaging Processor MUST ignore those elements.

In a reliable message, the following four elements are direct children of SOAP Header:

- **MessageHeader** element
- **Request** element
- **PollRequest** element
- **Response** element
- **Fault** element

3.1 MessageHeader Element

The MessageHeader element MUST be present for Reliable Message, PollRequest message, Acknowledgment message, or Fault message. The MessageHeader element includes basic information to be used for a reliable message. This element includes the following attributes and child elements:

- SOAP **mustUnderstand** attribute with a value of "1"
- **MessageId** element
- **SequenceNum** element
- **ExpiryTime** element
- **ReplyPattern** element

Table1 MessageHeader Element

Cardinality	1
Value	None
Attributes	mustUnderstand
Child elements	MessageId SequenceNum ExpiryTime ReplyPattern
Fault	InvalidMessageHeader

Example 4 shows an example of a MessageHeader element.

Example 4 MessageHeader Element

(Example are included later)

3.1.1 MessageId Element

The MessageId element MUST be present for Reliable Message, Acknowledgment message, Fault message and PollRequest message. A request message MUST contain a groupId attribute and MAY contain either groupExpiryTime attribute or groupMaxIdleDuration attribute corresponding to the group termination parameters specified in section 2.5.1:

– a **groupId** attribute

Table2 MessageId Element

Cardinality	1
Value	None
Attributes	groupId (RFC2396 *See 3.1.1 for details)
Child elements	SequenceNum
Fault	InvalidMessageId InvalidGroupParameter

(1) groupId attribute

This attribute MUST be present in the MessageId element. This attribute is to identify a sequence of messages, where each sequence is of length 1 or more. The sending RMP MUST use a distinct globally unique groupId for any distinct group of messages. Any group of messages will have a common groupId value. The syntax of this identification is URI, as defined in [RFC2396]. It is RECOMMENDED to use the Message-ID schema, as defined in [RFC2392].

3.1.2 SequenceNum Element

The SequenceNum element is a REQUIRED element for Group of more than one message.

When a message includes a MessageOrder element, the SequenceNum element is used for guaranteeing the message order within the group of messages specified by the same groupId value. In other words, the sequence of numbered messages that the receiver node presents to the application MUST be in the same order as the sequence of numbered messages that the sender application has produced, within the group of messages having the same groupId value.

When the sender requests Guaranteed Message Ordering, the sender MUST use Guaranteed Message Delivery and Duplicate Elimination for that message as well. In particular, the sender MUST include both an AckRequested element and a DuplicateElimination element, as well as the MessageOrder element for Guaranteed Message Ordering.

This element includes the following attribute:

- a **groupExpiryTime** attribute
- a **groupMaxIdleDuration** attribute
- a **number** attribute attribute
- a **status** attribute

If the MessageOrder element appears in the message sent, the receiver of the message MUST make messages available to the application layer only after all messages with lower number with the same groupId have been made available to the application. Example 5 illustrates this:

Example 5 SequenceNum Element

(Example will be added later, when the schema is decided)

Table3 SequenceNum Element

Cardinality	0 or 1 *See 3.1.2 for details
Value	unsignedLong
Attributes	groupExpiryTime (dateTime) groupMaxIdleDuration (duration) number (unSignedLong) status (string)
Child elements	None
Fault	InvalidSequenceNum

(1) groupExpiryTime attribute

This is an OPTIONAL attribute. This attribute is used to specify the the date and time at which the sender wishes the sequence group to terminate. The groupExpiryTime MUST be expressed as UTC and MUST conform to a [XML Schema] dateTime.

(2) groupMaxIdleDuration attribute

This is an OPTIONAL attribute. This attribute is used to specify the maximum idle time. On the receiver side, if the time interval since the last message was received exceeds the groupMaxIdleDuration, then the sequence group may be terminated. On the sender side, the same condition applies to the time since the last message was sent. The groupMaxIdleDuration MUST conform to a [XML Schema] duration.

(3) number attribute

The value of this attribute MUST be unique within the same groupId, and the combination of groupId and SequenceNum MUST be globally unique to be used for Message Identifier.

When a sender node communicates with a receiver node across several groupId values, the sender MUST maintain an independent counter of the value of number attribute for each groupId. When sending a message containing a MessageOrder element with a new groupId, the sender MUST start with “0” for the number attribute in the groupId.

The value of number attribute MUST conform to [XMLSchema] unsignedLong. For the initial message with a specific groupId that is sent to the receiver, the number value MUST be “0”. After the initial message has been sent to the receiver, the sender MUST increment the value by one for each message sent. When the value of a number reaches the maximum value, the sender MUST generate a new groupId for any following messages. This begins a new sequence that could overlap with the old in rare circumstances. From the receiver's perspective, no link exists between the two sequences. To improve the chances that the message ordering is maintained across this change, the sender SHOULD wait until all Acknowledgment messages have been received for the old groupId before starting the new sequence.

(4) status attribute

This OPTIONAL attribute is used to specify status of the group of messages. When this attribute is present, its value MUST be one of the following three:

- **Start**: Indicating the message is the first message for a group of messages.
- **Continue**: Indicating the message is in the middle of a group of messages.
- **End**: Indicating the message is the last message for a group of messages.

The sender node MUST send a very first message, to guarantee the order, with “Start” for this attribute. Also, the sender MUST send subsequent messages for the same series of messages with “Continue”, until the message sent is the last one for the series of messages, for which case the value MUST be “End”. When omitted, the default value for this attribute is “Continue.”

NOTE: Because delivery between the Reliable Messaging Processor and the application is not specified, this is not a complete guarantee of ordering to the application.

3.1.3 ExpiryTime Element

The ExpiryTime element is used to indicate the ultimate time after which the receiver RMP MUST NOT invoke the deliver operation for the received message. This element MUST be present in a MessageHeader element. After a message has been sent for the first time, the value of the ExpiryTime in a message MUST NOT be modified in any manner by the Sending RMP, when resending the message: two messages with same Message Identifier (duplicates) MUST have the same value for ExpiryTime. When a message expires on the Sender side before being successfully sent, a Sender RMP MUST NOT send it or resend it, and MUST communicate a delivery failure to the Sender application. The time MUST be expressed as UTC and MUST conform to a [XML Schema] dateTime. The message is considered expired if the current time, in UTC, is greater than the value of the ExpiryTime element. If a receiver receives an expired message, it MUST send the sender a Fault message with Error code of “InvalidExpiryTime”.

NOTES: Given the above definition of ExpiryTime, in case duplicate elimination is required, when a received message is processed, it is sufficient to only check for its duplicates among IDs of past messages that have not expired yet at the time of the duplicate check.

Table5 ExpiryTime Element

Cardinality	1
Value	dateTime
Attributes	None
Child elements	None
Fault	InvalidExpiryTime

3.1.4 ReplyPattern Element

The ReplyPattern element is used for a sender to indicate what reply pattern is requested. The ReplyPattern element MUST be present in a MessageHeader element. This element is used to specify whether the Acknowledgment message (or Fault message) should be sent back directly in the reply to the reliable message, in a separate callback request, or in the response to a separate poll request. This element MUST have one of the following three values:

- **Response** : An Acknowledgment message or Fault message MUST be sent back directly in the response to the Reliable Message. This pattern is not applicable for one-way application level MEP. The acknowledgment message that can be sent back in the response is for the message in the request message only. Acknowledgment message for the former request MUST NOT be sent back.
- **Callback**: An Acknowledgment message (or Fault message) MUST be sent as a callback request, using the address in the replyTo attribute. This pattern is not applicable for request-response application level MEP.
- **Poll**: An Acknowledgment message (or Fault message) MUST be sent as a response to a poll request. This pattern is not applicable for request-response application level MEP.

The value of this element in MessageHeader of the reply MUST be the same as that of the Request.

The ReplyPattern element contains the following OPTIONAL attribute:

– a **replyTo** attribute

Table6 ReplyPattern Element

Cardinality	1
Value	String : Response, Callback, or Poll

Cardinality	1
Attributes	replyTo (URL)
Child elements	None
Fault	InvalidReplyPattern

(1) replyTo attribute

This is an OPTIONAL attribute, used to specify the initial sender's endpoint to receive a callback Acknowledgment message or Fault Message. A value of this attribute MUST be present if the ReplyPattern element value indicates that the Callback reply pattern is requested.

If present, the replyTo attribute is required to be URL as defined in [RFC 1738].

3.2 Request Element

The Request element MUST be present in a Reliable Message. It includes specific information to be used for a Reliable Message. All messages in a group MUST have the same values for the three Reliable Messaging qos parameters(AckRequested, DuplicateElimination and MessageOrder) in their Message header. The Request element includes the following attributes and child elements:

- SOAP **mustUnderstand** attribute with a value of "1"

- **AckRequested** element

- **DuplicateElimination** element

- **MessageOrder** element

Table7 Request Element

Cardinality	0 or 1 * See 3.2 for details
Value	None
Attributes	mustUnderstand
Child elements	AckRequested DuplicateElimination MessageOrder
Fault	InvalidRequest

Example 6 shows an example of Request element.

Example 6 Request Element

(Example to be included later)

3.2.1 AckRequested Element

The AckRequested element MUST be present for guaranteeing message delivery and message ordering. If the MessageOrder element is present, the AckRequested element MUST also be present. This element is used by a sender to request the receiver to send back an Acknowledgment or Fault message for the message sent. If a receiver receives a message with AckRequested element, the receiver MUST send an Acknowledgment message even when the message is a duplicate.

The pattern used to send the Acknowledgment or Fault message is based on the value of the ReplyPattern element.

Table8 AckRequested Element

Cardinality	0 or 1
Value	None
Attributes	None
Child elements	None
Fault	InvalidAckRequested

3.2.2 DuplicateElimination Element

The DuplicateElimination element is used to request the receiver node to identify duplicate messages it has received and process them accordingly (see section 2.6). A duplicate message is a message with the same Message Identifier as another message. The sender MUST include DuplicateElimination element if the MessageOrder element is present.

Table9 DuplicateElimination Element

Cardinality	0 or 1
Value	None
Attributes	None
Child elements	None
Fault	InvalidDuplicateElimination

3.2.3 MessageOrder Element

The MessageOrder element is OPTIONAL element. This element is used to request the receiver node to invoke delivery operation with the same order that the sender has submitted. When a sender submits multiple messages with Guaranteed Message Ordering, the sender MUST include the MessageOrder element in the message. All messages to be delivered in order MUST have same groupId and MUST have sequence number as a value of SequenceNum element in order of the message to be delivered to receiver's application. When the sender requests Guaranteed Message Ordering, the sender MUST use Guaranteed message delivery and duplicate elimination for that message as well. In particular, the sender MUST include both AckRequested element and DuplicateElimination element, as well as the MessageOrder element for Guaranteed Message Ordering.

Table10 MessageOrder Element

Cardinality	0 or 1
Value	None
Attributes	None
Child elements	None
Fault	InvalidMessageOrder

3.3 PollRequest Element

(To be added later)

The PollRequest Element is an OPTIONAL element. This element is used only in the PollRequest message as shown in the Figure5. The PollRequest message contains two direct child elements for SOAP Header element. The one is MessageHeader element, and the other is the PollRequest element. The PollRequest message is used to query Acknowledgment message for specific message. Typically, the PollRequest message is to receive Acknowledgment message for a message sent with Polling RM-Reply Pattern.

This element includes the following child element:

- a **RefToMessageIds** element

Table11 PollRequest Element

Cardinality	0 or 1
Value	None
Attributes	mustUnderstand
Child elements	RefToMessageIds
Fault	InvalidPollRequest InvalidMessageId InvalidSequenceNum

3.3.1RefToMessageIds element

The RefToMessageIds element MUST be present for PollRequest message. This element is to be used to specify Acknowledgment message(s) to be returned. This element MUST have one groupId attribute and MAY contain zero or more SequenceNumRange element as follows:

- a **groupId** attribute

- zero or more **SequenceNumRange** element

Table12 RefToMessageIds Element

Cardinality	1 or more
Value	None
Attributes	groupId (URI)
Child elements	SequenceNumRange
Fault	InvalidMessageId

When this RefToMessageIds element has a groupId attribute, but doesn't have SequenceNumRange element, the receiver MUST send back all Acknowledgment messages for the messages received in the MessageId. When the RefToMessageIds element has a groupId attribute and SequenceNumRange element(s), the receiver MUST return Acknowledgment message for messages received that were specified by the combination of groupId of RefToMessageIds and SequenceNumRange element(s). When sender RMP requests multiple Acknowledgment messages with different groupId value in one PollRequest Message, it MUST include RefToMessageIds element for each groupId.

(1) groupId attribute

The groupId attribute MUST be present in the RefToMessageIds element. The groupId attribute is to be used to specify the groupId for Acknowledgment message to be returned. The syntax of this attribute is URI, as defined in [RFC2396].

3.3.1.1 SequenceNumRange element

The SequenceNumRange element is an OPTIONAL element. This element MUST contain the value of the SequenceNum of the message. This element MUST contain the following two attributes:

- a **from** attribute

- a **to** attribute

Table13 SequenceNumRange Element

Cardinality	0 or more
Value	None
Attributes	from (unsignedLong) to (unsignedLong)
Child elements	None

Cardinality	0 or more
Fault	InvalidSequeceNumberRange

(1) from attribute

This from attribute is to be used to specify the smallest SequenceNum of the message range. The value of this attribute is unsignedLong.

(2) to attribute

This from attribute is to be used to specify the largest SequenceNum of the message range. The value of this attribute MUST be the same with the value of the from attribute to specify only one message. The value of this attribute is unsignedLong.

3.4 Response Element

The Response element includes response information to be used for an Acknowledgment messages. This element MUST be present only when the message includes an Acknowledgment messages. This element includes the following attribute and child elements:

- SOAP **mustUnderstand** attribute with a value of “1”

- **RefToMessageIds** element

When using the callback reply pattern, if the reply and the new request share a common destination URI, a Response element can coexist with a Request element, enabling the combination of an Acknowledgment message with the business response to the original message. This coexistence also enables a receiver sending another independent message to the sender with an Acknowledgment message (e.g., to reduce network traffic).

Table14 Response Element

Cardinality	0 or 1
Value	None
Attributes	mustUnderstand
Child elements	RefToMessageIds
Fault	InvalidResponse

Example 7 shows an example of the Response element.

Example 7 Response Element

(Example will be added later, when the schema is decided)

3.4.1 RefToMessageIds Element

The RefToMessageIds element MUST be present in an Acknowledgment message or Fault message. This element MUST contain the value of the original MessageId of the message received successfully when used in an Acknowledgment message, or for the message in error, when used in a Fault Message.

Table15 RefToMessageIds Element

Cardinality	1 or more
Value	RFC2396
Attributes	groupId (URI)
Child elements	SequenceNumRange
Fault	InvalidRefToMessageIds

(1) groupId attribute

This groupId attribute is to be used to specify the group of message(s) to be acknowledged. The syntax of this attribute is URI, as defined in [RFC2396].

3.4.1.1 SequenceNumRange Element

The SequenceNumRange element MUST be present for an Acknowledgment or Fault message when the original message was delivered with Guaranteed Message Ordering. This element MUST contain the value of the original SequenceNum of the message received successfully when used in an Acknowledgment message, or for the message in error, when used in a Fault Message.

Table16 SequenceNumRange Element

Cardinality	1 or more
Value	None
Attributes	from (unsignedLong) to (unsignedLong)
Child elements	None
Fault	InvalidRefToSequeceNumber Range

4 Fault Processing

This section describes the protocol specific fault codes that are needed to better describe the reason for processing failures as SOAP Fault codes are very generic in nature. We categorize the faults into 2 categories based on whether the fault was generated because Reliable Messaging Headers are malformed or invalid or because of the some runtime processing errors. The former category is called Invalid Message Format fault set and the latter is called Request Processing fault set. They are explained in detail in the following sections.

These protocol specific fault codes are sent within a different Header for a SOAP 1.1 messages and for SOAP 1.2 messages, they are sent within the SubCode element of SOAP Fault. SOAP 1.1 disallows using the SOAP detail element to send processing specific faults that happen during the processing of Headers. Such faults should be sent in SOAP Headers itself. And hence for SOAP 1.1 messages, these RM specific fault codes are sent within the (RM) Fault SOAP Header. Since SOAP 1.2 doesn't have this restriction, protocol specific fault codes can be sent within the SubCode element of SOAP Fault Code.

4.1 Fault Codes For Reliable Messaging Failures

All these Fault Codes are of type QName. Prefix for SOAP 1.1 faults is targetNamespace="http://www.oasis-open.org/committees/wsrm/schema/1.1/SOAP1.1". Prefix for SOAP 1.2 faults is targetNamespace="http://www.oasis-open.org/committees/wsrm/schema/1.1/SOAP1.2".

4.1.1 Invalid Message Format Fault

These faults are thrown by the receiving RMP when the message format of the Reliable Messaging Headers are either invalid or wrong.

Chart 2 Invalid Message Format Fault Code Values

Local part name	Description and Cause(s)
InvalidMessageHeader	This fault is sent when the MessageHeader element is wrong or invalid. Examples are: 1.When any of the mandatory elements such as MessageId, ExpiryTime, ReplyPattern are missing 2.SOAP:mustUnderstand attribute is missing
InvalidRequest	This fault is sent when the Request element is wrong or invalid. Examples are: 1.When AckRequested, DuplicateElimination or MessageOrder elements appear twice 2.SOAP:mustUnderstand attribute is missing

InvalidPollRequest	This fault is sent when the PollRequest element is wrong or invalid. Examples are: 1.When the required RefToMessageId element is missing 2.SOAP:mustUnderstand attribute is missing
InvalidMessageId	This fault is sent in any of the following cases: 1. If groupId attribute (for MessageId or RefToMessageIds) doesn't exist, or if exists, and the value is wrong or invalid. 2. If number attribute in SequenceNum element doesn't exist, or if exist, the value is invalid or wrong. 3. Attributes (from and to) of SequenceNumRange doesn't exist, or if exists, the values are invalid or wrong.
InvalidMessageParameters	This fault is sent for any of these cases: 1. groupExpiryTime is wrong or invalid 2. groupMaxIdleDuration is wrong or invalid 3. when both group parameters are present 4. when groupExpiryTime decreases for a subsequent messages. in an ordered group 5. If the status attribute of SequenceNum element exist and is not one of allowed {begin|continue|end} value.
InvalidReplyPattern	This fault is sent if the ReplyPattern format is wrong or invalid or when the replyTo attribute is missing for the Callback pattern.
InvalidExpiryTime	This fault is sent if the ExpiryTime format is wrong or invalid.

For 1.1 SOAP Faults, faultcode (/s11:Envelope/s11:Body/s11:Fault/s11:faultCode) for all the above faults MUST be s11:Client. For 1.2 SOAP Faults, Value of the Code (/s12:Envelope/s12:Body/s12:Fault/s12:Code/s12:SubCode/s12:Value) MUST be s12:Sender

(P.S. s12 is the namespace of SOAP 1.2 envelope http://www.w3.org/2003/05/soap-envelope

and s11 is the namespace of SOAP 1.1 envelope http://schemas.xmlsoap.org/wsdl/)

4.1.2 Message Processing Failure Faults

These faults are thrown by the receiving RMP when there is an error processing a valid Reliable Messaging message.

Chart 3 Messaging Processing Failure Fault Code Values

Local part name	Description and Cause(s)
NonSupportedFeature	This fault is thrown by the receiving RMP when it receives a message with a RM feature that it doesn't support. An example is a RM message with MessageOrder element to a receiving RMP that doesn't support Message Ordering
PermamnentProcessingFailure	This fault is sent for permanent/fatal processing failures such as: 1. Persistence Storage failures 2. Message Delivery failures A PermamnentProcessingFailure fault indicates that the failure is fatal and subsequent retries of the same message will also fail.
MessageProcessingFailure	This fault is sent for transient failures such as: 1. Maximum number of buffered requests exceeded the limit. 2. Maximum number of threads reached the limit etc. A transient fault unlike a permanent fault is a temporary one and MAY succeed in subsequent retries.

For 1.1 SOAP Faults, faultcode (/s11:Envelope/s11:Body/s11:Fault/s11:faultCode) for all the above faults MUST be s11:Sender. For 1.2 SOAP Faults, Value of the Code (/s12:Envelope/s12:Body/s12:Fault/s12:Code/s12:SubCode/s12:Value) MUST be s12:Receiver.

Example 8 Fault Message for Reliable Messaging

(Add examples when schema is completed)

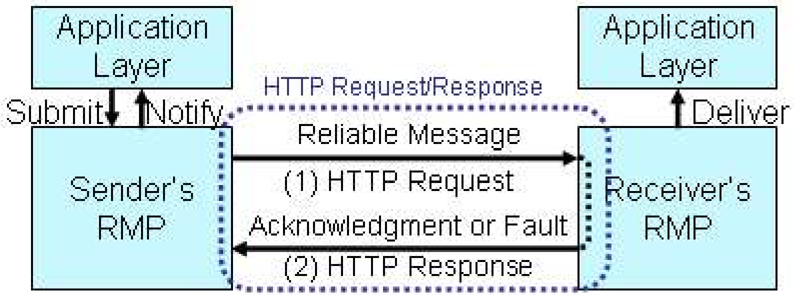
5 HTTP Binding (Needs to include examples)

This section describes the three binding pattern “Reponse”, “Callback”, and “Poll” binding pattern, when the underlying protocol is the HTTP. These binding pattern is identified by the value of ReplyPattern element(See Section3.1.5 for detail). This specification expects that the transport layer will not deliver a corrupted message to the reliability layer. When a request message contains AckRequested element, upon receipt of a reliable message, the receiver's RMP MUST send a reply. This reply MUST be either an Acknowledgment or a Fault message. This reply MUST be sent by specified binding pattern in the ReplyPattern element of the request message.

5.1 Reliable Messaging with “Response” binding pattern

The Reliable Messaging Acknowledgment or Fault message MUST be sent back on the same HTTP connection with the HTTP Request that the sender initiated to send the Message. This is illustrated in Figures 7. Both Acknowledgment Message and Fault message MUST be sent back to the sender on the same HTTP connection the sender sent a message.

Figure 7 Response binding pattern



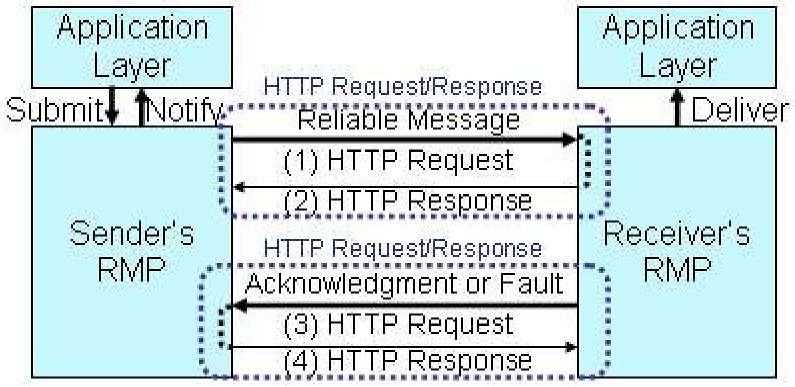
1) The sender initiates an HTTP connection, and sends a Message using the HTTP POST Request. Example 9 is an example of such a message.

2) The receiver sends back an Acknowledgment message to the sender on the same connection, with the HTTP response.

5.2 Reliable Messaging with “Callback” binding pattern

The Reliable Messaging Acknowledgment or Fault message MUST be sent back on a different HTTP connection from the HTTP connection that the sender initiated to send the message. The direction of the HTTP connection that receiver initiates is from the receiver to the sender. This is illustrated in Figure 8.

Figure 8 Callback binding pattern



(1) The sender initiates a HTTP connection, and sends a Message using HTTP POST Request. Example 9 is an example of this message.

(2) The HTTP response to the (1) has no content. Example 10 is an example of this HTTP response.

(3) The Acknowledgment Message is sent with another HTTP connection from the receiver to the sender. Example 11 is an example of this message.

(4) The HTTP response for (3) has no content. Example 10 is an example for this HTTP Response.

Example 9 Request Message with Callback binding pattern

(To be added later after schema is fixed.)

Example 10 HTTP response with no content

(To be added later after schema is fixed.)

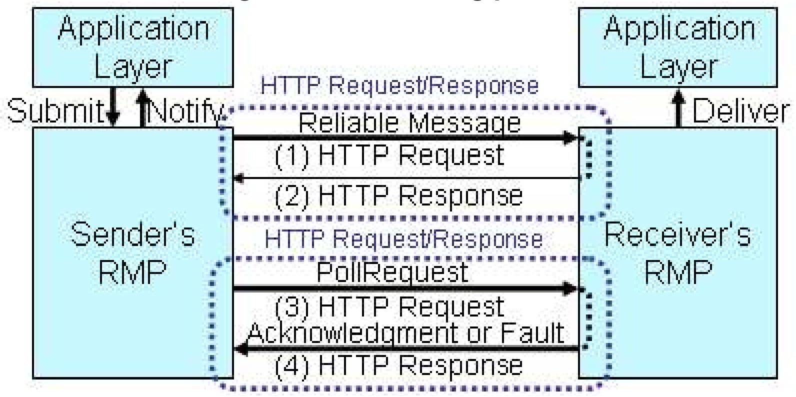
Example 11 Response Message with Callback binding pattern

(To be added later after schema is fixed.)

5.3 Reliable Messaging with “Poll” binding pattern

The Reliable Messaging Acknowledgment message MAY also be sent back on a different HTTP connection from the HTTP connection used to send the message being acknowledged. This is illustrated in Figure 8 and 9.

Figure 9 Poll binding pattern



Example 12 PollRequest with Poll binding pattern

(To be added later after schema was fixed.)

Example 13 Response with Poll binding pattern

(To be added later after schema was fixed.)

6 References

6.1 Normative

[RFC1738] "Uniform Resource Locators (URL)", T. Berners-Lee et al, RFC 1738, IESG and IETF, December 1994. Available at

http://www.ietf.org/rfc/rfc1738.txt

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, Bradner, S., IESG and IETF, March 1997. Available at

http://www.ietf.org/rfc/rfc2119.txt

[RFC2392] “Content-ID and Message-ID Uniform Resource Locators”, RFC2392, E. Levinson, IESG and IETF, August 1998. Available at

http://www.ietf.org/rfc/rfc2392.txt

[RFC2396] "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, Tim Berners-Lee et al, IESG and IETF, August 1998. Available at

http://www.ietf.org/rfc/rfc2396.txt

[RFC2616] "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, R. Fielding et al, IESG and IETF, June 1999. Available at

http://www.ietf.org/rfc/rfc2616.txt

[RFC2822] "Internet Message Format", RFC 2822, P. Resnick, Editor, IESG and IETF, April 2001. Available at

http://www.ietf.org/rfc/rfc2822.txt

[SOAP 1.1] "Simple Object Access Protocol (SOAP) 1.1", Don Box et al, W3C Note, 8 May, 2000. Available at

http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

"Extensible Markup Language (XML) 1.0, Second Edition", Tim Bray et al, eds., W3C Recommendation, 6 October 2000. Available at

http://www.w3.org/TR/2000/REC-xml-20001006/

[XML Namespaces] "Namespaces in XML", Tim Bray et al., eds., W3C Recommendation, 14 January 1999. Available at

http://www.w3.org/TR/1999/REC-xml-names-19990114/

[XML Schema] "XML Schema Part 2: Datatypes", Paul V. Biron and Ashok Malhotra, eds. W3C Recommendation, 2 May 2001. Available at

http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

6.2 Non-normative References

[ebMS] "Message Service Specification Version 2.0", OASIS ebXML Messaging Services Technical Committee, OASIS Standard, 1 April 2002. Available at

http://www.ebxml.org/specs/ebMS2.pdf

[SOAP 1.2] "SOAP 1.2 Part 1: Messaging Framework", Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, eds., W3C Recommendation, 24 June 2003. Available at

http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[WSS] "OASIS Web Services Security TC", documentation in progress, OASIS Technical Committee. Committee home page at

http://www.oasis-open.org/committees/wss/

"XML Schema Part 1: Structures", Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, eds., W3C Recommendation, 2 May 2001. Available at

http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

"SOAP Messages with Attachments", John J. Barton, Satish Thatte, Henrik Frystyk Nielsen, W3C Note, 11 December 2000, Available at

http://www.w3.org/TR/SOAP-attachments

Appendix A. Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Schema for WS-Reliability Protocol Headers
File: ws-reliability.xsd
Version: 0.51
Date: 2004-JAN-16
Editors:
  Sunil Kunisetty
  Scott Werden

Change Log:
0.1: 09/26/03:  Initial version
0.2: 10/01/03:  Added MessageOrder sub-element to Request
                Updated Response element to extend RmBaseType
0.3: 11/03/03:  Based on resolutions from 10/30/03 F2F
                  - Replace removeAfter with MaxGroupExpiryTime
                  - Add MaxIdleTime as an optional attribute to GroupId
                  - Remove Tiemstamp
                  - Add PollRequest Header
                  - Change Response to support batching
0.4: 01/06/04: Based on 0.85 Working Draft
                  - Rename MaxGroupExpiryTime to groupExpiryTime
                  - Rename MaxIdleTime to groupMaxIdleDuration
                  - Lower case all attributes
                  - Fix Fault QName
0.5: 01/16/04:Based on F2F Resolutions
                  - Change groupMaxIdleDuration type to xsd:duration
                  - Add minOccurs=0 to RmBaseType
                  - New Elements (MessageId, RefToMessageIds)
                  - New Fault QNames
```

```
0.51:01/16/04:Change MessageProcessing fault to MessageProcessingFailure
-->

<xsd:schema targetNamespace="http://www.oasis-
open.org/committees/wsrm/schema/1.1/SOAP1.1"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsrm="http://www.oasis-
open.org/committees/wsrm/schema/1.1/SOAP1.1" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="0.5">
    <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
schemaLocation="http://schemas.xmlsoap.org/soap/envelope"/>
    <!-- 4 Main WSRM Headers -->
    <xsd:element name="MessageHeader" type="wsrm:MessageHeaderType"/>
    <xsd:element name="Request" type="wsrm:RequestType"/>
    <xsd:element name="Response" type="wsrm:ResponseType"/>
    <xsd:element name="PollRequest" type="wsrm:PollRequestType"/>
    <xsd:element name="Fault" type="wsrm:FaultCodeEnum"/>
    <!-- Describe the BaseTypes -->
    <xsd:complexType name="RmBaseType">
        <xsd:sequence>
            <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute ref="soap:mustUnderstand" use="required" fixed="1"/>
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:complexType>
    <xsd:complexType name="EmptyType">
        <xsd:sequence/>
    </xsd:complexType>
    <!-- The main MessageHeader Type and it's elements-->
    <xsd:complexType name="MessageHeaderType">
        <xsd:complexContent>
            <xsd:extension base="wsrm:RmBaseType">
                <xsd:sequence>
                    <xsd:element name="MessageId"
type="wsrm:MessageIdType"/>
```

```
                    <xsd:element name="ExpiryTime" type="xsd:dateTime"/>
                    <xsd:element name="ReplyPattern" type="wsrm:ReplyPatternType"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ReplyPatternType">
        <xsd:simpleContent>
            <xsd:extension base="wsrm:ReplyPatternTypeValues">
                <xsd:attribute name="replyTo" type="xsd:anyURI" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:simpleType name="ReplyPatternTypeValues">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Response"/>
            <xsd:enumeration value="Callback"/>
            <xsd:enumeration value="Poll"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="MessageIdType">
        <xsd:complexContent>
            <xsd:extension base="wsrm:EmptyType">
                <xsd:sequence>
                    <xsd:element name="SequenceNumbe" type="wsrm:SequenceNumberType" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="groupId" type="wsrm:MIDType"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:simpleType name="MIDType">
```

```
			<xsd:restriction base="xsd:anyURI"/>
		</xsd:simpleType>
		<xsd:complexType name="SequenceNumberType">
			<xsd:complexContent>
				<xsd:extension base="wsrm:EmptyType">
					<xsd:attribute name="number" type="xsd:unsignedLong"/>
					<xsd:attribute name="status"
type="wsrm:SequenceNumberStatusType" use="optional"/>
					<xsd:attribute name="groupExpiryTime" type="xsd:dateTime"
use="optional"/>
					<xsd:attribute name="groupMaxIdleDuration"
type="xsd:duration" use="optional"/>
				</xsd:extension>
			</xsd:complexContent>
		</xsd:complexType>
		<xsd:simpleType name="SequenceNumberStatusType">
			<xsd:restriction base="xsd:string">
				<xsd:enumeration value="Start"/>
				<xsd:enumeration value="Continue"/>
				<xsd:enumeration value="End"/>
			</xsd:restriction>
		</xsd:simpleType>
		<!-- Request Header Type and it's elements -->
		<xsd:complexType name="RequestType">
			<xsd:complexContent>
				<xsd:extension base="wsrm:RmBaseType">
					<xsd:sequence>
						<xsd:element name="AckRequested"
type="wsrm:EmptyType" minOccurs="0"/>
						<xsd:element name="DuplicateElimination"
type="wsrm:EmptyType" minOccurs="0"/>
						<xsd:element name="MessageOrder"
type="wsrm:EmptyType" minOccurs="0"/>
					</xsd:sequence>
				</xsd:extension>
			</xsd:complexContent>
```

```
	</xsd:complexType>
	<!-- Response Header Type and it's elements -->
	<xsd:complexType name="ResponseType">
		<xsd:complexContent>
			<xsd:extension base="wsrm:RmBaseType">
				<xsd:sequence>
					<xsd:element name="RefToMessageIds"
type="wsrm:RefToMessageIdsType" maxOccurs="unbounded"/>
				</xsd:sequence>
			</xsd:extension>
		</xsd:complexContent>
	</xsd:complexType>
	<xsd:complexType name="RefToMessageIdsType">
		<xsd:complexContent>
			<xsd:extension base="wsrm:EmptyType">
				<xsd:sequence>
					<xsd:element name="SequenceNumberRange"
type="wsrm:SequenceNumberRangeType" minOccurs="0" maxOccurs="unbounded"/>
				</xsd:sequence>
				<xsd:attribute name="groupId" type="wsrm:MIDType"/>
			</xsd:extension>
		</xsd:complexContent>
	</xsd:complexType>
	<xsd:complexType name="SequenceNumberRangeType">
		<xsd:complexContent>
			<xsd:extension base="wsrm:EmptyType">
				<xsd:attribute name="from" type="xsd:unsignedLong"/>
				<xsd:attribute name="to" type="xsd:unsignedLong"/>
			</xsd:extension>
		</xsd:complexContent>
	</xsd:complexType>
	<!-- Poll Request Header Type -->
	<xsd:complexType name="PollRequestType">
		<xsd:complexContent>
```

```
				<xsd:extension base="wsrm:RmBaseType">
					<xsd:sequence>
						<xsd:element name="RefToMessageIds" 
type="wsrm:RefToMessageIdsType" maxOccurs="unbounded"/>
					</xsd:sequence>
				</xsd:extension>
			</xsd:complexContent>
		</xsd:complexType>
		<!-- Fault Codes Allowed-->
		<xsd:simpleType name="FaultCodeEnum">
			<xsd:restriction base="xsd:QName">
				<xsd:enumeration value="wsrm:InvalidMessageHeader"/>
				<xsd:enumeration value="wsrm:InvalidMessageId"/>
				<xsd:enumeration value="wsrm:InvalidMessageParameters"/>
				<xsd:enumeration value="wsrm:InvalidPollRequest"/>
				<xsd:enumeration value="wsrm:InvalidRequest"/>
				<xsd:enumeration value="wsrm:InvalidExpiryTime"/>
				<xsd:enumeration value="wsrm:InvalidReplyPattern"/>
				<xsd:enumeration value="wsrm:NotSupportedFeature"/>
				<xsd:enumeration value="wsrm:PermamnentProcessingFailure"/>
				<xsd:enumeration value="wsrm:MessageProcessingFailure"/>
			</xsd:restriction>
		</xsd:simpleType>
</xsd:schema>
```

Appendix B. Acknowledgments

The following individuals were members of the committee during the development of this specification:

David Ingham, Arjuna Technologies Limited

Joseph Chiusano, Booz Allen Hamilton

Peter Furniss, Choreology Ltd

Jeff Turpin, Cyclone Commerce

Pramila Mullan, France Telecom

Jacques Durand, Fujitsu

Kazunori Iwasa, Fujitsu

Tom Rutt (chair), Fujitsu

Jishnu Mukerji, Hewlett-Packard

Robert Freund, Hitachi

Eisaku Nishiyama, Hitachi

Nobuyuki Yamamoto, Hitachi

Ben Bloch, Individual

Mark Hansen, Individual

Paolo Romano, Individual

Dock Allen, Mitre Corporation

Junichi Tatemura, NEC Corporation

Alan Weissberger, NEC Corporation

Magdolna Gerendai, Nokia

Szabolcs Payrits, Nokia

Sunil Kunisetty, Oracle

Jeff Mischkinsky, Oracle

Marc Goodner, SAP

Pete Wenzel, SeeBeyond Technology Corporation

Doug Bunting, Sun Microsystems

Chi-Yuen Ng, University of Hong Kong

Patrick Yee, University of Hong Kong

Prasad Yendluri, webMethods, Inc.

Scott Werden, WRQ, Inc.

Tony Graham, Sun Microsystems Ireland Ltd

In addition, the following people made contributions to this specification:

(TBD)

Appendix C. Revision History

[This appendix is optional, but helpful. It should be removed for specifications that are at OASIS Standard level.]

Rev	***Date***	***By Whom***	***What***
WD-0.5	2003-09-04	Kazunori Iwasa	Initial version
WD-0.51		Kazunori Iwasa	Editorial update
WD-0.52		Kazunori Iwasa	Editorial update
WD-0.54	-2003-10-23	Kazunori Iwasa	Issue Rel-38 : Section 3.1.3 Timestamp Issue Rel-98 : Section 3.1.2 and 3.2.3 Issue Rel-40 : Section 3.1.4 Issue Rel-88 : Section 3.1.1 Issue Rel-16 : Section 3.2.1 to 3.2.3 Issue Rel-14 : Appendix C Editorial update
WD-0.60	-2003-10-28	Kazunori Iwasa	Editorial update at F2F in South SF.
WD-0.70	-2003-10-30	Kazunori Iwasa	Section2: Messaging models Section3: Message Format, and others Section4: PollRequest Section5: Binding patterns Editorial update

Rev	*Date*	*By Whom*	*What*
WD-0.83	-2003-11-18	Kazunori Iwasa	Section2.6: Added description of Figure3 Section3: Added tables for each element Rel-31: Section2.5 Rel-38: Timestamp was removed from Section 3 Rel-100: Added Section2.9 Attachments Rel-32: Added definitions to Section1.8 Rel-94: Figure5 and Section 3.3 (Needs additional descriptions and examples in Section3.3) Editorial updates, especially for : http://lists.oasis-open.org/archives/wsrm/200310/msg00054.html All editorial comments above are incorporated except one, which is a comment for line 357, to keep consistency with other sections.
WD-0.84	-2003-12-15	Kazunori Iwasa	Rel-33:Section 1.8: Update on Message Delivery and Acknowledgment Message Rel-50:Section 3.1.3 ExpiryTime Editorial updates
WD-0.85	-2004-01-06	Kazunori Iwasa	Section2.4, Section2.5, and Section 3.1.1 are updated to incorporate resolutions for Rel-52, Rel-57, and Rel-82.

Rev	*Date*	*By Whom*	*What*
WD-0.86	-2004-01-14	Kazunori Iwasa	Updated for comments at : http://www.oasis-open.org/archives/wsrm/200401/msg00010.html except for: - More faults for Tables1 (Need to list up all faults) - Section2.4 Line#259 in Spec20040106 (Ver0.85): It should read "after the message has been processed and delivered to the "next processing layer". (Need to confirm with TC for this change, since the current text was approved one.) - Figure1,2,and3 “New processor Entity” (Want to confirm with TC member) -New terminologies for “Group Termination”, “Removal”, “Complete”, and others. (Needs definitions) -- And other editorial updates.

Rev	*Date*	*By Whom*	*What*
WD-0.87	-2004-01-15	Kazunori Iwasa	Updated for: Prelim Wed minutes on 1/14/2004 at: http://www.oasis-open.org/archives/wsrm/200401/msg00038.html. It includes: Rel33: New definitions in 1.8(deliver, submit, notify) Rel37: editorial change in 3.1.1 Rel40: editorial change in 3.1.3 Rel44: updates for 3.1.1 Rel51: change definition for Message Acknowledgment Rel52: Moved some of 3.1.1(line546-571) to 2.5.1 Rel98: removed informative notes in 2.4 Tables: Changed “Required” to “Cardinality” (Yes-1, No-0) The following resolutions are not updated yet: Rel 83-86 and 56: Change of element names and location (Eg. GroupId -> MessageId)

Rev	***Date***	***By Whom***	***What***
WD-0.88	-2004-01-16	Kazunori Iwasa	Updated for: 1) Prelim minutes for Thursday on 1/15/2004 at: http://www.oasis-open.org/apps/org/workgroup/wsrm/email/archives/200401/msg00053.html 2) Remaining items including: Rel36: Message ID -> Message Identifier Rel37: Reference for RFC2392 Changed element names and location (Eg. GroupId -> MessageId) And others. The following items are still in progress: Rel22: usage for MUST, MAY, Should, Optional

Rev	***Date***	***By Whom***	***What***
WD-0.90	-2004-01-26	Kazunori Iwasa	Updated for remaining action items at: http://www.oasis-open.org/apps/org/workgroup/wsrm/download.php/5089/Minutes-Jan04f2f.htm This includes : 1) 2.4: Message Identifier -> MessageId Sequence Number -> SequenceNum Included “Next processing layer” 2) 2.11: Chart is updated (Now2.9) 3) 3.1.1and 3.1.2: two group attributes were moved from MessageId to SequenceNum 4) 3.1.4 Response : Some sentences are added to restrict sending back previous Acknowledgment message in the other Response. 5) 3.3, 3.3.1 and 3.4.1: MessageId -> RefToMessageIds value -> groupId 6) Section4: Replaced with new text 7) Appendix A:Replaced with new schema Other changes includes: Cover page: location of the spec and errata were added. Section 1 and 2: Editorial review 1.9:New section was added (RM agreement) 1.6: Description for WS-I BP1.0 was included. 6.2: Added non-normative Reference for SOAP messages with Attachments

Rev	*Date*	*By Whom*	*What*
			Remaining Action items and editorial changes for 2004-01-26(0.90): 1. Using same word: e.g. sender RMP, sending RMP or sender's RMP 2.Removing MAY, Optional 3.NotSupportedFeaturesFault 4.Explanatin for cardinality and others 5.SOAP1.2 statement in the new section in 3 on rm:fault element 6.Update fault in section3 7.Examples

Appendix D. Futures List

The features and issues in the table below are listed as forward-looking statements regarding possible enhancements or the evolution of this specification.

	Category	***Details***
1	WSDL	Define WSDL extensions profiling the use of RM SOAP extensions.

Appendix E. Notices