OASIS ebXML Messaging Services Version 3.0: Part 2, Advanced Features

Committee Draft 01

30 June 2010

Specification URIs:

This Version:

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/part2/201004/ebms-v3-part2-cd-01.html

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/part2/201004/ebms-v3-part2-cd-01.odt

(Authoritative)

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/part2/201004/ebms-v3-part2-cd-01.pdf

Previous Version:

N/A

Latest Version:

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/part2/201004/ebms-v3-part2.html

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/part2/201004/ebms-v3-part2.odt

(Authoritative)

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/part2/201004/ebms-v3-part2.pdf

Technical Committee:

OASIS ebXML Messaging Services Technical Committee

Chair(s):

Ian Jones, British Telecommunications plc <ian.jones@bt.com>

Editor(s):

Jacques Durand, Fujitsu Limited <jdurand@us.fujitsu.com>
Sander Fieten, Individual <sander@fieten-it.com>
Pim van der Eijk, Sonnenglanz Consulting BV <pvde@sonnenglanz.net>

Related Work:

This specification is related to:

- OASIS ebXML Messaging Services Version 2.0.
- OASIS ebXML Messaging Services Version 3.0 Part 1: Core Features.
- SOAP 1.1, 1.2
- Web Services Security: SOAP Message Security 1,0, 1.1
- WS-Reliability 1.1;
- WS-ReliableMessaging 1.2
- WS-Addressing 1.0
- WS-I BP 1.1, 1.2 and 2.0; SSBP; AP 1.0; BSP 1.0, 1.1; RSP 1.0

Declared XML Namespace(s):

http://docs.oasis-open.org/ebxml-msg/ns/ebms/v3.0/multihop/200902/
http://docs.oasis-open.org/ebxml-msg/ns/v3.0/mf/2010/04/

Abstract:

This specification complements the ebMS 3.0 Core Specification by specifying advanced messaging functionality for message service configuration, message bundling, messaging across intermediaries (*multi-hop*) and transfer of (compressed) messages as series of smaller message fragments.

Status:

This document was last revised or approved by the ebXML Messaging Services Technical Committee on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/ebxml-msg/.

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 Technical Committee web page (http://www.oasis-open.org/committees/ebxml-msg/ipr.php).

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/ebxml-msg/.

Notices

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS" is a trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

1 Introduction

This specification complements the ebMS 3.0 Core Specification by specifying advanced messaging functionality for:

- messaging across intermediaries (*multihop*),
- message bundling,
- compressing, splitting and joining large messages,
- and advanced processing modes.

1.1 Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in [RFC2119].

1.2 Normative References

[AS2-RESTART] IETF Informational Document. T. Harding, ed. *AS2 Restart for Very Large Messages.* April 2010. http://www.ietf.org/id/draft-harding-as2-restart-01.txt

[EBMS3CORE] OASIS Standard, *OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features,* October 2007, http://www.oasis-open.org/committees/download.php/24618/ebms_core-3.0-spec-cs-02.pdf.

[EBBPSIG] OASIS Standard, *OASIS ebXML Business Signals Schema*, December 2006, http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0

[HTTP11] IETF RFC. R. Fielding, et al, *Hypertext Transfer Protocol -- HTTP/1.1*, 1999. http://www.ietf.org/rfc/rfc2616.txt

[RFC2119] IETF RFC. S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt

[RFC2392] IETF RFC. E. Levinson, Content-ID and Message-ID Uniform Resource Locators, 1998. http://www.ietf.org/rfc/rfc2392.txt

[RFC3987] IETF RFC. M. Duerst. Internationalized Resource Identifiers (IRIs). RFC 3987, January 2005. http://www.ietf.org/rfc/rfc3987.txt

[RFC5246] IETF RFC. T.Dierks et al. The Transport Layer Security (TLS) Protocol Version 1.2. http://www.ietf.org/rfc/rfc5246.txt

[MTOM] W3C Recommendation. M. Gudgin et al. SOAP Message Transmission Optimization Mechanism. January 2005. http://www.w3.org/TR/soap12-mtom/

[SOAP11] W3C Note. D. Box, et al, *Simple Object Access Protocol (SOAP) 1.1*, 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[SOAP12] W3C Recommendation. M. Gudgin, et al, *SOAP Version 1.2 Part 1: Messaging Framework. Second Edition*, April 2007. http://www.w3.org/TR/soap12-part1/

[SOAPATTACH] W3C Note. J. Barton, et al, *SOAP Messages with Attachments*, 2000. http://www.w3.org/TR/SOAP-attachments

[WSADDRCORE] W3C Recommendation. M. Gudgin et al, eds. *Web Services Addressing 1.0 – Core*, May 2006. http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

[WSADDRSOAP] W3C Recommendation. M. Gudgin et al, eds. *Web Services Addressing 1.0 - SOAP Binding*. May 2006. http://www.w3.org/TR/ws-addr-soap/

[WSIAP10] WS-I Final Material. Chris Ferris, et al, eds, *Attachments Profile Version 1.0*, 2004. http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html

[WSIBSP10] WS-I Final Material. Abbie Barbir, et al, eds, *Basic Security Profile Version 1.0*, 2005. http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html

[WSR11] OASIS Standard, *WS-Reliability 1.1*, November 2004, http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf

[WSRM11] OASIS Standard, *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1*, January 2008, http://docs.oasis-open.org/ws-rx/wsrm/v1.1/wsrm.pdf

[WSSC13] OASIS Standard, *WS-SecureConversation 1.3,* March 2007. http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.pdf

[WSS10] OASIS Standard, *Web Services Security: SOAP Message Security 1.0*, March 2004. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[WSS11] OASIS Standard, *Web Services Security: SOAP Message Security 1.1*, February 2006, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[WST13] OASIS Standard, *WS-Trust 1.3,* March 2007, http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.pdf

[XDM] W3C Recommendation. Mary Fernández, et al, eds. *XQuery 1.0 and XPath 2.0 Data Model (XDM)*. January 2007. http://www.w3.org/TR/xpath-datamodel/

[XML10] W3C Recommendation. Tim Bray, et al, eds., *Extensible Markup Language (XML) 1.0 (Third Edition)*, February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/

[XMLDSIG] W3C Recommendation. Donald Eastlake, et al, eds, *XML-Signature Syntax and Processing*, 2002. http://www.w3.org/TR/xmldsig-core/

[XMLENC] W3C Recommendation. D. Eastlake, et al, *XML Encryption Syntax and Processing*, December, 2002. http://www.w3.org/TR/xmlenc-core/

[XMLNS] W3C Recommendation. Tim Bray, et al, eds, *Namespaces in XML*, August 2006. http://www.w3.org/TR/REC-xml-names/

[XMLSCHEMA-P1] W3C Recommendation. Henry S. Thompson, et al, eds., *XML Schema Part 1: Structures Second Edition*, October 2004. http://www.w3.org/TR/xmlschema-1/

[XMLSCHEMA-P2] W3C Recommendation. Paul Biron and Ashok Malhotra, eds. *XML Schema Part 2: Datatypes Second Edition*, October 2004. http://www.w3.org/TR/xmlschema-2/

1.3 Non-normative References

[AS4] OASIS Committee Specification 01, *AS4 Profile of ebMS V3 Version 1.0,* April 2010, http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/profiles/200707/AS4-profile.pdf

[ebCPPA 3.0] OASIS Working Draft, *ebXML Collaboration Protocol Profiles and Agreements, Version 3.0. Editor's Draft,* October 2009. http://www.oasis-open.org/committees/download.php/31996/ebcppa-v3.0-Spec-wd-r01-en-pete4.pdf

[HL7ebMSv3] P. Knapp. *HL7 Version 3 Standard: Transport Specification - ebXML*, Release 2. July 2007. <http://www.hl7.org/v3ballot2007sep/html/infrastructure/transport/transport-ebxml.htm>.

[PARTYIDTYPE] OASIS Public Review Draft, *ebCore Party Id Type Technical Specification Version 1.0,* April 2010, http://docs.oasis-open.org/ebcore/PartyIdType/v1.0/PartyIdType-1.0.odt

[RFC4130] IETF RFC. D. Moberg and R. Drummond. *MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2).* IETF, July 2005. <http://www.ietf.org/rfc/rfc4130.txt>.

[WSIBP11] WS-I Final Material. *Basic Profile Version 1.1.* April 2006. <http://www.ws-i.org/Profiles/BasicProfile-1.1.html >

[WSIBP12] WS-I Working Group Approval Draft. *Basic Profile Version 1.2.* June 2010. <http://ws-i.org/profiles/BasicProfile-1.2-WGD.html>

[WSIBP20] WS-I Working Group Approval Draft. *Basic Profile Version 2.0.* June 2010. <http://ws-i.org/profiles/BasicProfile-1.2-WGD.html >

[WSIBSP11] WS-I Final Material. Basic Security Profile Version 1.1. January 2010. <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html>

[WSIRSP10] WS-I Working Group Approval Draft. Reliable Secure Profile Version 1.0. Working Group Approval Draft. June 2010. < http://www.ws-i.org/profiles/attach_5_reliable_secure_profile-version1_0(WGAD).htm >

[WSMC] OASIS Standard, *Web Services MakeConnection (WS-MakeConnection) Version 1.0.* January 2008. http://docs.oasis-open.org/ws-rx/wsmc/v1.0/wsmc.html

[XPATH] W3C Recommendation. James Clark, et al, eds., *XML Path Language (XPath) Version 1.0*, November, 1999. <http://www.w3.org/TR/xpath>

[XSLT] W3C Recommendation. M. Kay. XSL Transformations (XSLT). Version 2.0. January 2007. <http://www.w3.org/TR/xslt20/>.

1.4 Namespaces

Prefix	Namespace	Specification
ds	http://www.w3.org/2000/09/xmldsig#	[XMLDSIG]
eb3	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/	[EBMS3CORE]
ebint	http://docs.oasis-open.org/ebxml-msg/ns/ebms/v3.0/multihop/200902/	This specification - multihop routing by intermediaries
ebbp	http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0	[EBBPSIG]
mf	http://docs.oasis-open.org/ebxml-msg/ns/v3.0/mf/2010/04/	This specification - message fragments
S11	http://schemas.xmlsoap.org/soap/envelope	[SOAP11]
S12	http://www.w3.org/2003/05/soap-envelope	[SOAP12]
wsa	http://www.w3.org/2005/08/addressing	[WSADDRCORE]
wsmc	http://docs.oasis-open.org/ws-rx/wsmc/200702	[WSMC]
wsr	http://docs.oasis-open.org/wsrm/2004/06/ws-reliability-1.1.xsd	[WSR11]
wsrm	http://schemas.xmlsoap.org/soap/envelope/	[WSRM11]
wssc	http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512	[WSSC13]
wsse	http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd	[WSS11]
wst	http://docs.oasis-open.org/ws-sx/ws-trust/200512	[WST13]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-	[WSS10]

	wss-wssecurity-utility-1.0.xsd	

2 Multi-hop Messaging

2.1 Introduction

The OASIS ebXML Messaging Services (ebMS) Version 3.0 core specification [EBMS3CORE] defines an advanced Web Services-based message protocol that leverages standards for SOAP-based security and reliability. It supports and extends the core functionality of version 2.0 of ebMS. The core specification is focused on point-to-point exchange of messages between two ebMS message service handlers. It does not explicitly consider multi-hop messaging. Messaging across intermediaries is a common requirement in many e-business and e-government communities and is functionality provided by many messaging protocols, including the version 2.0 of ebXML Messaging.

This chapter defines a multi-hop profile of ebMS 3.0 that extends the functionality of the version 3.0 ebMS core specification to multi-hop messaging across ebMS intermediaries. The main function of intermediaries as defined in this specification is to provide message routing and forwarding based on standardized SOAP message headers, allowing a sending MSH to ignore the ultimate message destination and abstract away from lower-level transport parameters such as the URL of the ultimate receiving MSH and message exchange pattern bindings. The intermediary functionality defined here supports message relaying across segmented networks, synchronous and asynchronous bindings and both active (push) and passive (pull message stores) forwarding styles. Multi-hop paths may consist of any number of intermediaries.

A key end-user requirement that this specification supports is end-to-end reliable messaging and end-to-end security and compliance with Web Services interoperability profiles.

2.2 Terminology

The following definitions are used throughout this section:

Forwarding role: The ebMS V3 Core specification defines two roles in which a MSH can act: “Sending” and “Receiving”. In this extension of the messaging model to multi-hop messaging, an ebMS V3 MSH can also act in a new role called “**Forwarding**”. An MSH acting in the Forwarding role forwards a received ebMS message based on its ebXML header content to another MSH without modifying the SOAP message or any attached payload. Section 2.5.2 describes these message forwarding models in more detail.

Intermediary MSH (or ebMS Intermediary): An MSH acting in the new Forwarding role and configured for doing so for at least some messages, in a network of MSHs. ebMS Intermediaries support a **routing function** that maps messages based on header content to the next MSH destination or a pull channel as described in section 2.5 .

Endpoint MSH: An MSH that is able to act either in the Sending role or in the Receiving role, and that is configured for doing so for at least some messages, in a network of MSHs. The ability to act as a Sender or Receiver in a multi-hop transfer imposes certain requirements on the MSH which are detailed in section 2.6 .

NOTE: an Endpoint MSH may also act as an Intermediary MSH: Sending, Receiving and Forwarding roles can be combined in any way, subject to configuration.

ebMS Multi-hop path: a multi-hop path is a sequence of MSHs, starting with an Endpoint MSH and ending with an Endpoint MSH, connected via one or more ebMS Intermediaries, which are configured to allow the end-to-end transfer of some ebMS messages from one Endpoint MSH (called **path origin**) to the other Endpoint MSH (called **path destination**). The following figure illustrates two multi-hop paths between MSHs A and B: one from A to B and another one in the reverse direction from B to A. The components I_0 to I_N in between MSHs A and B are ebMS Intermediaries.

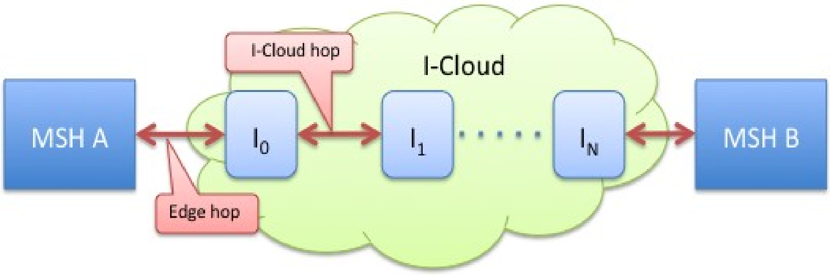

Figure 1: Edge hops and I-Cloud hops

ebMS multi-hop topology: An ebMS multi-hop topology is a network of ebMS nodes connected via one or more multi-hop paths. Note that not every pair of ebMS nodes in a multi-hop topology has to be part of the same multi-hop path, i.e. the topology does not require to enable message transfer from any ebMS node to any ebMS node. In a multi-hop topology one usually finds MSHs that are only able to act as Endpoints on its periphery, although this is not always the case: for example, in a ring topology all MSHs are Intermediaries that can also act as Endpoints for some multi-hop paths.

I-cloud (or Intermediary-cloud): The I-cloud is the network of ebMS Intermediaries that is at the core of a multi-hop topology. The I-cloud does not comprise those Endpoint MSHs that are neither capable nor configured to act as Intermediaries (Forwarding role). However, when considering a single multi-hop path, we will call I-Cloud the set of Intermediaries involved in this path at the exclusion of the origin and destination Endpoint MSHs (even if these are able to act as Intermediaries for another multi-hop path).

The hops that relate the Endpoint MSHs to the I-Cloud (i.e. hop: MSH A - Intermediary 0, and hop Intermediary N – MSH B) are called **Edge-hops**, while the hops over the I-Cloud are called **I-Cloud hops**. The ebMS Intermediaries that participate in the Edge-hops (I0 and IN in the Figure) are called **Edge Intermediaries.**

2.3 Multi-hop Topologies

This multi-hop profile is designed to support multiple multi-hop topologies. This chapter describes some typical topologies.

2.3.1 Assumptions about Multi-hop Topologies

The following assumptions are made about Intermediaries, which further define the multi-hop topologies supported here in a way that is considered relevant to the majority of situations:

- The topologies considered here all involve ebMS intermediaries, not exclusive of other non-ebMS nodes. Other nodes (SOAP nodes, HTTP proxies, etc.) may be involved in transferring ebMS messages over multi-hop paths, but they are not considered as ebMS intermediaries in the sense that they are not required to understand any of the ebMS data or metadata available in the headers and are not supposed to behave depending on this data. Their presence is orthogonal to the definition of ebMS multi-hop topologies.

- The same MSH may play different roles for different multi-hop paths: it can be an Intermediary for some messages, a destination Endpoint for others, and an origin Endpoint for others. The multi-hop model described here must support this, although in practice many topologies will restrict the roles that an MSH can play. For simplicity we will assume that in a Hub-and-Spoke model as well as in the Interconnected-Hubs model, the Endpoints are not acting as Intermediaries.

2.3.2 Hub-and-Spoke

In the Hub-and-Spoke topology, a single Intermediary MSH (called Hub) is used, to which all Endpoint MSHs are connecting. In this configuration, every multi-hop path is actually a 2-hop path. Every Endpoint MSH connected to the Hub is either a destination or an origin to at least one multi-hop path.

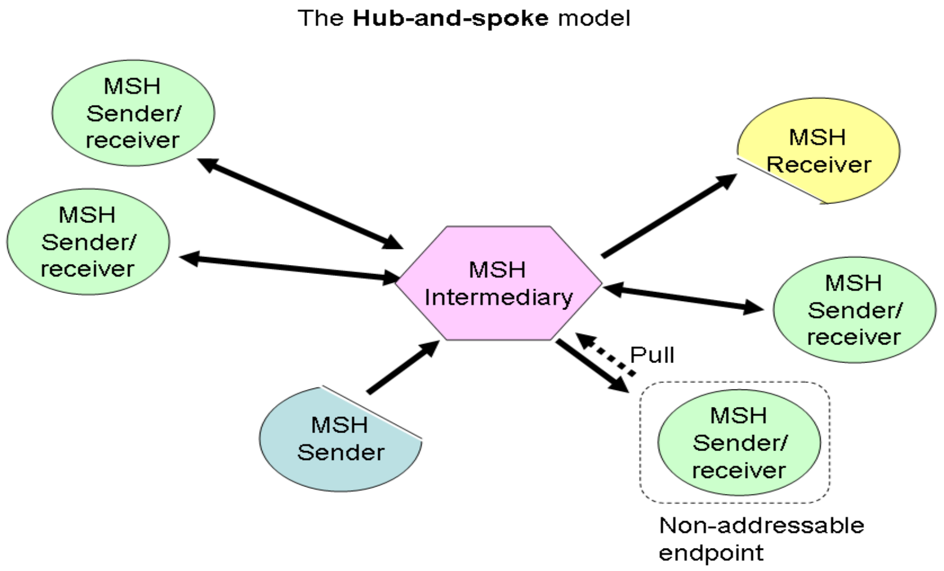

Figure 2: Hub -and-Spoke Topology

2.3.3 Interconnected Hubs

This topology is a generalization of the Hub-and-Spoke model. It applies when each Hub is only serving “regional” Endpoint MSHs, e.g. for security, manageability or scalability reasons. The group of endpoints directly served by the same Hub is called here an Endpoint cluster. Each Hub can be configured for routing messages intended to an Endpoint MSH of another cluster.

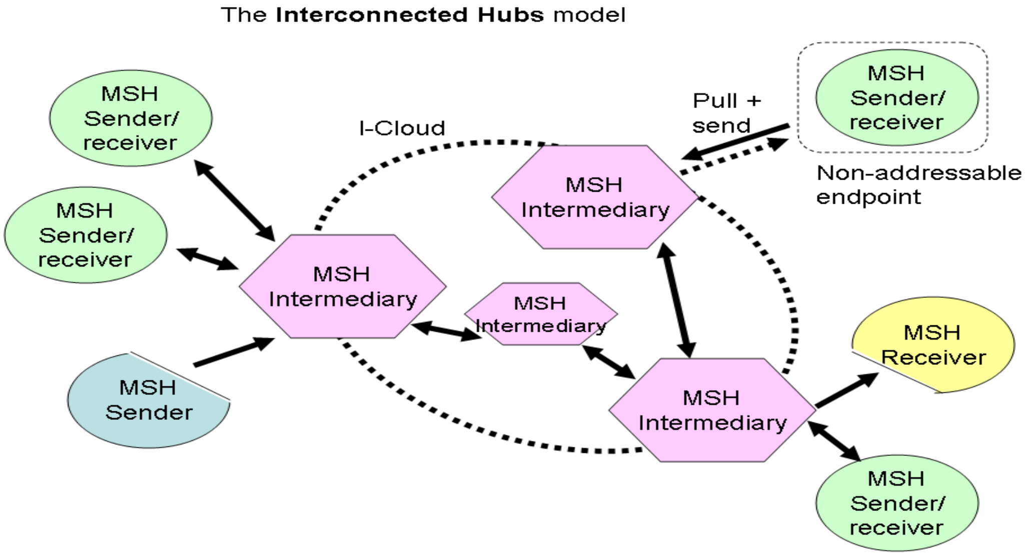

Figure 3: Interconnected Hubs Topology

Some Intermediaries may not serve any cluster of endpoint MSHs, but only act as relays between Intermediaries.

A special case of the interconnected hubs topology is the Bridge Domains topology. In this topology some interconnected ebMS MSHs are acting as gateways to sub-networks. Each sub-network is only reachable from the outside via its Gateway and can use internal addresses and DNS names that are not publicly reachable or resolvable outside the sub-network.

The assumption is that every Gateway is reachable from any other Gateway and knows how to route messages intended to other domains. This topology mostly departs from the Interconnected Hub topology in its constraints about the addressing function and its partitioning into domains bridged by these Gateways.

2.4 Usage Requirements

2.4.1 Operation Assumptions

The following assumptions are underlying to the operation of ebMS multi-hop messaging as specified in this document:

- The multi-hop mode considered here is of **transparent** multi-hop. Transparent multi-hop is defined as involving only ebMS Intermediaries that do NOT modify in any way the SOAP messages they are forwarding. Other multi-hop modes are out of scope of this specification, although this specification does not preclude them.

- An ebMS Intermediary is able to support pulling (i.e. to process a received `eb3:PullRequest`) at least over Edge-hops, i.e. from Endpoint MSHs.

- A multi-hop path is decomposed into (a) edge-hops, and (b) I-Cloud hops (see section 1.1 for definitions), and a different configuration construct is controlling message transfer over these two types of hops:

    1. The P-Mode that is deployed on each endpoint. This P-Mode controls the communication over edge-hops (origin Endpoint to I-Cloud, or I-Cloud to destination Endpoint). These P-Modes only control what intermediary is in contact with the endpoint MSHs, and has no bearing on how these intermediaries transfer messages over the I-Cloud.

    2. The routing function for transfer inside the I-Cloud (ICloud hops). The Endpoint MSHs never have to be aware of the way messages are transferred in the I-Cloud, nor can they control it besides setting header content used as input by the routing functions.

2.4.2 Connectivity and Addressability constraints

An Endpoint MSH may or may not be addressable. Addressability is defined here as readiness to accept incoming requests on the underlying transport protocol – e.g. to be on the receiving side of a One-way / Push MEP. This often implies a static IP address, appropriate firewall configuration as well as general availability of the endpoint (no extended downtime).

If not addressable, an Endpoint MSH will pull messages from the Intermediary it is connected to in the multi-hop topology (i.e. must be able to act as the initiator of a One-way / Pull MEP, and the Intermediary to act as the responding MSH of such an MEP).

There may be other reasons for message pulling in addition to non-addressability, e.g. intermittent connectivity of endpoints, security aspects, and risk mitigation in reducing the time between message reception and message processing.

2.4.3 QoS of Exchanges

When using reliable and secure multi-hop exchanges, the following requirements hold:

- It must be possible to configure a multi-hop topology so that end-to-end message transfer is possible without breaking signatures. This implies that Intermediaries do not modify ebMS messages – as well as any message involved in an ebMS MEP over multi-hop (transparent multihop).

- It must be possible to configure a multi-hop topology so that end-to-end reliable transfer of a message is possible, i.e. over a single reliable messaging sequence.
- It must be possible to configure a multi-hop topology so that end-to-end secure conversations may be established, based on WS-SecureConversation [WSSC13].

These requirements are consistent with current drafts of the WS-I Reliable Secure Profile [WSIRSP10], which specifies how to combine reliable messaging and security in an interoperable way.

When message forwarding does not involve pulling, and when there is no other connectivity impediment, an Intermediary may either use streaming to forward a message without persisting any part of it, or store the message for subsequent forwarding, see section 2.5.2 on forwarding models. An intermediary that is capable of both streaming and store-and-forward messaging may select one or the other option using its routing function.

2.4.4 Intermediary Configuration and Change management

As in point-to-point communication, P-Modes governing message exchanges should only be known from Endpoint MSHs, and some subset of P-Modes features may need to be configured on the Intermediary that participates in the edge-hop. For example when an Intermediary has to support message pulling, it must have knowledge of authorization data related to each pulling Endpoint. This requires partial knowledge of P-Modes associated with message pulling.

Multi-hop exchanges between two Endpoint MSHs may be re-routed without knowledge from the Endpoints. In particular, messages sent over a single end-to-end reliable sequence may be routed on different paths, provided they reach the same destination. This may happen when an Intermediary is out of order, requiring routing via an alternate path.

2.4.5 Compliance with the SOAP Processing Model

In the SOAP processing model for intermediaries, any header that is understood and processed may be reinserted in the message before forwarding. A primary requirement for ebMS intermediaries is end-to-end security, meaning that they MUST NOT break the signature of messages they forward. This requirement, combined with the requirement of [WSIRSP10] that reliability headers and WS-Addressing headers MUST be included in the signature, whenever WS-Security is used, implies that intermediaries MUST NOT not alter in any way the ebMS 3.0 header and related headers (`eb3:Messaging`, `wsse:Security`, reliability headers and WS-Addressing headers), when forwarding a signed ebMS 3.0 SOAP message.

As defined in the SOAP specification [SOAP12], each SOAP node acts in a role. This role determines which SOAP headers may be processed by that node. For ebMS intermediaries a new role *nextMSH* is defined, identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/nextmsh`. ebMS intermediaries MUST act in this new role.

The sending endpoint SHOULD target the header required for routing the ebMS message to the destination, i.e. the `eb3:Messaging` or `ebint:RoutingInput` header (which will be introduced in section 2.5.5), to the *nextMSH* role and set the `mustUnderstand` attribute to true. In situations where both an `ebint:RoutingInput` element and an `eb3:Messaging` element are present, ONLY the `ebint:RoutingInput` element SHALL be targeted to the *nextMSH* as that is used for routing (see section 2.5.5).

Because ebMS intermediaries do not change the ebMS messages they route, the `eb3:Messaging` header can be targeted to the nextMSH when received by the ultimate endpoint. Therefore, the ultimate receiving endpoint SHOULD also act in the nextMSH role. If the ultimate receiving endpoint is unable to act in the nextMSH role, the sending endpoint SHOULD target `eb3:Messaging` headers to the default ultimateReceiver role, and SHOULD make use of the WS-Addressing reference parameter `ebint:RoutingInput` targeted to the *nextMSH* role for the routing information. Whether the `eb3:Messaging` header should be targeted to *nextMSH* is a matter of agreement between endpoints. This can be controlled using a new parameter **Pmode[1].Protocol.AddActorOrRoleAttribute**, described in section 12.1

The WS-Addressing `wsa:To` header which targets the I-Cloud is also to be processed by ebMS intermediaries and therefore MUST also target this role.

2.4.6 MEPs and Channel Bindings

Section 2.2 of the Core V3 Specification defines the notion of ebMS message exchange patterns. These MEPs represent how ebMS User Messages are exchanged between two MSHs. The Core Specification defines two MEPs:

- One-Way for the exchange of one message and
- Two-Way for the exchange of a request message followed by a reply in response.

Also defined in section 2.2.3 of the Core Specification is the concept of MEP Bindings. Such a MEP binding defines how the abstract MEP is bound to the underlying transport layer / protocol.

Although the Core Specification restricts the definition of MEP to the exchanges between two MSHs, the above MEPs are actually independent from the network topology as the MEPs represent the exchange pattern between the application-level Producer and Consumer of the message. Therefore two partners evolving from a point-to-point topology toward a multi-hop topology would still use the same message exchanges patterns (One-Way, Two-Way) as defined in the Core specification (V3). The way these MEPs bind to the underlying transport protocol does change however, as the transfer is now divided into multiple hops. This implies that the binding of MEPs to the underlying transport may vary in a way that is not covered by the Core Specification.

Message transfer over a multi-hop path including the way the underlying transport protocol is used, is controlled by two different means depending on the type of hop:

- For the **edge hops** the transfer is primarily controlled by the PMode deployed on the endpoint MSHs;
- Transfers within the I-Cloud, i.e. on the **I-Cloud hops** are controlled by the routing function deployed on each Intermediary. The routing function of intermediaries is defined in section 2.5

The following figure illustrates the control of multi-hop transfers and the related partitioning of multi-hop paths.

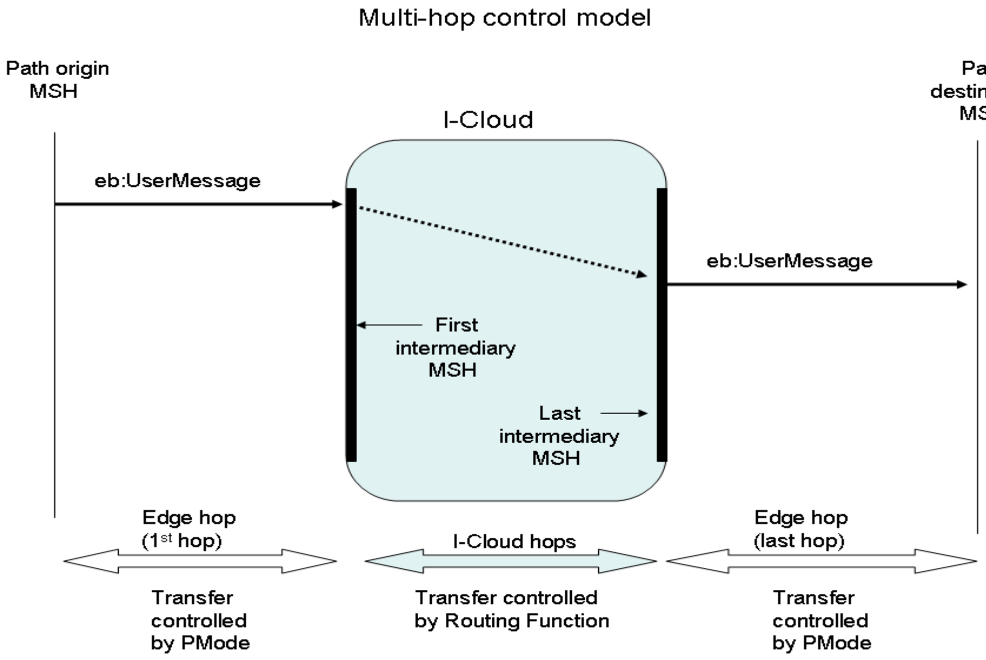

Figure 4: Multi-hop control model

Throughout this specification, the notion of “multi-hop MEP binding” will only be defined in terms of the binding of edge hops, and will make abstraction of the binding of I-Cloud hops.

As only the channel binding of the edge hops is controlled by the P-Mode, the P-Mode MEP Binding parameter only defines the binding of the edge hop (e.g. push or pull) between this endpoint and the first (or last) Intermediary of the I-Cloud. These MEP bindings are therefore called “edge-bindings”.

The following subsections describe the most common multi-hop MEP bindings from the endpoint perspective (edge-bindings). They remain independent from the channel binding of the multi-hop section that occurs inside the I-Cloud.

These multi-hop MEP bindings are entirely determined by the combination of point-to-point MEP bindings for both edge hops, as explained in sub-sections 2.4.7 and 2.4.8 . Therefore they will not be defined by specific URI values for the **PMode.MEPbinding** parameter. For ease of reference they will be given composed names like “First-and-Last-Push”.

While in a point-to-point context equivalent P-Modes are deployed by each endpoint to control the point-to-point exchange, slightly different P-Modes on each endpoint *may* be deployed for controlling the same exchange over a multi-hop path. Section 2.7 will describe these differences, one of them being the **PMode.MEPbinding** parameter value which may now differ on both ends, as the first and last hops (edge hops) may be channel-bound quite differently over a multi-hop path.

2.4.7 Edge-bindings for Multi-hop One-Way

NOTES:

1. In the following, the path origin MSH is also called the Sender, and the path destination MSH, the Receiver.
2. This section lists the edge-binding combinations that are expected to be most commonly used. They do not preclude other combinations.

2.4.7.1 Pushing Messages from the Sender Endpoint

Both of the following edge-binding combinations assume a Sender pushing the message to the I-Cloud. These edge bindings are configured via the **PMode.MEP** and **PMode.MEPbinding** parameters deployed on each endpoint, using conventional values defined in Core V3:

Case 1: "**First-and-last-push**". Both endpoint MSHs interact with the I-Cloud using the same MEP bindings they would use with their partner in a direct point-to-point mode.

P-Mode MEP configuration:

- Edge hop Sender side:
  - Sender as PMode.Initiator
  - MEP and binding = One-way / Push
- Edge hop Receiver side
  - Receiver as PMode.Responder
  - MEP and binding = One-way / Push

Case 2: "**First-push-last-pull**". This edge-binding will be used when both Sender and Receiver endpoints are not addressable, or when these Endpoints are not willing to receive incoming requests.

P-Mode MEP configuration:

- Edge hop Sender side:
  - Sender as PMode.Initiator (Intermediary as PMode.Responder)
  - MEP and binding = One-way / Push
- Edge hop Receiver side
  - Receiver as PMode.Initiator (Intermediary as PMode.Responder)
  - MEP and binding = One-way / Pull

2.4.7.2 Pulling Messages from the Sender Endpoint

Both of the following edge-binding combinations assume that the I-Cloud pulls messages from the Sender. These edge bindings are configured via the PMode.MEP and PMode.MEPbinding parameters deployed on each endpoint, using conventional values defined in Core V3:

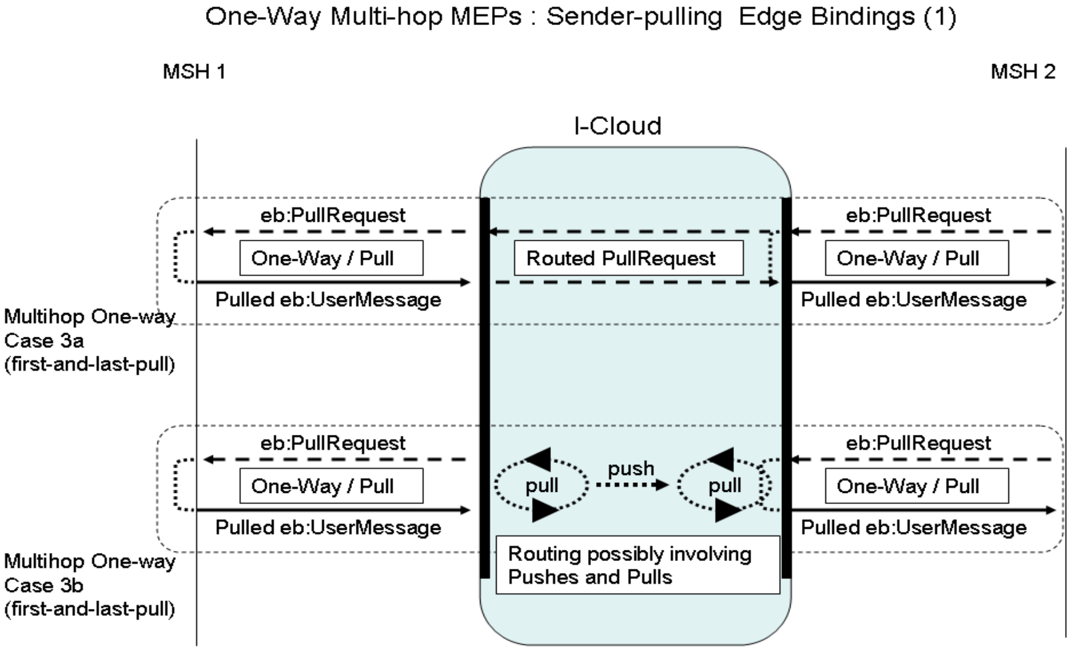

Figure 5: One-Way Multi-hop MEPs: Sender-Pulling Edge Bindings (1)

In both Case 3a and 3b above, the edge bindings are quite similar: all of them are One-way / Pull. The difference lies in the way the pulling is propagated across the I-Cloud, which may affect endpoints even if this is still the same edge-binding, named here "first-and-last-pull"..

Case 3a: In this MEP binding, the `eb3:PullRequest` signal is generated by the Receiver endpoint MSH, and routed all the way by the I-Cloud as any other message, to the Sender MSH. In other words, the same `eb3:PullRequest` message is used in both edge-hops. The main advantage is that there is a single authorization point for the pulling: the Sender MSH to which the Pull signal is intended. The I-Cloud has no responsibility in authorizing the pulling and has no authorization information (passwords or certificates) to maintain. On the other hand, the implication of such multi-hop pulling, is that each Intermediary on the path must keep its transport connections open.

P-Mode MEP configuration:

- Edge hop Sender side:
  - Sender as PMode.Responder (Intermediary as PMode.Initiator)
  - MEP and binding = One-way / Pull
- Edge hop Receiver side
  - Receiver as PMode.Initiator (Intermediary as PMode.Responder)
  - MEP and binding = One-way / Pull

Case 3b: In this MEP binding, the `eb3:PullRequest` signal is not routed, and only goes over a single hop. The `eb3:PullRequest` signals over each edge-hop are different messages, which could have different authorization credentials (the Pull signal is authorized for the next node only). These edge pulls are relayed by the I-Cloud in unspecified ways – the pulled message could be pushed and/or pulled across the I-Cloud.

P-Mode MEP configuration: same as for Case 3.

The difference between Case 3a and Case 3b affects endpoint behavior in the way reliable messaging is supported. In Case 3b the `eb3:PullRequest` from the Receiver is only sent to the Intermediary and not routed to the Sender endpoint, so there is no use for sending the `eb3:PullRequest` reliably to Sender as would be the normal way when operating in a point-to-point context (see Core specification). See section 4.2 " Details of Reliable and Secure Multi-hop" for a detailed description and a new P-Mode parameter **Pmode[1].Reliability.AtleastOnce.ReliablePull**

Another less common one-way edge-binding is illustrated below in Case 4:

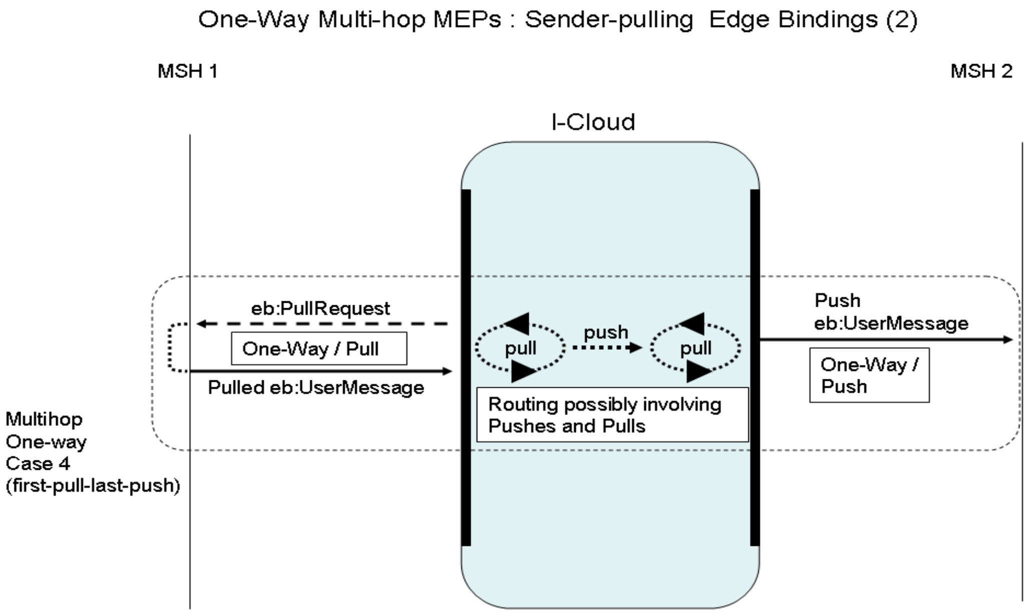

Figure 6: One-Way Multi-hop MEPs: Sender-Pulling Edge Bindings (2)

Case 4: "**First-pull-last-push**". In this MEP binding, the first edge-hop is pulled, while the last edge-hop is pushed. These edge pulls are relayed by the I-Cloud in unspecified ways – the pulled message could be pushed and/or pulled across the I-Cloud. The difference between Case 4 and Case 3 is in the last edge-hop binding.

P-Mode MEP configuration:

- Edge hop Sender side:
  - Sender as PMode.Responder (Intermediary as PMode.Initiator)
  - MEP and binding = One-way / Pull
- Edge hop Receiver side
  - Receiver as PMode.Responder (Intermediary as PMode.Initiator)
  - MEP and binding = One-way / Push

Other combinations of edge-bindings are expected to be used and supported by Intermediaries that are not described here although they will automatically be supported by Intermediaries that already support the edge-binding combinations specified here.

2.4.8 Edge-bindings for Multi-hop Two-Way

NOTE:
In the following, two multi-hop paths are involved: one for the “request” message, one for the “reply” message. The origin MSH for a path is also called the Sender, and the destination MSH, the Receiver.

2.4.8.1 Asynchronous Edge-bindings

Both of the following edge-binding combinations assume a reply message that is sent back asynchronously by the request Receiver endpoint MSH. The routing through the I-Cloud is independent from these edge-bindings. The edge bindings are configured via the PMode.MEP and PMode.MEPbinding parameters deployed on each endpoint, using conventional values defined in Core V3:

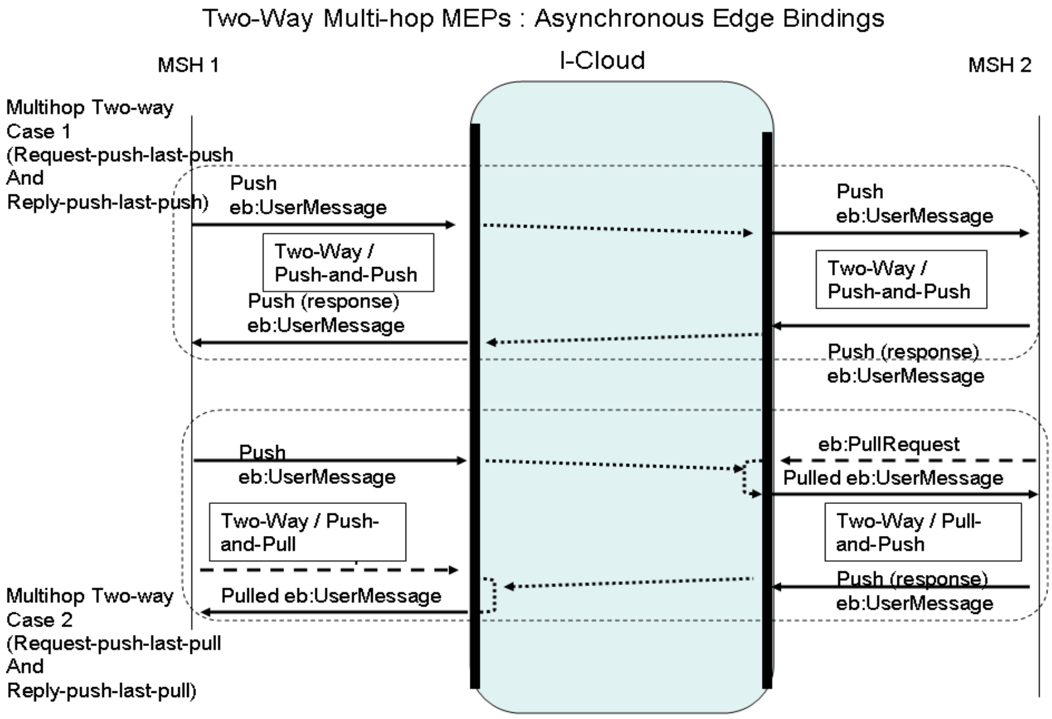
Two-Way Multi-hop MEPs : Asynchronous Edge Bindings
MSH 1
I-Cloud
MSH 2
Multihop Two-way Case 1 (Request-push-last-push And Reply-push-last-push)
Push eb:UserMessage
Two-Way / Push-and-Push
Push (response) eb:UserMessage
Push eb:UserMessage
Two-Way / Push-and-Push
Push (response) eb:UserMessage
Push eb:UserMessage
Two-Way / Push-and-Pull
Pulled eb:UserMessage
eb:PullRequest
Pulled eb:UserMessage
Two-Way / Pull-and-Push
Push (response) eb:UserMessage
Multihop Two-way Case 2 (Request-push-last-pull And Reply-push-last-pull)

Figure 7: Two-Way Multi-hop MEPs: Asynchronous Edge Bindings

Case 1: "**Request-push-last-push and Reply-push-last-push**". Both endpoint MSHs interact with the I-Cloud using the same MEP bindings they would use with their partner in a direct point-to-point mode. Both are addressable.

P-Mode MEP configuration:

- Edge hops on Request Sender side:
  - Request Sender as PMode.Initiator (Intermediary as PMode.Responder)
  - MEP and binding = Two-way / Push-and-Push.
- Edge hops on Request Receiver side:
  - Request Receiver as PMode.Responder (Intermediary as PMode.Initiator)
  - MEP and binding = Two-way / Push-and-Push

Case 2: "**Request-push-last-pull and Reply-push-last-pull**". This case applies when both endpoint MSHs are not addressable: both are pulling messages from the I-Cloud.

P-Mode MEP configuration:

- Edge hops on Request Sender side:
  - Request Sender as PMode.Initiator (Intermediary as PMode.Responder)
  - MEP and binding = Two-way / Push-and-Pull
- Edge hops on Request Receiver side:
  - Request Receiver as PMode.Responder (Intermediary as PMode.Initiator)
  - MEP and binding = Two-way / Pull-and-Push

Other combinations of asynchronous edge-bindings are expected to be used and supported by Intermediaries that are not described here although they will automatically be supported by Intermediaries that already support the edge-binding combinations specified here. An example would combine Case 2 and 3 above: { first edge-binding = Two-way / Push-and-Push, last edge-binding = Two-way / Pull-and-Push}.

2.4.8.2 Synchronous Edge-bindings

Both of the following edge-binding combinations assume a reply message that is sent back synchronously by the request Receiver endpoint MSH. The routing through the I-Cloud is independent from these edge-bindings. These edge bindings are configured via the PMode.MEP and PMode.MEPbinding parameters deployed on each endpoint, using conventional values defined in Core V3:

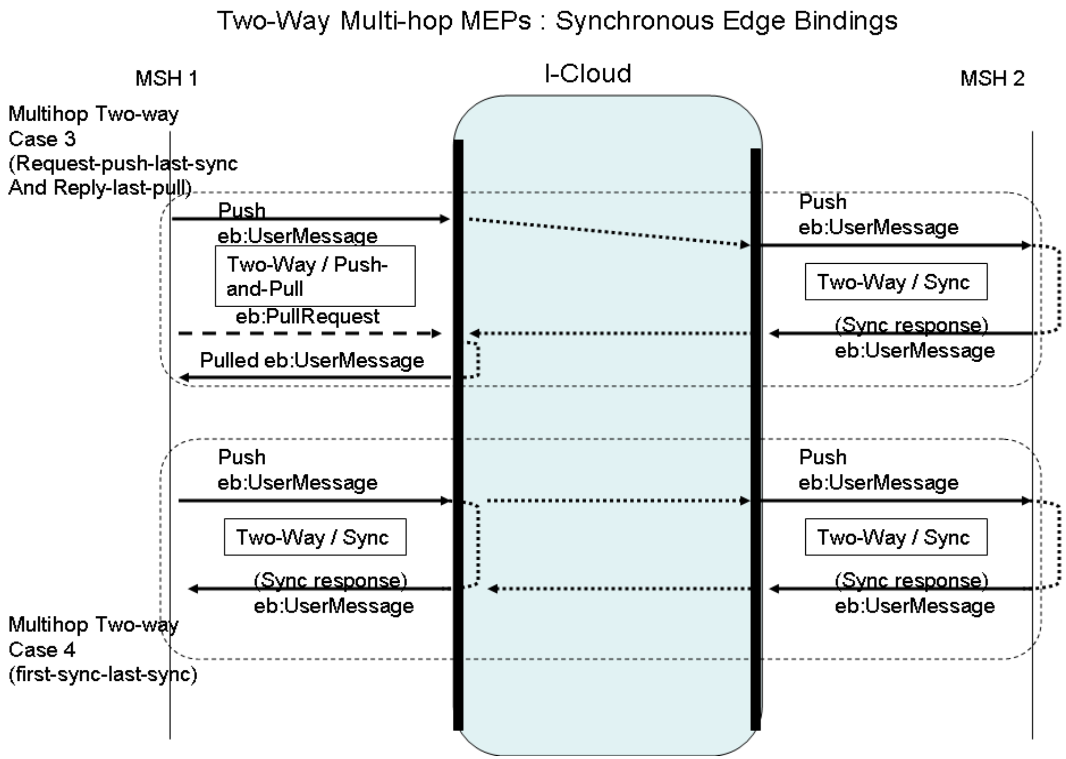

Figure 8: Two-Way Multi-hop MEPs: Synchronous Edge Bindings

Case 3: "**Request-push-last-sync and Reply-last-pull**". The request Receiver endpoint MSH interacts with the I-Cloud using the same MEP binding it would use with its partner in a direct point-to-point mode. The request Sender MSH may be non-addressable, and will pull the reply message. No connection has to be kept open on the first edge-hop.

P-Mode MEP configuration:

- Edge hops on Request Sender side:
    - Request Sender as PMode.Initiator (Intermediary as PMode.Responder)
    - MEP and binding = Two-way / Push-and-Pull
- Edge hops on Request Receiver side:
    - Request Receiver as PMode.Responder (Intermediary as PMode.Initiator)
    - MEP and binding = Two-way / Sync

Case 4: "**First-sync-last-sync".** Both endpoint MSHs interact with the I-Cloud using the same MEP binding they would use between themselves in a direct point-to-point mode. The request Sender MSH may be non-addressable, and will get the reply message over the same transport connection as the request message – so this connection has to be kept open on the first edge-hop.

P-Mode MEP configuration:

- Edge hops on Request Sender side:
    - Request Sender as PMode.Initiator (Intermediary as PMode.Responder)
    - MEP and binding = Two-way / Sync
- Edge hops on Request Receiver side:
    - Request Receiver as PMode.Responder (Intermediary as PMode.Initiator)
    - MEP and binding = Two-way / Sync

NOTE:

Although both Endpoints in this case use synchronous communication with their respective intermediaries there is no guarantee that the response message is communicated synchronously end-to-end as the I-Cloud hops might use asynchronous communication. When an I-Cloud uses synchronous communication on the I-Cloud hops to enable end-to-end synchronous communication this may be very resource intensive on the intermediaries because of all open connections.

2.5 The Intermediary MSH

2.5.1 Intermediary Functions

An ebMS intermediary is expected to support the following functions:

Message forwarding. Besides the Send and Receive operations defined in the model of V3 Core specification, a new operation called Forward is added to the messaging model. It is defined as: sending (operation Send) a message that has been received (operation Receive) without altering it, and without external intervention (i.e. without the message being delivered in-between, then re-submitted). Doing so requires no additional processing other than that needed by the routing function.

The following figures illustrate a Hub Intermediary MSH (Hub-and-Spoke topology) forwarding a message either for pushing to or for pulling by the Receiver endpoint.

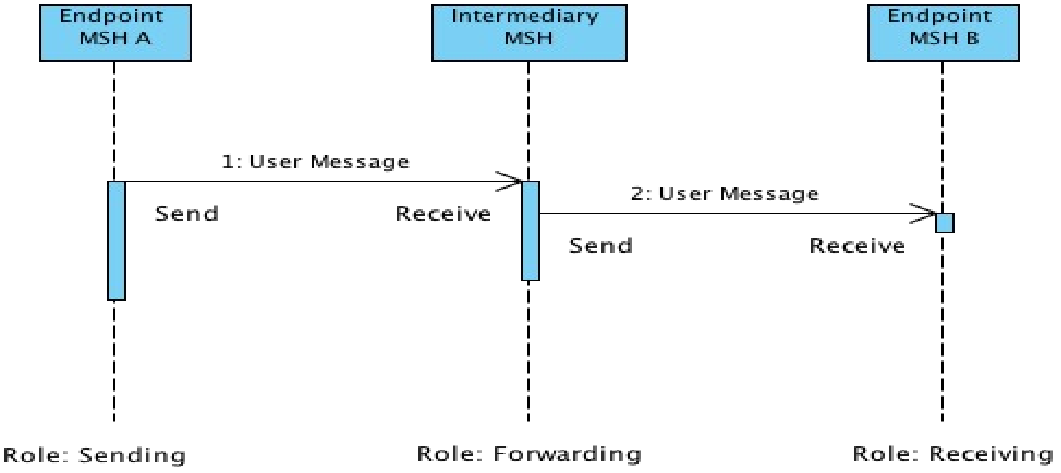

Figure 9: Message forwarding

NOTE: The diagram could easily be generalized for the Interconnected Hubs topology, where every Intermediary on the multi-hop path would have to act in the Forwarding role.

Message Routing. As an intermediary is not an endpoint of the multi-hop path, every message received MUST be forwarded to another MSH. The routing function of an intermediary defines to which MSH a message must be forwarded based on metadata carried by the received message.

MEP bridging. Forwarding MAY also imply the ability to bridge between different MEP channel bindings. For example, a message bound to a One-way / Push MEP when received, may be bound to a One-way / Pull after forwarding, as illustrated in the following figure.

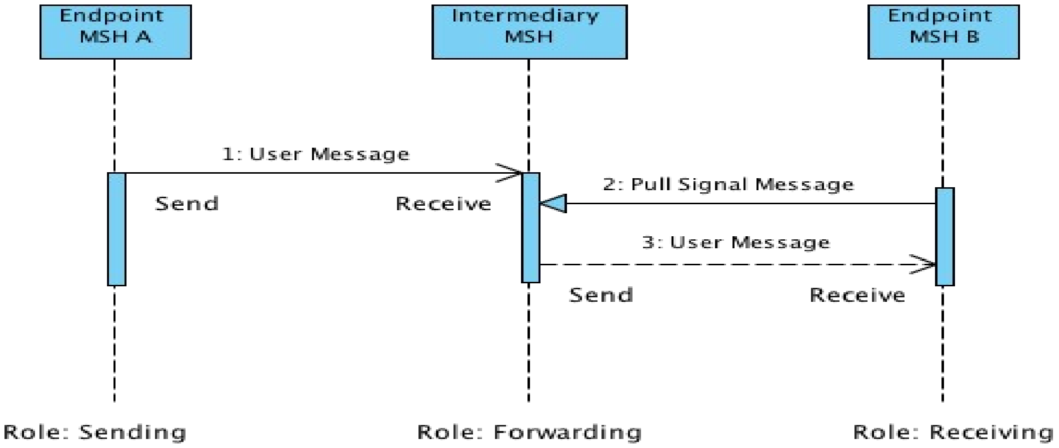

Figure 10: MEP Bridging

Message Pulling support. This implies some partial knowledge of PModes that determine a push vs. a pull channel. In the case of pulling, a scalability feature like sub-channels (see section 2.5.4) needs to be supported and configured with authorization info. In addition to assigning `eb3:UserMessage`s to Pull channels, an Intermediary MUST also be able to assign `eb3:SignalMessage`s and non-ebMS, e.g. RM signals such as `wsrm:CreateSequence` or sequence acknowledgments, to a Pull channel based on the `@mpc` attribute on the routing parameter in the SOAP header so that these signals become available for pulling by the endpoint MSH. This represents an extension to the pull protocol as defined in [EBMS3CORE]: in addition to ebMS user messages and ebMS signal messages containing an `eb3:Error` element with code: EBMS:0006 (EmptyMessagePartitionChannel), a third valid response message type is: any SOAP message containing a valid `ebint:RoutingInput/ebint:UserMessage` header element.

Error Handling. An Intermediary MUST be able to generate `eb3:Error`s: as discussed in section 2.5.6 .

2.5.2 Message Forwarding Models

For the new abstract operation Forward two implementation models exist. First is the store-and-forward model in which the Send operation only starts after the Receive operation has successfully been completed. So the Send and Receive are executed sequentially and the successful execution of the Receive operation is not dependent on the successful execution of the Send operation. The sequence diagram for this model is shown in the next figure.

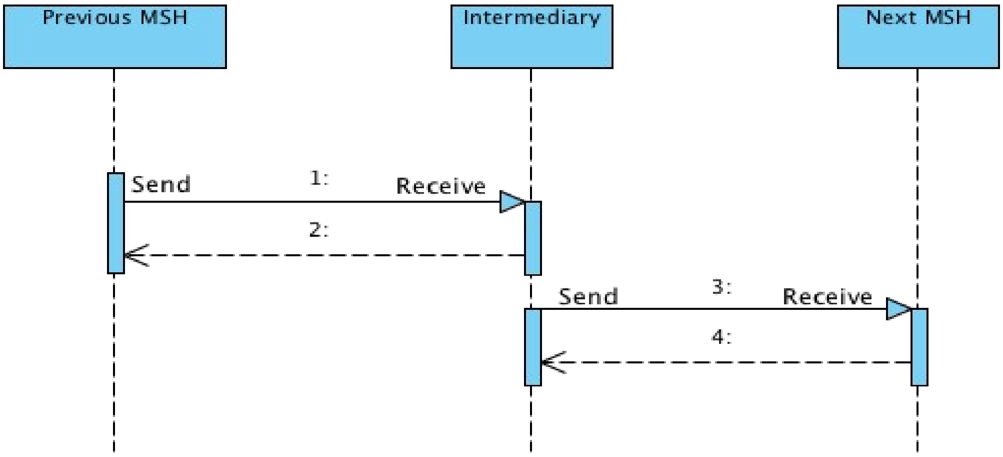

Figure 11: Store-and-forward model

An alternative forwarding model is the streaming model in which both operations are executed in parallel, i.e. data is directly transferred to the next hop. In this model the Send operation starts as soon as possible after the Receive operation has started. Because the next hop will only be known after the message header has been received the Send operation will always start some time after the start of the Receive operation.

Depending on whether the Receive operation ends before or at the same time as the Send operation two sub cases of the streaming model exist. In the first sub case, called asynchronous streaming, the Receive operation completes successfully after receiving all message data, i.e. a transport channel acknowledgment is sent to the previous hop and the channel is closed. The figure below shows the sequence diagram for this streaming model sub case.

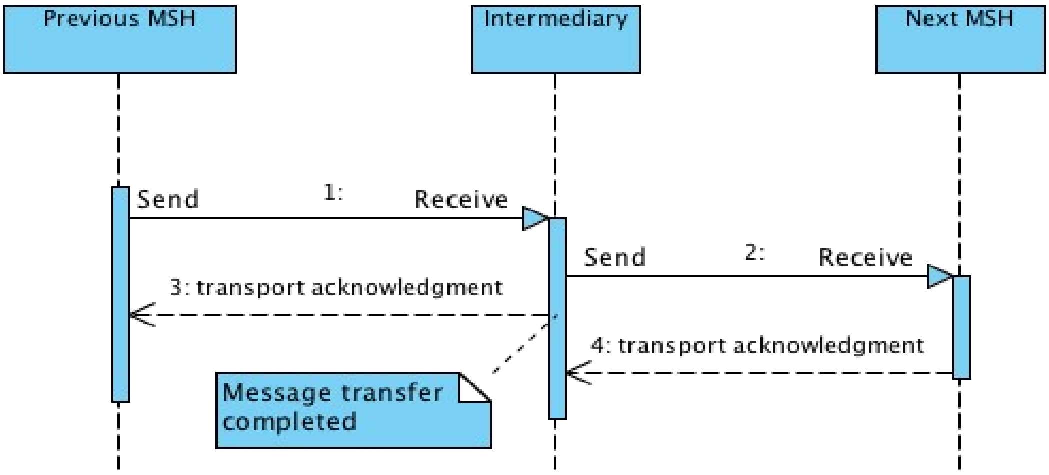
Previous MSH
Intermediary
Next MSH
Send
1:
Receive
Send
2:
Receive
3: transport acknowledgment
4: transport acknowledgment
Message transfer completed

Figure 12: Asynchronous Streaming

In the other case, called synchronous streaming, the Receive operation only ends when the Send operation is complete successfully, i.e. the transport acknowledgment is only sent back to the previous hop when a transport acknowledgment is received from the next hop. The sequence diagram for this sub case of the streaming model is shown in the next figure.

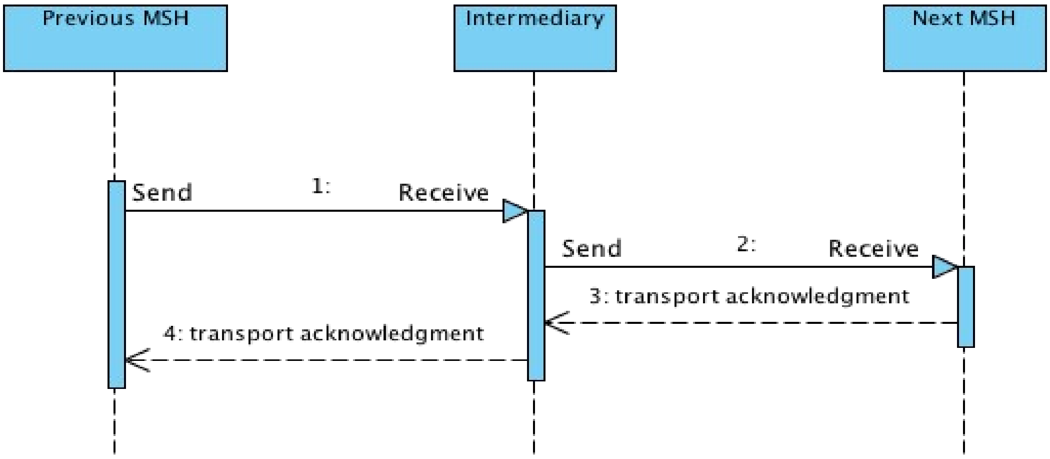

Figure 13: Synchronous streaming

Note that synchronous refers to the transport acknowledgments and not the ebMS message transfer. Even when using synchronous streaming intermediaries the ebMS MEP could be asynchronous. For synchronous ebMS message transfers, the only option is to use the synchronous streaming model because in this model the intermediary MSH will wait with responding to the previous hop until a response is received from the next hop.

An advantage of both streaming models over the store-and-forward model is that they require less storage space because there is no need to store complete messages. Also the end-to-end latency is reduced.

The main disadvantage of both streaming models is that they require more bandwidth as the Receive and Send operation run in parallel. Especially in the synchronous model where the incoming connection stays open until the outgoing one is closed, this will lead to congestion on the intermediary if the outgoing connection does not have at least the same bandwidth as the incoming connection. Therefore the [synchronous] streaming model SHOULD only be used when bandwidth is known to be available.

2.5.3 MEP Bridging

The forwarding function may involve message pushing as well as message pulling. Message pulling at Intermediary level is controlled by the routing function. More precisely, message forwarding falls into one of the following patterns:

(a) **"Push-on-push" :** Messages pushed to the Intermediary over MPC x are forwarded in push mode on MPC x to the next node;

(b) **"Pull-on-push" :** Messages pushed to the Intermediary over MPC x, are forwarded in pull mode, on MPC x or a sub-channel of x (see section 2.5.4 on sub-channels) to the next node. I.e. the message will be pulled from MPC x in the Intermediary by the next node MSH which can either be an Intermediary or an endpoint;

(c) **"Push-on-pull" :** Messages are pulled by the Intermediary from MPC x or a sub-channel of x and forwarded to the next MSH in push mode on MPC x. I.e. the sub-channel used to pull the message from is not forwarded to the next MSH;

(d) **"Pull-on-pull" :** Messages are pulled by the Intermediary from MPC x or a sub-channel of x and forwarded in pull mode on MPC x or a sub-channel of x (see next section on sub-channels) to the next node. If used the sub-channel the message if forwarded on may differ from the sub-channel the messages is pulled from.

The routing function MUST specify which forwarding pattern is in use for each MPC it handles. It also includes authorization credentials associated with pulling – either inbound or outbound – when some forwarding pattern involves pulling.
This specification does not define which ones of these forwarding patterns must be supported by an intermediary: this is to be determined by future conformance profiles.

2.5.4 Sub-channels

The nature of message partition channels (MPC) as defined in the core ebMS V3 specification is such that an intermediary may forward the flow of user messages with the same MPC value (i.e. having the same `eb3:Messaging/eb3:UserMessage/@mpc` value) to different destinations. This requires the routing function to use other criteria than [only] the `@mpc` value. Indeed, an MPC is not associated with a particular pair of Sending / Receiving MSHs.

However this forwarding of messages from the same MPC to multiple destinations requires an additional capability when these destinations are pulling these messages, i.e. when the last intermediary is using either "pull-on-push or "pull-on-pull" forwarding patterns.

An example of such a use case is:

A party is sending messages to a large number of recipients over the I-Cloud. These recipients are supposed to pull these messages from their common edge-intermediary. Because each recipient is not supposed to pull messages for other recipients', each recipient will pull from a different MPC that is authorized differently from others. However, the sending party does not want to be aware of all the MPCs that are associated with each recipient, and is sending all messages over the same MPC. The last intermediary alone is aware of which message should be forwarded to which recipient.

Sub-channels are a means to support the previous use case. A sub-channel is associated with an existing or "parent" MPC, and plays the role of a new MPC in case of a "pull" forwarding. Every ebMS User Message is always assigned to an MPC - if only to the default one (see also chapter 3 of the Core specification on message pulling). An Intermediary may decide to "sub-channel" this MPC, which means it associates one or more sub-channels to this MPC, and assigns the received message to one of these sub-channels when forwarding, based on message metadata. Now the different endpoint MSHs can pull their messages from the specific sub-channel assigned to them instead of the general parent channel.

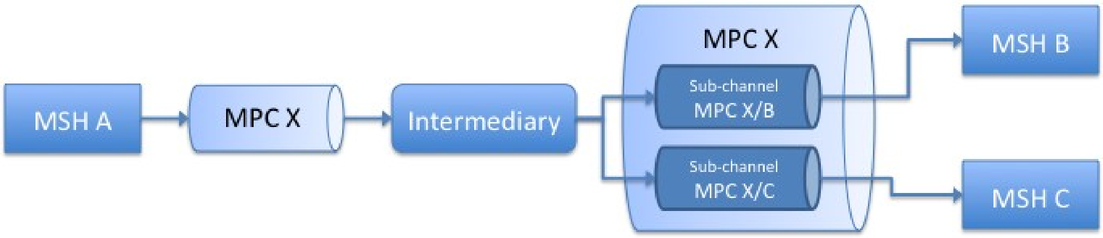

Figure 14: MPC sub-channels

In order to pull from a particular sub-channel, an MSH must use the sub-channel ID in the `@mpc` attribute of the `eb3:PullRequest`. This pulling may be authorized differently than pulling from the parent MPC, i.e. the `eb3:PullRequest` must contain authorization credentials that are specific to this sub-channel. The same message can also by pulled directly from its parent MPC, if the `eb3:PullRequest` contains the right authorization credentials (which may be different from those associated with its sub-channels). However a message assigned to sub-channel SC1 cannot be pulled from sub-channel SC2, even if SC1 and SC2 have same parent channel. Typically, when sub-channels are associated with different recipients who need to ensure that their messages are not pulled by others, each sub-channel needs to be authorized differently.

The identifier of a sub-channel is an extension of the identifier of the parent MPC. MPCs are identified with URIs. Different ways to indicate hierarchy in URIs exist, which can be used to indicate a channel / sub-channel relation. When an MPC (y) is a sub-channel of another MPC (x), the following rules apply:

- the URI scheme in (x) and (y) must be same (and the URI authority if used in one must also be used in the other with same value)

- the path of the URI of the sub-channel (y) must start with the path of of the URI of the parent channel (x).

Example: If the MPC identifier is an URI of the form:

```
http://sender.example.com/mpc123
```

A sub-channel of this MPC may have an identifier of the form:

```
http://sender.example.com/mpc123/subc42
```

Because intermediaries should not modify forwarded messages, the @mpc attribute value in the message is not altered so the message is still considered as sent over this MPC, the sub-channel assignment is purely 'virtual'. This also implies that sub-channel assignment can only take place in the forwarding patterns (b) (pull-on-push) and (c) (pull-on-pull). Intermediaries SHALL only forward a message over the MPC specified in the message or a sub-channel thereof. This behavior is determined by the routing function associated with the intermediary.

From the receiver viewpoint (in the above use case, the ultimate recipient), sub-channels are used in the same way as normal MPCs: sub-channel identifiers are to be used in P-Modes in the same way as other MPC identifiers. An `eb3:PullRequest` message for a sub-channel will specify the sub-channel ID in its `@mpc` value. Because the sub-channel assignment is only virtual, the pulled message however will still have the parent MPC identifier in its `@mpc` value. Therefore a receiving MSH SHOULD accept messages from a parent MPC when pulling.

2.5.5 The Routing Function

A function every intermediary MUST implement is the routing function that defines to which MSH a received message is to be forwarded. The input to this routing function, called the **routing input**, is a set of metadata elements taken from the received message. As the main purpose of an ebMS Intermediary is to route ebMS User Messages the metadata used for routing is based on information elements available in such messages. Non-normative Appendix A provides some use cases and good practices for routing in a multi-hop context.

The routing function can be modeled as:

f(«RoutingInput») → «next destination»

The output of the routing function is the next destination of the message. This is not just the URL of the next MSH, but also includes information on how the forwarding should be done - i.e. by pushing or pulling. When for example the destination is the ultimate receiver which uses pulling to get its messages there is no URL where the message must be sent to. The routing function must then specify which MPC is intended for pulling and specify the credentials for pulling authorization.

The input for the routing function, the *RoutingInput* is a subset of the header content of ebMS User Messages, more precisely the `eb3:UserMessage` element. The routing function of an Intermediary MAY use all information available in the `eb3:UserMessage` element or it descendants. However it MUST be able to parse and use the following metadata as its routing input:

- The `mpc` attribute, when present.
- The `eb3:PartyInfo` element and its sub-elements.
- The `eb3:CollaborationInfo` element and its sub-elements

Furthermore the routing function SHOULD be able to parse and use:

- The `eb3:MessageProperties` element and its sub-elements,
- The `eb3:PayloadInfo` element and its sub-elements

The routing function must be able to route all messages that need to be forwarded by the intermediary, including:

1. ebMS signal messages which are not also ebMS user messages;
2. non-ebMS messages that are involved in facilitating the transfer and quality of service of ebMS messages (such as various signals supporting reliable messaging).

Such messages however do not contain an `eb3:UserMessage` element with the input needed for the routing function. In order to provide the routing function with proper routing input these messages MUST contain a WS-Addressing endpoint reference parameter that includes the routing input – except for PullRequest signals when the routing function can determine the destination of the PullRequest based on the MPC alone.

This reference parameter is defined as an element named `ebint:RoutingInput` which contains exactly one child `ebint:UserMessage` element. For non ebMS user messages like 1. and 2. above the SOAP header must contain the following WS-A reference parameter:

```
<ebint:RoutingInput
  wsa:IsReferenceParameter='true'
  S12:role="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/nextmsh"
>
   <ebint:UserMessage>...</ebint:UserMessage>
</ebint:RoutingInput>
```

The embedded `ebint:UserMessage` element is similar to the XML schema defined for the similar element in the packaging section of Core V3 and reuses some of its type and element definitions, except for the `eb3:MessageInfo` element which can be absent here. The schema is defined in Appendix C Reference Parameter.

NOTE: The `UserMessage` element in the reference parameter is in a different namespace than the `UserMessage` element from the Core V3 specification because it allows for the `MessageInfo` element to be absent and therefore must be redefined for use in the reference parameter. An Intermediary MUST NOT fault a message with an `ebint:UserMessage` element that does not contain an `eb3:MessageInfo` element, and MUST consider such an `ebint:UserMessage` as a valid input for the routing function.

In the XML Schema definition for the ebMS 3.0 Core Specification, defined in section [EBMS3CORE] and available from http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd, the `eb3:MessageInfo`, `eb3:PartyInfo`, `eb3:CollaborationInfo`, `eb3:MessageProperties` and `eb3:PayloadInfo` elements are defined in the `UserMessage` complex type. XML schemas that import the ebMS 3.0 Core Specification cannot reference these elements. To support reuse of these elements within the `ebint:UserMessage` element, a refactored version of the ebMS 3.0 schema is provided (see Appendix B) where these elements are defined separately. The `ebint:RoutingInput` schema defined in Appendix C imports this refactored schema. Note that an `eb3:Messaging` rooted XML document is valid with respect to this refactored XML schema if and only if it is valid according to the original XML schema and that processing such a document using the refactored XML schema results in the same Post Schema Validation Infoset as from the original ebMS 3.0 XML schema.

An ebMS Intermediary MUST be able to obtain the routing input from a message (provided that information is targeted to it) in one of the following ways:

- By parsing the `eb3:Messaging` header in an ebMS user message, and by extracting its `eb3:UserMessage` child element.
- By parsing the `eb3:PullRequest` header element in a PullRequest message and using the value of its `@mpc` attribute;
- By parsing the WS-Addressing reference parameter header `ebint:RoutingInput` and by extracting its `ebint:UserMessage` child element

It is possible that an ebMS user or PullRequest message also contains a WS-Addressing reference parameter in the SOAP header. In such cases where multiple routing input options are present in the same message, the intermediary MUST use the WS-Addressing reference parameter as input for the routing function.

A routing function must be able to map the `ebint:RoutingInput` header either to a URL (for pushing the forwarded message) or to a pull channel.

A routing function will generally use only a subset of the `eb3:UserMessage` element or of the `ebint:RoutingInput` element. This subset is called the *effective* routing input, as opposed to the *available* routing input represented by the entire `eb3:UserMessage` element, `ebint:RoutingInput` element or pull request.

When the effective routing input is reduced for some messages to the sole MPC value – or in other words, when the I-Cloud routing function uses only the MPC value (`@mpc`) for some routings – then each one of these MPC values resolved by the routing function MUST NOT be shared by different destination endpoints. This is the case when `eb3PullRequest` messages that do not contain a WS-Addressing reference parameter must be routed through the I-Cloud as described in case 3a in section 2.4.7.2

2.5.6 Error Handling and Multi-Hop Messaging

Similarly to endpoints, intermediaries can generate ebMS errors. This section describes the reporting of errors by intermediaries and the types of errors that can be generated by them. The requirements on endpoints for routing Error response messages are described in section 2.6.2 .

The reporting of an error generated by an intermediary MUST follow one of these three patterns:

(a) <u>Fixed configured reporting</u>: errors are handled and reported in an implementation-specific manner: either logged locally or sent to a fixed, pre-configured destination, or yet assigned to a specific MPC that another MSH may pull from. In all cases this is subject to a QoS agreement between I-Cloud providers and endpoints connecting to their intermediary message handlers.

(b) Message-determined reporting: if there is no provision for solution (a), the error is sent to the destination based on data contained in the faulty message. This may in turn be subdivided into a couple of cases:

- Either the WS-Addressing header `wsa:FaultTo` or `wsa:ReplyTo` is present in the message in error and its value indicates an URL that can be directly resolved. If both headers are available the error MUST be sent to the URL given by the `wsa:FaultTo` header. The Error message MUST be sent directly to this location;
- One of these WS-Addressing headers is present and contains an `ebint:RoutingInput` reference parameter. The intermediary MUST then add this EPR to the error message (as described in 2.6.2 - the ebMS Error message being considered as a response message). The error message MUST then be sent to its destination through the I-Cloud. Again when both headers are available the EPR from the `wsa:FaultTo` header MUST be used;
- None of these WS-Addressing headers is present in the message in error, but an `ebint:RoutingInput` header can be inferred (section 2.6.2 case 4) from other headers available in the message in error. The intermediary MUST add the inferred `ebint:RoutingInput` header to the Error message and sent it using the I-Cloud.

(c) Synchronous reporting: in this case, the error related to the routing of a message that is sent to an intermediary as an HTTP request and generated upon the receipt of the message by this intermediary, is sent back as an error message over the HTTP response. This reporting mode is sufficient for hub-and-spoke configurations. It is of particular relevance with connected endpoints of small and medium size enterprises or other light clients that may only connect occasionally to the network to transmit messages to their trading partners. In case there are errors, the pattern ensures immediate feedback.

Of the errors specified in section 6.7.1 of [EBMS3CORE] at least the following errors SHOULD be supported:

- EBMS:0005 ConnectionFailure, in situations where the routing function of the intermediary is configured to push the message but a connection to a next hop cannot be established.
- EBMS:0009 InvalidHeader, in situations where the intermediary receives a message containing an invalid ebMS header.

Some of the errors specified in the core specification are not relevant in the context of Intermediaries. Therefore an intermediary:

- MUST NOT generate an EBMS0010, ProcessingModeMismatch, error, as intermediaries process messages based on a routing function and not based on Pmodes.
- SHOULD NOT generate EBMS0011, ExternalPayloadError, failures because, unlike the intermediaries involved, the ultimate recipient ebMS MSH MAY still be able to resolve such references even if the intermediaries are not.

This specification defines four new error types:

(1) RoutingFailure. Since ebMS intermediaries are configured using a routing function (see section 2.5.5), an ebMS Error MUST be generated when a message cannot be routed by the Intermediary because the message does not match any entry in the routing function. This new type of error MUST be supported by intermediaries and SHOULD be reported:

- Error ID: EBMS:0020,
- Short description: RoutingFailure
- Severity: failure
- Description: the Intermediary MSH was unable to route an ebMS message and stopped processing the message.

Note that finding a match for a message in the routing function is separate from applying the forwarding action associated with the pattern. For instance, in situations where the intermediary operates in a store-and-forward mode (see section 2.5.2), the actual forwarding of a message may take place well after the initial processing of the incoming message and after the incoming transport connection on which it came in was closed. If synchronous reporting pattern (c) is defined for a message pushed to an intermediary, then the intermediary SHOULD match the message, immediately upon receiving it and before the incoming connection is closed, against its routing table, in order to establish whether a matching rule exists. This allows the intermediary to provide immediate feedback to the MSH that pushed the message to it.

A non-normative example of a routing error signal is provided in appendix E.5 .

Two other new ebMS error types are relevant in situations where the routing rule of the intermediary specifies that another MSH (either the final destination or the next intermediary) is to pull the message based on its message partition channel identifier:

2) MPCCapacityExceeded. When the intermediary attempts to store the message, it is possible that this attempt fails due to a lack of capacity, similar to the EBMS:0005 “push” connection failure. In this situation, the intermediary SHOULD generate and report the following new error type:

- Error ID: EBMS 0021
- Short description: MPCCapacityExceeded.
- Severity: failure
- Description: an entry in the routing function is matched that assigns the message to an MPC for pulling, but the intermediary MSH is unable to store the message with this MPC.

This error is compatible with, and can be reported using any of the three reporting options.

3) MessagePersistenceTimeout. The intermediary MAY also impose a limit on the time it will wait for the message to be pulled, and discard messages that have not been pulled after the time limit is reached. If such a limit is set, the intermediary SHOULD generate and MAY report the following new kind of error:

- Error ID: EBMS0022
- Short Description: MessagePersistenceTimeout.
- Severity: Failure
- Description: An intermediary MSH has assigned the message to an MPC for pulling and has successfully stored it. However the intermediary set a limit on the time it was prepared to wait for the message to be pulled, and that limit has been reached.

This error, by definition, is typically generated well after the message is submitted to the intermediary. This error therefore in general cannot be reported using the synchronous reporting option (c).

4) MessageExpired. It is possible that delays in the forwarding of messages in the I-Cloud cause messages to expire before they reach the destination endpoint MSH. An intermediary MAY check if messages have expired before forwarding them. Intermediaries MAY use one or multiple mechanisms to determine expiration. In particular, an intermediary MAY inspect message headers, even if it is not explicitly identified as the target of those headers, or inspect message payloads. Such mechanisms could be specified in profiles as an extension to this specification. A mechanism that an intermediary MAY use to determine expiration for an intermediary is to interpret expiration information in the WS-Reliability [WSR11] header or in the WS-Security header. These headers express expiration information as follows:

- WS-Reliability defines the `wsr:Request/wsr:ExpiryTime` element.
- WS-Security defines the `wsu:Timestamp/wsu:Expires` element.

If an intermediary applies this or other expiration detection mechanisms and determines a message has expired, it SHOULD discard the message and SHOULD generate the following error:

- Error ID: EBMS0023
- Short Description: MessageExpired
- Severity: Warning

- Description: an MSH has determined that the message is expired and will not attempt to forward or deliver it.

The intermediary MAY generate the error silently (without any notification to any other MSH), or it MAY report the error using one of the three error reporting mechanisms.

Note that message expiration detection at intermediaries is just an optimization to prevent unnecessary forwarding of messages. If endpoint MSHs implement expiration checking, the end-to-end exchange is not affected if intermediaries do expiration checking too.

To prevent looping of error messages, intermediaries SHOULD NOT return standalone Error messages for messages that themselves transmit routing errors in situations where the reporting mechanism is other than (c). Note that this does not prevent an intermediary from generating and reporting such errors in other situations, as long as they are not sent out as error messages. In situations where errors are piggy-backed on an ebMS user message, it is still possible to fault the `eb3:UserMessage` unit and send the related error. In situations where an intermediary is operating in “streaming” mode, it MAY pass on an error message received synchronously on an outgoing connection from the next hop back on the incoming connection to the preceding hop.

2.6 Endpoint requirements

Intermediaries are responsible for the routing of messages through the I-Cloud based on the metadata available in the routed message. The MSH sending a message to the I-Cloud is responsible for providing appropriate header data in the message so that the routing functions of the intermediaries can perform their function and forward the message to its destination.

For ebMS user messages this poses few additional requirements on endpoints: the routing input is already included in the `eb3:Messaging` SOAP header block. This endpoint MUST target the header to the `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/nextms` role if specified in the **Pmode[1].Protocol.AddActorOrRoleAttribute** parameter described in sections 2.4.5 and 12.1 . For other messages (ebMS Signal messages, or non-ebMS messages) the endpoint MUST insert one or more WS-Addressing headers and use these to carry routing input. The following requirements MUST be observed by endpoints when involved in a multi-hop message exchange.

2.6.1 Routing Support for Initiating Messages

The Initiating message is defined as the first message sent by the Initiating MSH as defined in section 2.2.3 of the Core specification. As shown in section 2.4.7 , in a multi-hop configuration both an endpoint MSH (cases 1, 2, 3a) and an Intermediary MSH (case 3b and 4) can be the initiator of a message exchange. Three kinds of initiating messages can be distinguished:

1. **ebMS User Messages**: Such messages are either the “request” leg of an ebMS Two-way or the “oneway” leg of an ebMS One-way. The `eb3:UserMessage` header element contains all routing input. The content of this header is determined jointly by the PMode configuration of the Sending endpoint and/or by the submitting message Producer.

2. **ebMS Signal Messages:** The most common case of an initiating ebMS Signal message is the `eb3:PullRequest` message sent by an Intermediary to pull a message from the endpoint (see cases 3a, 3b and 4 in section 2.4.7). Some `eb3:Error`s may also be generated as initiating messages, i.e. not as responses to faulty messages.

When these signals are sent by the Intermediary directly to the Endpoint and therefore do not need to be routed (cases 3b and 4 in section 2.4.7), no routing info is needed. When the messages are sent by an Endpoint (case 3a) and do need to be routed to another endpoint, they should contain routing information. The ebMS header of such messages does not contain an `eb3:UserMessage` element.

For Pull requests, there are two options (configurable via the Pmode parameters **Pmode[1].Protocol.AddActorOrRoleAttribute** and **Pmode[1][s].Addressing.EPR**):

- The Pull Request is contained in an `eb3:Messaging` header targeted to the ebMS intermediary that references a channel via its `@mpc` attribute. Intermediaries can use routing tables based on these channel identifiers to route the requests through the I-Cloud .

- The PMode associated with the messages to be pulled (i.e. defining a One-way / Pull MEP) is shared between partners, along with the EPR of the sender of the pulled message. From this EPR the `ebint:RoutingInput` reference parameter is extracted and added as a WS-Addressing header to the SOAP header and targeted to the ebMS intermediary. In this case the signal message is targeted to the SOAP ultimate receiver and the content of the `eb3:PullRequest` itself is not used for routing.

The EPR for the request signal to pull a user message can be inferred from the Pmode as follows:

- The `eb3:From` and `eb3:To` values should be swapped.
- The `eb3:Service` is the same, but the fixed string “`.pull`” is appended to the value of `eb3:Action`.
- The MPC, if mentioned, is used in the `eb3:PullRequest`.

3. **non-ebMS messages:** Such messages are typically bound to the first leg of an underlying two-way transport protocol (e.g. HTTP). They do not contain an `eb3:Messaging` header (unless they are piggybacked on ebMS Messages in which case the routing rules for these messages apply). They may contain a `wsa:ReplyTo` header. The WS-Addressing headers and reference parameters contain all routing input. The content of these headers is determined by the PMode configuration of the Sending endpoint as in (2). Such messages may be:

- WS-ReliableMessaging sequence management messages (such as `wsrm:CreateSequence` and `wsrm:TerminateSequence`)

- WS-SecureConversation and WS-Trust messages, used to establish and manage security contexts (`wst:RequestSecurityToken`, etc.).

2.6.2 Routing Support for Response Messages

A response message is defined as a message that is sent in response to - and by the Receiver of - a previous (request) message, in general back to the Sender of this request. For example, a “reply” message in an ebMS Two-way MEP. More precisely: a message that relates to a previous message in any of the three following ways:

- The response message is an ebMS user message that is the second leg of an ebMS two-way MEP, and relates to the first message using `eb3:RefToMessageId`

- The response message is an ebMS signal message that is referring to a previous ebMS message using `eb3:RefToMessageId`, regardless of the type of MEP the previous message is involved in and its role in the MEP. This concerns `eb3:Error` messages and `eb3:Receipt` messages.

- The response message is NOT an ebMS message, but an accessory message such as an RM signal that is responding to a previous RM message (e.g. `wsrm:CreateSequenceResponse`) or to a group of such messages (e.g `wsrm:SequenceAcknowledgment` message), or yet a WS-SecureConversation message such as `wst:RequestSecurityTokenResponse`.

Support for response message routing falls into one of these four cases:

1. ebMS User Messages: the `eb3:UserMessage` header element contains all routing input. The content of these headers is determined as for request user messages. In case the `wsa:ReplyTo` header was present in the request message, and if its EPR value contains the `ebint:RoutingInput` element, then the corresponding WS-Addressing reference parameter header block MUST be present in the response message – and will therefore take precedence as the routing input for the response message. In case the `wsa:To` element is present in the response message (from the `wsa:Address` element present in the `wsa:ReplyTo` EPR) it MAY be used as routing input by an intermediary.

2. ebMS Signal Messages: These are either `eb3:Error`s or `eb3:Receipt`s sent in response to a previously received ebMS message. The ebMS header of such messages does not contain an `eb3:UserMessage` element. However, either one of these conditions MUST be met and decided per agreement between communicating parties:

1. the wsa:ReplyTo header was present in the request message, and if its EPR value contains the ebint:RoutingInput element. In that case (and unless the EPR also contains a reachable URI – see Section 2.5.6) this element is used to generate a corresponding WS-Addressing header that will be used by the I-Cloud routing function. For eb3:Error, the wsa:FaultTo if present must take precedence over wsa:ReplyTo.

2. the PMode associated with the request message is shared between partners, and the sender of the response message extracts the EPR of the initial sender from this PMode (even if it has an anonymous URI). It inserts the ebint:RoutingInput reference parameter obtained from this EPR in the header of the response message. This header will be used by the I-Cloud routing function.

3. non-ebMS messages: Such messages do not contain any eb3:Messaging header (unless they are piggybacked on ebms Messages in which case the routing rules for these messages apply). The same rules as for above case (2) apply: the routing is based on WS-Addressing reference parameter header ebint:RoutingInput (unless the EPR also contains a reachable URI – see Section 2.5.6) An exception is made for RM protocol messages (such as acknowledgments, or sequence management messages): In that case – and assuming that end-to-end reliable messaging is used - the EPR of the request sender MUST be specified in the wsrm:AcksTo element provided when requesting or accepting an RM sequence creation. Either one of the following conditions MUST be met:

- This EPR contains the ebint:RoutingInput element suitable for routing across the I-Cloud back to the initial message sender.
- The wsa:Address element in this EPR contains a URI identifying the request message sender so that it can be used as routing input by Intermediaries or used to directly reach the destination MSH, bypassing the I-Cloud and not relying on any ebMS-related routing function.
- This EPR has an wsa:Address element that contains a reachable URI identifying the edge intermediary used by the request message sender, as well as an ebint:RoutingInput containing an @mpc value, so that the message can be pulled by the request sender on this MPC.

4. Inferred RoutingInput for the reverse path

In case no WS-Addressing header indicates the return destination (wsa:ReplyTo or wsa:FaultTo are not used in the initiating message), and in case no response EPR is known from the responding endpoint (not specified in its PMode), then the reverse RoutingInput value for response messages SHOULD be automatically inferred from the RoutingInput in the request message as follows:

- The eb3:From and eb3:To values should be swapped.
- The eb3:Service is the same, but eb3:Action is appended with ".response", generalizing over all response messages.
- In case an MPC is mentioned, derive a new MPC by similarly concatenating a ".response" suffixes . E.g. on the first hop, a One-way / Push message is sent to MPC "abc123". The eb3:Receipt or eb3:Error is automatically routed back to MPC "abc123.response". Inferring a new MPC for these response messages allows the original Sender to automatically know which MPC it can then pull to retrieve Receipts, Acknowledgements and Errors, in case this Sender is not addressable for callbacks.

More fine-grained variations of this scheme are possible, distinguishing different types of responses, e.g. appending ".receipt" (in case an eb3:Receipt is sent back), ".acknowledgment" (reliable messaging acknowledgments) or ".error" (eb3:Error messages).

Note that this same mechanism of inferring the ebint:RoutingInput can be applied by intermediaries in the case of routing errors. This is important because intermediaries have no access to Pmode configurations at the endpoints.

2.6.3 WS-Addressing Support

Endpoints are required to support WS-Addressing, to use and interpret WS-Addressing endpoint references as described in section 2.6.3.1 and use WS-Addressing headers as described in section 2.6.3.2 .

2.6.3.1 WS-Addressing Endpoint References

WS-Addressing EndPoint References (EPRs) are used in this specification to identify endpoint MSHs. WS-A EPRs primarily use URLs to identify endpoints. However, part of the rationale of ebMS multi-hop topologies is that the Internet address (URL) of destination endpoints is either not resolvable directly by the original sender, or even not to be published (e.g. it may change overtime, or be meaningful only for a local network). Consequently a WS-A reference parameter is used to allow the routing of messages through the I-Cloud as defined in section 2.5.5 .

When the actual address (URL) of the destination Endpoint of a multi-hop path is supposed to remain unknown from other MSHs, or is not supposed to play any role in the transfer of messages addressed to this Endpoint, the EPR of the Endpoint MUST use the following URI in its `wsa:Address` element (also called the “icloud” URI):

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/icloud

The EPR of such Endpoint MSHs will be of the form:

```
<wsa:EndpointReference>
 <wsa:Address
   S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"
  >http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/icloud</wsa:Address>
  <wsa:ReferenceParameters>
   <ebint:RoutingInput
    S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh">
        <ebint:UserMessage>...</ebint:UserMessage>
      </ebint:RoutingInput>
  /wsa:ReferenceParameters>
</wsa:EndpointReference>
```

The `@S12:role` attribute indicates the endpoint reference addres and reference parameters MUST be interpreted by ebMS routing intermediaries in the I-Cloud, as discussed in section 2.4.5 . If using SOAP1.1 instead of SOAP 1.2, the EPR would reference the `@S11:actor` attribute instead.

This EPR indicates the need to rely exclusively on information in the reference parameter `ebint:RoutingInput` to route the messages intended to such an endpoint. The “icloud” URI must then appear in the `wsa:To` header of any message intended to this destination Endpoint, along with the `ebint:RoutingInput` header block that represents the reference parameter.

The minimum set of WS-Addressing headers that must be present in a message intended to an Endpoint described by the above EPR, is illustrated below:

```
<S12:Header>
  <wsa:To
    S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"
    >http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/icloud</wsa:To>
  <wsa:Action >…</wsa:Action>
  <ebint:RoutingInput wsa:IsReferenceParameter='true'
    S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh">
      <ebint:UserMessage>...</ebint:UserMessage>
  </ebint:RoutingInput>
```

```
</S12:Header>
```

The value of the `@S12:role` attribute is the value introduced in section 2.4.5 . When using SOAP 1.1 the attribute to use is `@S11:actor` as discussed before.

2.6.3.2 Use of WS-Addressing Headers

Reliance on WS-Addressing headers other than those supporting the routing function (i.e. headers containing reference parameters) is not required for ebMS multi-hop processing, However, in order to comply with the SOAP binding of message addressing properties defined in [WSADDRSOAP] when a reference parameter header block is present – indicating a destination endpoint described by an EPR - at minimum the `wsa:To` and `wsa:Action` headers must be present, with values corresponding to those in the EPR definition.

Using `wsa:Action`:

Unless specified otherwise (e.g. determined by a WSDL definition for a back-end Service), it is RECOMMENDED to set the `wsa:Action` to the following default values which indicate the ebMS MEP definition URI that the message participates in, with a suffix of the form "`.request`" or "`.reply`" specifying the "role" of the message in the MEP in case of a two-Way MEP. Here are the expected values for `wsa:Action`:

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/twoWay.request

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/twoWay.reply

In case the message contains an `eb3:Receipt` for an ebMS user message, it is RECOMMENDED that the `wsa:Action` value has a similar value based on whether the whether the received message is One Way message or the request or a response in a Two Way message exchange, again with "`.receipt`" as suffix:

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay.receipt (for the receipt of a One Way message exchange)

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/twoWay.request.receipt (for the receipt of a request message in a Two Way message exchange)

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/twoWay.reply.receipt (for the receipt of a response messages in a Two Way message exchange)

In case the message is an `eb3:Error` for a message containing any of the above Action value, it is RECOMMENDED that the `wsa:Action` value reuses the same value, with "`.error`" as additional suffix.

Note that the `wsa:Action` attribute MUST be an IRI as defined in [RFC3987]and has basically the same structure as an URL.

If a value different from these recommended values is to be used (e.g. to target the message to a particular Web Service), this SHOULD be indicated using the the WS-Addressing processing mode parameter defined in section 2.7.1.3 . There is no `S12:role="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/nextmsh"` on the `wsa:Action` block as it is the ultimate receiver of the SOAP message, not ebMS intermediaries, that needs to interpret the value of `wsa:Action`.

Another situation in which the default values are not to be used for `wsa:Action` is when the reference parameter is used to route Web Services protocol messages that have required values. For instance, WS-ReliableMessaging [WSRM11] reserves the value http://docs.oasis-open.org/ws-rx/wsrm/200702/CreateSequence for the action of a sequence creation message.

Using `wsa:ReplyTo`:

Some Endpoint MSHs MAY decide to make a more advanced use of WS-Addressing, e.g. by using `wsa:ReplyTo` EPR for providing response message routing information in a two-way MEP, instead of relying on PMode configuration. Reliance on `wsa:ReplyTo` for the routing of a response message is neither necessary nor required, as the PMode associated with the MEP that describes this response will normally contain sufficient routing information (e.g. in form of the destination EPR for this response).

2.7 PModes for multi-hop

2.7.1 PMode Extension for Web Services Addressing

When it is necessary to insert WS-Addressing header elements in a message, an endpoint needs to get this information from the PMode. The PMode governing the sending of messages SHOULD specify whether a reference parameter must be added and what should be included in the parameter, as well as the values for the WS-Addressing Address and Action. This requires an extension of data model for PModes as presented in the core specification. A new composed PMode parameter named **Addressing** is introduced to specify this additional information. As WS-Addressing headers are involved in the sending of both request and response messages, the EPR parameter may occur multiple times within one PMode. The next section defines the parameter and in section 2.7.1.2 its usage is defined.

2.7.1.1 Addressing.Address

Addressing.Address indicates the address URI of the WS-Addressing endpoint reference. In a multi-hop context where the routing function is used, its value is expected to be: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/icloud.

2.7.1.2 Addressing.EPR

The **EPR** parameter is composed of the sub-element **RoutingInput. Addressing.EPR.RoutingInput** represents the value of the RoutingInput element that is used as the reference parameter of the WS-Addressing EPR. The parameter is composed of the following optional sub-elements which define the value to use in the child elements of RoutingInput. The mapping of these parameters to the child elements is analogous tot the mapping of the existing PMode parameters.

- **Addressing.EPR.RoutingInput.Initiator.Party**;
- **Addressing.EPR.RoutingInput.Initiator.Role**;
- **Addressing.EPR.RoutingInput.Responder.Party**;
- **Addressing.EPR.RoutingInput.Responder.Role**;
- **Addressing.EPR.RoutingInput.BusinessInfo.Service**;
- **Addressing.EPR.RoutingInput.BusinessInfo.Action**;
- **Addressing.EPR.RoutingInput.BusinessInfo.Properties[]**;
- **Addressing.EPR.RoutingInput.BusinessInfo.MPC**

Notes:

- The definition of the EPR parameter does not include sub-elements for all children of the `ebint:RoutingInput` element. For example, there is no parameter for configuration of the MessageInfo element. It is expected that, when used, these values will be derived in some way from the message being sent and that the configuration of this derivation is implementation dependent.

- The content of the `ebint:RoutingInput` XML header in ebMS SOAP messages MUST conform to the RoutingInput schema. This does not mean that all elements defined as required in the RoutingInput schema must be defined in the Pmode. An MSH SHOULD set the values of elements not defined in the EPR based to the inferred values as described in section 2.6 . The EPR in the Pmode should define values for the *effective* routing input elements.

- **RoutingInput.BusinessInfo** concerns the destination endpoint. All or a subset of the children parameters {Service, Action, Properties, MPC} of this parameter may be specified. Although several Service instances may be associated with an endpoint (same for Action and MPC), it is possible that only one instance of {Service, Action, Properties, MPC} values is known from the I-Cloud routing function and is necessary to route all messages to this endpoint. In that case, the EPR PMode parameter associated with this endpoint will only contain this instance of {Service, Action, Properties, MPC} under its BusinessInfo sub-parameter. All PModes associated with this destination endpoint will refer to this same EPR and its unique RoutingInput.BusinessInfo parameter even though each one of them may use a different Service / Action pair (or different MPC) in its PMode.BusinessInfo parameter. In such cases the EPR.RoutingInput reference parameter must always be used even for routing User Messages, as the routing function would be unable to use other Service / Action pairs - as specified under the `eb3:Messaging/eb3:UserMessage` element - that are also supported by this endpoint.

- For reasons of reuse of types and elements defined in the ebMS 3.0 XML schema, the `ebint:RoutingInput` contains a `CollaborationInfo` group. This group contains a `ConversationId` element. This value of this element is not set via a Pmode parameter, and is left to implementations. It MAY be set in ways to support monitoring:
    - A forward message containing a CreateSequence request sent to establish (just-in-time) an RM sequence for a particular user message MAY use the `ConversationId` of that user message for its `ConversationId`.
    - A reverse message containing an ebMS error or receipt for an ebMS message MAY reuse the ConversationId of that message.

2.7.1.3 WS-Addressing Action

Section 2.6.3.2 describes situation in which the `wsa:Action` element is required and defines default values. In situations where a different value is required, this value can be agreed using the following new parameter:

Pmode.Addressing.Action: Specifies the value to use for `wsa:Action`. If this parameter is not specified, and the value is not determined by other SOAP processing modules (e.g. reliable messaging) an implementation SHOULD use the recommended default values.

2.7.1.4 PMode extensions

The Addressing parameter defined above may be repeated at different places in the PMode, where it is necessary to indicate a multi-hop destination. In particular, it may appear in the PMode as follows:

Pmode.Initiator.Addressing: To specify the EPR value to be used for routing any message to the Initiator endpoint of a PMode.

Note: In case this EPR parameter only contains the Address element and no RoutingInput element, the Address value MUST be sufficient to directly address the endpoint (meaning “response” messages sent by the Responder are not routed via the I-Cloud) .

Pmode.Responder.Addressing:To specify the EPR value to be used for routing any message to the Responder endpoint of a PMode.

Pmode[2][s].Addressing

This parameter is needed to be able to express whether or not an `ebint:RoutingInput` reference parameter and other WS-Addressing headers needs to be added to Pull Request messages. These two options are explained in section 2.6.2 case (2). The contents of the reference parameter MAY also be inferred from the Pmode.BusinessInfo for the user message using the algorithm described in that section.

Pmode[1].Errorhandling.Report.SenderErrorsTo.Addressing: To specify the destination to be used for routing any Error generated by the UserMessage-sending endpoint for this PMode leg. The default is the EPR associated with the Sending MSH for this PMode leg (either Pmode.Initiator.Addressing.EPR or Pmode.Responder.Addressing.EPR)

Pmode[1].Errorhandling.Report.ReceiverErrorsTo.Addressing: To specify the destination to be used for routing any Error generated by the UserMessage-receiving endpoint for this PMode leg. The default is the EPR associated with the Receiving MSH for this PMode leg (either Pmode.Initiator.Addressing.EPR or Pmode.Responder.Addressing.EPR)

Pmode[1].Security.SendReceipt.Addressing: To specify the destination to be used for routing any `eb3:Receipt` generated in response to a User Message send over this PMode leg. If not present, Receipts will be routed using the EPR associated with the UserMessage-sending endpoint (either PMode.Initiator.Addressing or PMode.Responder.Addressing)

Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo.Addressing: To specify the destination to be used for routing any reliable messaging response messages related to a User Message send over this PMode leg. If not present, RM response messages will be routed using the EPR associated with the UserMessage-sending endpoint (either PMode.Initiator.Addressing or PMode.Responder.Addressing)

Note: When **PMode[1].Reliability.AtLeastOnce.Contract** is set to "true" then Acknowledgements as well as other RM signals sent by the RMD MUST be sent back to the Sending party for this PMode leg. In case the **PMode[1].Reliability.AtLeastOnce.Contract.AcksTo** parameter is set, but no value is supplied for **Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo.Addressing**, then its value MUST be a URI as described in ebMS V3 Appendix D.3.5., that resolves to the RMS associated with the sending endpoint, in which case these signals will be directly addressed to this RMS without multi-hop routing.

When using WS-ReliableMessaging as reliable messaging protocol, the value of the **Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo.Addressing** when available MUST be used to construct the `wsrm:AcksTo` element in the `wsrm:CreateSequence` initiatialization message.

If a value is supplied for **Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo.Addressing** with an address http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/icloud then the value of **Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo** MUST be the URI of the intermediary that will route the acknowledgment back through the I-Cloud. Note that this parameter may be specified differently for the Sender (see section 2.7.2 , **init** Pmode) than for the Receiver (idem, **resp** Pmode).

If a value is supplied for **Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo.Addressing** with an address other than the I-Cloud IRI, then this MUST be an EPR associated with the RMS of the sending endpoint, containing an address that does not need be resolved by the I-Cloud routing function, meaning these RM signals are directly sent back to the MSH without multi-hop routing. In this case no value SHOULD be specified for **Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo**.

The use of the EPR.RoutingInput extension will generally be required for non-ebMS messages (e.g. RM protocol messages) and may be required for signals (`eb3:Receipt`, `eb3:Error`) although the reverse routing of these messages may rely on information provided in the WS-Addressing header of the related request message (`wsa:ReplyTo`) as described in section 2.6.2.

2.7.2 PMode units

The PMode governing a multi-hop MEP is actually divided in two parts or "units", each one of these is actually structured as a separate PMode:

- The PMode unit that governs the Initiator edge-hop (linking the Initiator endpoint to the first ebMS Intermediary on a multi-hop path originating in the Initiator). The ID of this PMode (PMode.ID) MUST be of the form: <ID>".init", where <ID> is a valid value for PMode.ID. This unit is called the "**init**" PMode for the multi-hop MEP.
- The PMode unit that governs the Responder edge-hop (linking the Responder endpoint to the last ebMS Intermediary on a multi-hop path originating in the Initiator). The ID of this PMode (PMode.ID) MUST be of the form: <ID>".resp", where <ID> is the same value used in the PMode.ID for the corresponding Initiator edge-hop. This unit is called the "**resp**" PMode for the multi-hop MEP.

The ebMS 3.0 Core Specification states that the optional `pmode` attribute of the `eb3:AgreementRef` element, if used, contains the Pmode.ID parameter. In a multi-hop context, it MUST contain the Pmode.ID parameter without the ".init" and ".resp" suffixes.

As the two edge hops of a same multi-hop path used by an MEP instance are now controlled by two different PMode units, these two PMode units must be aligned for proper interoperability.

Some PMode parameters are expected to be shared between an "init" PMode and its "resp" counterpart PMode, while other parameters are likely to be different, in order to accommodate different MEP binding conditions on each edge-hop.

The following PMode parameters MUST be common between two related PMode units:

- **PMode.ID** (with ".init" appended for the init PMode unit, and ".resp" appended for the resp PMode unit.)
- **PMode.MEP (**either one-Way or two-Way)
- **PMode.Initiator** (and sub-elements)
- **PMode.Responder** (and sub-elements)
- **PMode[1].Protocol.SOAPVersion** (as the SOAP envelope must not change over a multi-hop path.)
- **PMode[1].BusinessInfo** (and sub-elements)
- All the Reliability parameters **(PMode[1].Reliability** and sub-elements), except for **PMode[1].Reliability.AtLeastOnce.Contract.ReplyPattern** which may be different in the init and the resp PMode units.
- All the Security parameters **(PMode[1].Security** and sub-elements), except for **PMode[1]. Security.SendReceipt.ReplyPattern** which may be different in the init PMode and the resp PMode units.

Some PMode parameters are likely to be different between two related PMode units:

- **PMode.MEPbinding,** which may vary over edge-hops for the same MEP (as shown in the section "MEPs and Channel Bindings").

  In order to minimize the impact over an already deployed endpoint MSH when upgrading a point-to-point exchange into a multi-hop exchange, the existing PMode model as defined in ebMS core V3, is not extended with new MEP binding values in order to implement the new multi-hop bindings (edge bindings) defined in 2.4.7 and 2.4.8 .

  This allows each unit to have its own value for the MEPbinding parameter indicating how messages are exchanged over the edge hop:

  - For the "init" unit it defines the exchange between the **Initiator** MSH and the **first Intermediary** MSH;
  - For the "resp" unit it defines the exchange between the **last Intermediary** MSH and the **Responder** MSH
- **PMode[1].Protocol.Address**: The PMode deployed on the Initiator side will update this to the URL of the immediate Intermediary MSH. (In case of a Pull MEP binding where the endpoint MSH acts as Initiator, this address indicates the URL of the Intermediary to be pulled from.) Note that in a multi-hop path, some hops may use HTTP and others SMTP transport.
- **PMode[1].ErrorHandling.Report.AsResponse**: This parameter only concerns the receiving side of an ebMS message. It may need be modified as it may no longer be the same for each edge-hop of the same path. This will be more often the case for the Sender endpoint (first hop) than for the Receiver destination endpoint. For example, the first edge-binding may not be able to keep the transport connection open to wait for the error generated by the ultimate destination endpoint. In that case, the value will be "false" on the Sender side, while it may still be "true" on the Receiver side
- **PMode[1].Reliability.AtLeastOnce.ReplyPattern**: This parameter may differ between the init PMode unit and the resp PMode unit, as it governs the bindingof RM Acknowledgements and other RM signals to the underlying protocol (e.g. HTTP response vs. HTTP request.)

- **PMode[1].Security.SendReceipt.ReplyPattern**: This parameter may differ between the init PMode unit and the resp PMode unit, as it governs the binding of `eb3:Receipt`s to the underlying protocol (e.g. HTTP response vs. HTTP request)

4.1.1 Migrating PModes from point-to-point to multi-hop

When two partners who were communicating in point-to-point mode need to migrate to a multi-hop configuration, it is expected they will derive their PMode units for the multi-hop exchanges from the existing PModes that govern their point-to-point exchanges.

When migrating from a point-to-point to a multi-hop exchange, endpoints however may have to modify their behavior compared with a point-to-point exchange. For example, depending on the multi-hop channel binding (see section 2.4.6) they may have to pull `eb3:Receipt` messages related to pushed User messages, whereas in a point-to-point exchange such messages could be received over the same connection as the User message (e.g. on the HTTP response). Also, when using reliable messaging (WS-RM) the routing of RM signal messages such as `wsrm:CreateSequence`, requires the addition of reference parameters (conforming to WS-Addressing) in the SOAP header.

4.2 Details of Reliable and Secure Multi-hop

End-to-End reliability as well as end-to-end secure conversations MUST be supported by the I-Cloud, meaning an entire multi-hop path between two endpoints can be made reliable with these endpoints acting as source and destination of a reliable messaging sequence. This does not exclude other reliability schemes that MAY be supported, e.g. where a multi-hop path is split into several reliable segments i.e. where some Intermediary is acting as reliability endpoint.

4.2.1 Controlling Sequencing and Grouping

As required in Core ebMS V3 specification, an endpoint MSH MUST be able to control which reliable (RM) sequence is used by the User Message it sends, if only because different reliability QoS may be associated with different reliable sequences. This is also the case for security: a particular security context may be associated with an RM sequence, or more generally with a sequence of messages (conversation).

This includes the ability to initiate RM sequences - possibly secure - as required by the PModes of messages to be sent, and to associate such sequences with any further message sent for this PMode. The main difference between a multi-hop environment and a peer-to-peer environment, is the need to add sufficient routing input to:

- RM Lifecycle Messages (CreateSequence, CloseSequence, TerminateSequence, Acknowledgements)
- WS-SecureConversation messages used for establishing and managing the security context.

This routing input is obtained from Reference Parameters associated with the destination EPR. Such a Reference Parameter value SHOULD be specified in the EPR.RoutingInput PMode parameter associated with the destination.

The following rules apply when creating reliable sequences and secure conversations for multi-hop:

- An end-to-end reliable (RM) sequence is always associated with a unique destination MSH, although it MAY be used for several types of message exchanges governed by different PModes, if all these messages require the same level of reliability QoS (see the PMode parameters: AtLeastOnce, AtMostOnce, ExactlyOnce). It is however RECOMMENDED that all messages sent over the same sequence, are sent over the same MPC.
- When a CreateSequence message contains an `ebint:RoutingInput` reference parameter (as a SOAP header block), and if the MPC value (`@mpc` attribute) is specified (`S12:Header/ebint:RoutingInput/ebint:UserMessage/@mpc)` then all User messages sent reliably over this sequence MUST be intended for the same MPC, i.e. have same `ebint:UserMessage/@mpc` value.

- An end-to-end secure conversation is always associated with a unique destination MSH, although it MAY be used for several types of message exchanges governed by different PModes and MAY be associated with different MPCs.
- The combination of a reliable sequence and of a secure conversation, is a "secure sequence" - as defined in WS-I Reliable and Secure Profile [WSIRSP10] - meaning that a security context is associated with the RM sequence. A secure RM sequence combines the above requirements, and must be initiated with an `wsrm:CreateSequence` element that has been extended with a `wsse:SecurityTokenReference` element.

Messages with different *available* routing inputs (see 1.6.4) may share the same RM sequence provided they are routed to the same destination. This routing is determined by a subset of the routing input: the *effective* routing input. The routing function of the I-Cloud may be configured to route messages with different Effective routing inputs to the same destination. However, for robustness of the reliability mechanism it is RECOMMENDED to only send messages with same effective routing input over the same RM sequence. The same recommendation applies for secure conversations.

The PMode parameter **PMode[1].Reliability.Correlation** can help enforce that only messages with the same Effective routing input will be assigned to a particular RM sequence. For example, if the routing is determined by the following Effective routing input:

1. `eb3:CollaborationInfo/eb3:Service`
2. `eb3:CollaborationInfo/eb3:Action`
3. `eb3:PartyInfo/eb3:To/eb3:PartyId`

Then the following PMode value will ensure (or specify) that only messages sharing the same values for the above elements are assigned to the same RM sequence:

PMode[1].Reliability.Correlation =

```
”ebint:UserMessage/eb3:CollaborationInfo/eb3:Service,
ebint:UserMessage/eb3:CollaborationInfo/eb3:Action,
ebint:UserMessage/eb3:PartyInfo/eb3:To/eb3:PartyId”
```

if an RM sequence is intended to be shared by messages governed by different PModes, this will only be possible if these PModes use the same Correlation expression, and if messages across these PModes share same values for these expressions.

NOTE: the Correlation expression does not have to replicate the effective Routing input elements. It can use a different expressions(e.g. an MPC value, a message property) as long as it is known that the same value for this expression will result in the same routing destination.

4.2.2 Configuring Secure Conversations

Similarly, messages related to a PMode may be associated with the same secure conversation. This indicated with the PMode parameter:

PMode[1].Security.Conversation = true.

The assignment of messages to the same conversation follow the same rule as for RM sequences. The **PMode[1].Security.Conversation.Correlation** PMode parameter will allow to determine which messages can be assigned to the same secure conversation, in the same way as for RM sequences.

In the case of a secure (RM) sequence, both **PMode[1].Reliability.Correlation** and **PMode[1].Security.Conversation.Correlation** will define the same expression.

The responses to RM lifecycle messages (CreateSequence, CloseSequence, TerminateSequence, AckRequested).as well as responses to the establishment of secure conversations are routed according to cases 3 or 4 in section 2.6.2 (Routing support for response Messages) .

Note that these parameters do not fully specify the use of secure conversations with ebMS. Future profiles MAY define additional processing mode parameters for this.

4.2.3 Reliable Message Pulling

When messages must be pulled reliably (contract: AtLeastOnce), the Core V3 specification requires that the PullRequest signal be also sent reliably. This requirement is valid for point-to-point, but not always for multi-hop. Indeed, in a point-to-point context, resending a PullRequest message may represent the only opportunity for the pulled endpoint to resend a message that has already been pulled but possibly lost on its way (e.g. in the case of an HTTP transport, resending on HTTP back-channel would require the receiving MSH to open an HTTP connection).

In a multi-hop context, the following multi-hop One-Way pulling cases must be considered:

1. **End-to-end pulling** (as described in section 2.4.7.2 , Pulling Messages from the Sender Endpoint, Case 3 – "end-to-end pulling"). This case is handled in the same way as reliable point-to-point, from a reliable messaging viewpoint. The `eb3:PullRequest` message itself SHOULD be sent reliably (AtLeastOnce contract) in order to support AtLeastOnce contract for pulled messages, as the resending of PullRequest messages will provide – for some request-response transports such as HTTP – a back-channel for resending pulled messages that may have already been sent. The behavior of sending PullRequest signals reliably is normally assumed when ensuring reliable One-Way / Pull, as in point-to-point.

2. **One-hop pulling(s).** Here there is no routing of the `eb3:PullRequest` signal. This case covers two subcases:

   a. Receiving-side pulling and Sending-side pulling, or one-hop pull for each edge-hop. The case is illustrated in section 2.4.7.2 , Pulling Messages from the Sender Endpoint, Case 4

   b. Receiving-side pulling and Sending-side pushing. This case is illustrated in Section 2.4.7.1 Pushing Messages from the Sender Endpoint (Case 2), involving pushing from the Sending endpoint (first edge hop) and Pulling from Receiving endpoint (last edge hop).

In both cases, the routing function inside the I-Cloud may be configured for any combination of pull and push along the multi-hop path, and any pull is only a one-hop pull (Pull may be relayed, as in the "pull-on-pull" forwarding pattern (d), section 2.5.3) , For this reason, there is no use for sending `eb3:PullRequest` signals reliably from the initiating endpoint, as the resending MSH is not the one receiving the resent `eb3:PullRequest. Eb3:PullRequest` signals SHOULD NOT be sent reliably, in spite of the response expected to be sent reliably. This is controlled by a new P-Mode parameter that overrides the default behavior for One-Way / Pull MEPs:

Pmode[1].Reliability.AtleastOnce.ReliablePull: Indicates whether the `eb3:PullRequest` signal must be sent reliably or not. If "false", the initiating endpoint will not send `eb3:PullRequest` signal messages using reliable messaging.

4.2.4 Considerations about Reliable Messaging Specifications

The Core V3 specification [EBMS3CORE] allows for binding with either WS-Reliability 1.1 [WSR11]or WS-ReliableMessaging 1.1 [WSRM11]OASIS Standards, in order to ensure reliable messaging. Both standards provide the same level of reliability and can be used for end-to-end multi-hop reliability. The following minor differences must be noted:

The routing of RM signals (CreateSequence, etc) and the need to add routing input headers to these, are mostly a requirement tied to the use of WS-ReliableMessaging, as WS-Reliability does not use such signals to manage sequences. Initiating and closing a reliable sequence with WS-Reliability requires less effort in managing routing information.

On the other hand, in the previous sub-case of one-hop pull for each edge-hop (2.a), WS-ReliableMessaging allows for combination with WS-MakeConnection [WSMC] in order to create – in the case of HTTP transport – back-channel opportunities for resending non-acknowledged pulled messages. For this subcase (one-hop pull for each edge-hop), unless the Intermediary is supporting a connection mechanism such as the one specified in WS-MakeConnection, reliability of the multi-hop path for a pulled message (AtLeastOnce contract) would likely be restricted to no more than notification of delivery failure, as the resending capability would require such a mechanism at Intermediary level.

5 Message Bundling

This specification defines a new protocol to bundle message units in ebMS messages and to submit, send, receive, forward and deliver these units as a single SOAP message. This protocol extends the ebMS 3.0 Core Specification and is compatible with the multi-hop profile.

5.1 Definition, Rationale and Requirements

5.1.1 Definition

Message bundling is defined here as aggregating several ebMS message units (see section 2.1.2 of [EBMS3CORE]) in the same SOAP message. The core V3 specification supports some bundling patterns, all of which involve only one user message unit and possibly several signal message units (see section 5.2.4 of [EBMS3CORE]). This specification extends this functionality and supports bundling several user message units and signal message units in the same ebMS envelope.

5.1.2 Rationale

The main rationale for bundling is to handle situations where two endpoints exchange a large number of small messages in a short time interval, where efficiency of message processing is an issue and where it is acceptable that some messages submitted to an MSH are not always sent immediately. This happens in situations such as the following:

- An application generates many small business documents that need to be transferred to business partners. They are not (very) time-critical and many of them have the same ultimate destination.
- A (legacy) batch business application is scheduled to run once every 24 hours outside office hours to process requests. These requests have accumulated over the previous 24 hours and many of them have the same source. The requests are processed and the responses are all submitted to the MSH in a short period.
- An application uses messaging to synchronize or replicate large sets of relatively static data with business partners. With existing partners, only the updates need to be exchanged, so the required bandwidth is limited. However, setting up the exchange for new partners causes a large number of messages to be exchanged.
- A system has been unavailable for some time or has been disconnected from business partners due to network issues. Once it resumes processing its backlog, it causes a large number of messages to be sent.

In situations like these, an MSH that sends the outgoing response messages can use bundling to share the overhead of processing a SOAP message -including security and reliability quality of service- across several message units. Bundling increases the processing throughput of an MSH and allows it to scale to a higher volume of transfer without increasing processing capacity.

A major benefit of supporting bundling at the message protocol level is that it obviates the need for a bundling or batching mechanism at the business document level. This simplifies the design and implementation of business applications that process these documents.

5.1.3 Requirements

This specification aims to address the following requirements:

- For the Producer and Consumer layers, the semantics of bundling is equivalent to sending and receiving a sequence of distinct messages.

- The processing model for the bundle is similar to the model for non-bundled messages (see section 4.1 in [EBMS3CORE]), and the order of processing headers (security, reliability, ebMS for inbound, the reverse for outbound) is the same as for non-bundled messages.
- Business partners must be able to control what message units are bundled, and to control properties of the resulting bundle (such as its maximum size or the maximum delay) using processing mode parameters.
- There is no requirement that bundling only concerns messages that have been submitted to the MSH in a consecutive way or as an explicit bundle. In other words, bundling may be applied (internally in the MSH) to an arbitrary set of submitted messages, regardless of their submission time or mode.
- Implementations MAY also offer an interface to allow more explicit control over bundling or to allow for submission of bundles, as long as the constructed bundles comply with the agreed bundling parameters.
- Errors occurring in message units affect processing of related units in the bundle, but not of unrelated units.

5.2 Packaging and Processing Bundles

Bundling is defined as an extension of regular ebMS processing. By definition, a bundle consists of a single designated message unit called the *primary message unit* and any number of additional *secondary message units* that are packaged with it. The primary message unit, any secondary units and their payloads are packaged as described in section 5.2.1 and processed as described in section 5.2.2 .

5.2.1 Packaging of bundles

The packaging of a bundled ebMS message follows these rules:

- *SOAP header*: the unique `eb3:Messaging` element contains the header of each message unit - either an `eb3:UserMessage` element or an `eb3:SignalMessage` element.
- *SOAP Body*: At most a single child element (payload of one of the bundled messages) must be present in the Body, in order to comply with WS-I Basic Profile requirements. This must be the payload (if any) of the *primary message unit.* Every other payload must appear as a MIME attachment. The payload in the Body is therefore referred to by the `eb3:PayloadInfo` element of the primary message unit header.
- *Attachments*: the payloads of each message unit other than the primary unit become MIME parts of the bundled message. As is the case for the payload included in the SOAP Body, they are referred to by the `eb3:PayloadInfo` element of the related message unit header. The relation between the payload information element and the MIME part is based on MIME Content-ID values, which are required to be globally unique [RFC2392].

There is no requirement that the order of payloads is in any way related to the order in which the ebMS message units are placed in the `eb3:Messaging` container.

When receiving a message bundle, an ebMS MSH MUST process the first child element of the `eb3:Messaging` container element (in message unit order) as the designated primary message unit. When bundling multiple message units, a sending MSH must similarly insert the intended primary message unit as the first element. Any following message units are processed as secondary message units.

A message bundle that contains no secondary message units is equivalent to, and indistinguishable from, a regular ebMS message that contains only a single message unit.

The value of the `eb3:MessageInfo/eb3:Timestamp` element in each message unit SHOULD reflect the time at which the message unit is created, NOT the time at which the message unit is bundled with other message units and sent. As a consequence, the values of this element in different units in a bundle may be different. As defined in [EBMS3CORE], these time stamps have an XML schema dateTime type and MUST be expressed as UTC. If the SOAP message is protected using WS-Security, and the `wsse:Security` header contains a `wsu:Timestamp/wsu:Created` element, then that time stamp MUST be more recent than any of the `eb3:MessageInfo/eb3:Timestamp` values at it reflects the time the SOAP message containing all message units is serialized for transmission.

Message bundling applies at the level of individual message exchanges. User message units may be one way exchanges, or the request or response messages in a two way exchange. A signal message may pull or be a response to a user message (which, in a two way exchange, can be the business request or a business response). As an example, subject to the bundling control mechanism of 5.3 a one way user message unit from MSH A to MSH B can be bundled with:

- other one way user message units for MSH B.
- request user message units from A to B that are the first half of a two way message exchange.
- response message units to B related to earlier messages from B to A.
- a pull request signal message on a particular MPC on B.

In the ebMS 3.0 Core Specifications, cross-references exist at the level of message units, and bundling does not change this. Each message unit has its own `eb3:MessageId`, and can refer to another user message unit using `eb3:RefToMessageId`. Similarly, error messages can reference individual message units using the `eb3:Error/@refToMessageInError` attribute. A bundle has no identity and cannot be referenced.

5.2.2 Processing Bundles

A message bundle is processed based on the processing mode of the primary message unit for the following processing mode sections: **Pmode[1].Protocol**, **Pmode[1].ErrorHandling**, **Pmode[1].Reliability**, and **Pmode[1].Security** with the exception of parameters for non-repudiation of receipt. This means that:

- The message security parameters of the primary message unit determine the way the entire message bundle is processed. If the Pmode of this unit is configured to sign ebMS SOAP headers, body and any attached payloads, then the signature MUST also cover any header content and payload from the other message units. The SOAP message has a single `wsse:Security` header that contains a single `ds:Signature` for the SOAP message, not separate `ds:Signature` elements for each message unit. The certificate(s) used will be the certificate(s) specified for the primary unit. If the Pmode is configured to encrypt the message payloads, all message payloads related to all message units in the bundle will be encrypted using the configuration (algorithms, certificates) specified for the primary unit.
- Non-repudiation of receipt is handled at the ebMS message unit level. For each user message that requires a receipt, a separate receipt signal is generated, configured via the **Pmode[1].Security.SendReceipt** and **Pmode[1].Security.SendReceipt.ReplyPattern** parameters of the received message unit. When the receipt is expressed as an `ebbp:NonRepudiationInformation` element, the `ds:Reference` elements in the `ebbp:MessagePartNRInformation` elements MUST cover all elements in the received message that relate to the message unit. The receipt MAY cover elements related to other message units. In particular, a reference to the `eb3:Messaging` container (using its `id` attribute value) covers all content in that container, not just content related to the unit the receipt is generated for. For non-repudiation of receipt, the receiving MSH MUST sign the receipt using the configuration (algorithms, certificates) specified for the received unit.
- The reliable messaging parameters of the primary unit will apply to the entire bundle. Acknowledgments and any required resending of messages apply to the entire bundle. The entire bundle counts as a single message in a single sequence. When InOrder delivery is specified for the primary unit, the bundle must be sent using InOrder reliable messaging and bundles MUST only include sub-series of message units without leaving gaps. I.e. when message units 1-10 are submitted, it is possible to bundle 1-4 and 5-10 in two bundles, but not to bundle 1, 2 and 4 in the first bundle and 3, 5-10 in the second bundle.

- Transport configuration such as the protocol, the URL or IP address of the destination or (in a multi-hop context) the next MSH, and any transport security to be applied is based on the configuration of the primary unit.

- The message will be packaged as a SOAP 1.1 or 1.2 message based on the configuration of the primary unit Pmode.

Bundling is an operation in the ebMS module of an MSH. As defined in the ebMS Core Specification, this module operates before the reliability and security module on the sender side. By definition, the message (the bundle) must be complete on the sender side before applying any reliability and security operations. Even if the primary message unit is known from the start, no signing is done until the message is complete. In the WSI Reliable Secure Profile [WSIRSP10], a signature protects all the SOAP Body, any contained payloads, and several SOAP headers. All message unit headers are signed together.

5.3 Controlling Bundling

Message unit bundling is configured using a number of parameters. These control which units can be bundled together, which message unit in a bundle can act as primary message unit, constraints on the resulting message unit bundle and on the timing between submitting and sending.

Pmode[].bundling.policy

This parameter indicates whether the message unit, respectively, MUST NOT (value *never*), MAY (value *optional*) or MUST (value *always*) be sent in a bundle with other message units. The default value is *never*, which means that the only allowed types of bundling are those defined in [EBMS3CORE]. If the value is *optional* or *always*, the parameter **Pmode[].bundling.maxdelay** MUST NOT have the value *0*.

Pmode[].bundling.compatibility.pmodelist

For message units that have a value other than *never* for **Pmode[].bundling.policy**, this parameter indicates which other message units (acting as primary message units) the message unit may be bundled with. The message units that have this parameter specified MAY become secondary units in bundles that have a message unit with the referenced unit as primary unit. These referenced message units MUST NOT have a value *always* for **Pmode[].bundling.policy**.

Pmode[].bundling.maxsize

This parameter defines the maximum size of a message resulting from bundling. It includes the size of the SOAP envelope and any payloads in MIME attachments. If the size of the single primary message unit is bigger than this parameter, then this means that no other secondary message may be bundled with this unit.

Pmode[].bundling.maxdelay

This parameter (of type duration) defines the maximum time between the oldest message unit and the newest message unit in a bundle, based on the `eb3:MessageInfo/eb3:Timestamp` values of the message units. A receiving MSH SHOULD validate that the bundles satisfies this requirement. A sending MSH MAY use this parameter to determine, for a submitted message unit, to bundle it with other submitted messages , to send it by itself (as an unbundled) message or to defer sending it (as other message units may be submitted later that it could be bundled with).

5.4 Submitting and Bundling

Section 2.1.1 of [EBMS3CORE] describes the ebMS messaging model. In this model, a Message Producer invokes the *submit* abstract operation of a Sending MSH, which instructs that MSH to *send* the message. Section 2.1.2 and 2.1.4 clarify that the submit operation is to be understood at the level of user message units. The bundling protocol can be viewed as an extension of this model in one of the following two ways:

- The *submit* operation of MSH is extended to allow the Producer to submit not just the data needed to generate a single message unit (which will become the primary message unit), but to also provide a (possibly empty) lists of such data structures, which are used to generate a message bundle containing the primary and any secondary message units. In this interpretation, the Producer must be aware of the constraints of the parameters defined in section 5.3 and the Sending MSH must validate that the submitted bundle complies with its configuration. If this is not the case, the EBMS:0030 bundling error (see 5.5) is generated in the Sending MSH.

- The *submit* operation is not modified. Instead, the Sending MSH determines (subject to the control parameters described in section 5.3) whether to send a submitted user message unit as an unbundled message, to bundle it as secondary message unit to an available primary unit, to designate it as a primary unit that other (separately submitted) message units can be bundled with, or to defer any decision on bundling for some specified time.

An implementation of this protocol MAY support either or both approaches. For interoperability with other implementations, the only requirement is that the agreed values for the bundling parameters of section 5.3 are correctly applied in the exchanged message bundle.

In the second situation, where bundling is an MSH-internal operation, more optimizations are possible. This is because multiple business applications may submit units of different types (e.g. one application submits invoices and the other updates catalogs) that have the same destination and can be bundled. Another advantage is that the bundling logic is not duplicated in the various business applications.

In situations where there is more than one option for a unit (send unbundled, send as primary unit with other secondary units attached, send as secondary), the behavior of the MSH is not specified. An implementation MAY take factors like system or network load, number of pending message units, connectivity to the business partner into consideration. An event (see section 5.7 for discussion) that MAY cause a bundle to be formed and sent is an incoming pull request.

5.5 Errors in Bundling

An ebMS message bundle consists of multiple message units. Errors may occur in the processing of any of these messages. For any message unit in the bundle that has an error, an `eb3:SignalMessage` is generated that contains one or multiple `eb3:Error` elements. The MSH SHOULD bundle these signal messages and use the **Pmode[1].ErrorHandling.Report.{SenderErrorsTo, ReceiverErrorsTo, AsResponse}** processing modes parameter of the primary message unit to determine the address to send this bundle to.

The impact of errors in message units on the processing of other message units in a bundle is subject to configuration and is discussed in section 5.10.3 . This specification defines a new error type related to the actual bundling.

- Error ID: EBMS:0030
- Short description: BundlingError
- Severity: Failure
- Description: a message bundle is being processed that has a structure that is not compatible with the bundling processing mode parameters of one or more user message units in the bundle.

While this error concerns the bundling, the bundle as such has no identity and therefore the signal message containing the error MUST reference the primary message unit as the unit that represents the bundle.

A receiving MSH SHOULD generate this error in the following situations:

- A bundle is received where the oldest message is more than **Pmode[].bundling.maxdelay** older than the newest message.
- A secondary message is present that does not have the primary message unit listed in its **Pmode[].bundling.compatibility.pmodelist** list.
- The message is a bundle and the overall size of the message is greater than **Pmode[].bundling.maxsize** of the primary message unit.

A sending MSH MUST generate this error in the following situation:

- A message unit with value *always* for **Pmode[].bundling.policy** is submitted at time *t*, but no message unit in its **Pmode.compatibility** configuration is submitted before *t*+**Pmode[].bundling.maxdelay**.

Section 5.10.3 discusses delivery policies for bundles containing units that have errors and introduces a second new type of error. An ebMS Intermediary is to forward message bundles transparently and MUST NOT generate any of these errors.

5.6 Bundling Best Practices

The parameters in section 5.3 allow business partners to control which message units are bundled. In practice, some combinations of units will be more common or acceptable than others. This non-normative subsection defines some best practices regarding bundling configuration.

5.6.1 Compatibility

First of all, **Pmode[].bundling.compatibility** is typically reflexive: multiple message units of the same type can normally be bundled as the sender would process them in the same way and any intermediaries in an I-Clode would route them (assuming no changes to the routing function) to the same ultimate destination. However, a potential exception to this are cases where a message unit of some type transmits very large payloads. In that situation it may be acceptable to bundle the unit other small units, but the message would become too big if a second unit of the same type were to be added.   (See section 6 and 12.1.3 for ebMS 3.0 support for handling large messages).

For different message unit types, the **Pmode[].bundling.compatibility** relation is generally asymmetric: it is acceptable (but unnecessary) to apply “stronger” quality of service to a message unit if it becomes part of a bundle that has units that require it. The reverse is not. This “stronger”/“looser” relation depends on the exact values of processing mode parameters, or combinations of such values. For instance, a bundle secured using a particular configuration of hashing or signing algorithms, using other certificates, or a different key length may, in combination, provide the same or better protection than some other combination of values for these parameters.

5.6.2 Security Compatibility

A bundled ebMS message uses a single `wsse:Security` header targeted to the ultimate ebMS receiver, if security is specified in the processing mode of the primary message unit. This security header contains a single `ds:Signature` element that has `ds:Reference` elements for all payload parts, if payload signing is specified. Any payload parts brought in via secondary message units will be treated in the same way as parts that came with the primary message unit.

In ebMS, non-repudiation of receipt is handled via receipt signals and handled at the level of message units. These signals are generated for each user message that requests them. A receiving MSH MAY construct Receipt signals using the `ds:Signature` element in the received message, and the `ds:Reference` message parts identifiers and digest values (verified by the security module). In that situation, each receipt references all signed message parts in a bundle, not just parts that relate to a specific user message. A user message that requires a signed Receipt in its **Pmode[].bundling.compatibility.pmodelist** SHOULD NOT reference any candidate primary user message that does not require message signing.

5.6.3 Reliability Compatibility

An ebMS message bundle is a single reliable messaging (RM) entity. All bundled message units are subject to the same delivery assurance (either AtLeastOnce, AtMostOnce, or ExactlyOnce, InOrder). Consequently the same RM headers apply to all units, as well as the same RM sequence and message number. All these message units MUST then be subject to the same reliability QoS. The delivery assurance specified for the primary message unit SHOULD be similar or better than the assurance of any secondary message units.

Attention must be paid to message units that require InOrder delivery assurance. These

- MUST NOT be bundled with a primary primary message unit that has a value "`undefined`" for the processing parameter **Pmode[].bundling.ordering.policy.** This is because this value means the submission order is not preserved in the bundling process.
- MUST NOT be bundled with primary message units that have an incompatible value for **Pmode[].reliability.correlation**. This is needed to make sure that the units are sent using the same sequence/group as would have been applied when sent using their own Pmode.

5.7 Bundling and Pull

The description of bundling in section 5.2.2 allows a sending MSH to queue submitted user messages temporarily in the ebMS module, bundle them after a configurable interval with other units and then process the bundle in the security and reliability processing modules, like unbundled ebMS messages. When receiving a Pull Request on a particular MPC, an MSH SHOULD NOT deliver an EBMS:0006 warning ("EmptyMessagePartitionChannel") in situations where a unit has been submitted on that MPC which allows for bundling with other units, but which has not yet been bundled. An MSH MAY use any mechanism that has the effect that an ebMS message is available for pulling on an MPC in situations where at least one message unit has been assigned to that MPC. Such mechanisms MAY include (but are not limited to) not bundling such units, "on the fly" bundling or even "re-bundling" (optimizing already bundled units that have not yet been pulled into more efficient bundles).

5.8 Bundling and Responses

An ebMS message may result in various types of response messages: business responses, ebMS receipts, ebMS errors and reliable messaging acknowledgments.

For two way message exchanges, a "sync" channel binding is typically used for situations where the response business message is needed immediately, and can be produced within the short duration of an HTTP connection. It is RECOMMENDED that **Pmode[].bundling.policy** is set to *never* for request message units in Two Way exchanges that have a "sync" channel binding.

When processing a message bundle, the receiving MSH MUST look up the processing mode parameter setting of the primary message unit in the bundle and determine for each of these response message types what channel they are to be sent back on.   In some configurations, response message units are to be sent on the HTTP back channel. The receiving MSH MUST collect all these synchronous response messages and MUST send a message bundle containing all of them on the HTTP back channel. (This means that if a secondary message had specified a different channel than the HTTP back channel, this value is ignored. The configuration of the primary message unit determines the back channel for all message units). If the address of the response message is not the HTTP back channel, then bundling is still allowed, but not required.

5.9 Bundling for Multi-hop

The concept of transparent intermediaries defined in chapter 2 supports end-to-end security and requires messages to be forwarded without any modification. This requirement means that intermediaries MUST NOT bundle or un-bundle messages.

In a multi-hop environment, intermediaries MUST process a message bundle as a regular ebMS message. This means that:

- If a routing parameter needs to be generated (for instance to route a WS-ReliableMessaging life cycle message to its intended destination), its content is determined based on the primary message unit.
- If no routing parameter is present, the primary user message unit determines the way the message is processed.

This means that there is no ambiguity about which information is to be used for routing decisions. Business partners SHOULD set up the **Pmode[].bundling.compatibility** in a way that makes sure that message units are only bundled with other primary message units that will be routed to the same ultimate destination. This assumes some knowledge of the routing functions in the I-Cloud. If the configuration of these functions is unknown or subject to unanticipated changes, the safe option is to only allow bundling of message units of the same type.

5.10 Delivery Policies

When business partners exchange message units that are logically related, there often is a requirement for those units to be delivered and processed in order. For instance, a first user message unit may make a request that a subsequent user message cancels. The cancellation user message unit must not be processed before the request message unit as it references an as yet unknown request and will therefore most likely cause a processing error, while the request is not canceled. When an error occurs in the processing of message unit in a series, it may not be possible or desirable to process subsequent related messages.

When related messages units are sent in separate SOAP messages, the message ordering capabilities of Web Services reliable messaging specifications [WSRM11] or [WSR11] can be used to specify the order in which they are delivered, even if for some reason the later message unit is received before the earlier message unit. When using message bundling, it is possible that the two message units are sent as part of the same bundle, in situations where the second message unit is submitted shortly after the first one. The reliable messaging In-Order delivery contract does not help here, as it applies to the message bundle as a whole, not to the individual message units.

Sender and receiver are able to agree on policies that affect the delivery of message units in a bundle. These policies can be set using the processing mode parameters defined in section 5.10.1 and help account for dependencies among message units. These parameters determine the order of delivery (described in section 5.10.2) and the behavior in case of errors (see 5.10.3).

5.10.1 Delivery Parameters

Delivery is controlled using the parameters **Pmode[].bundling.ordering.policy** and **Pmode[].bundling.ordering.scope**.

The parameter **Pmode[].bundling.ordering.policy** specifies the semantics of the order of message units in a bundle as related to the order of submission or the order in the `eb3:Messaging` container. It has three values:

- `undefined`: the order of message units in the `eb3:Messaging` container has no meaning. This value means that InOrder delivery MUST NOT be specified for any secondary message unit in the bundle.
- `documentorder`: the document order ([XDM], section 2.4) of the message units in the `eb3:Messaging` container MAY reflect the order of submission and MUST be interpreted as the required order of delivery of these units. This means that the primary message unit is the first submitted message and that any secondary message units have been submitted subsequently, appended to the list of child elements in the `eb3:Messaging` container.
- `timestamp`. This indicates that the sending MSH units MAY insert units in any order and that the primary message therefore is not necessarily the oldest message in the container. However, timestamps on message units MUST reflect their order of submission.

The default value is `undefined`. This processing mode parameter can be used to control the order in which message units in a bundle are to be delivered (section 5.10.2) and whether units are delivered in case of errors (section 5.10.3).

When using a value other than `undefined`, the Pmode configuration SHOULD also specify the use of InOrder delivery assurance at the reliable messaging level, to make sure that all message units in a bundle are processed before all message units in a later bundle. A message unit U1 intended to be processed before a unit U2 MUST either be:

- Inserted in a bundle, or packaged as a standalone message, and sent before the message (or message bundle) containing U2, both using an InOrder delivery assurance; or:

- Inserted in the same bundle as U2, but preceding it in document order (document order ordering policy) or submitted before U2 and bearing an time stamp older than the time stamp for U2 (time stamp ordering policy), also both using an InOrder delivery assurance.

The **Pmode[].bundling.ordering.scope** parameter MUST ONLY be specified when the value for **Pmode[].bundling.ordering.policy** is other than `undefined`. Its value is a list of XPath expressions that are applied to the message units in a bundle to select subsets of message units at which ordering is defined. For example:

- A value of {`eb3:CollaborationInfo/eb3:ConversationId`} indicates that conversations are separate and can be processed independently.
- A value of {`eb3:PartyInfo/eb3:From, eb3:PartyInfo/eb3:To`} indicates that the scope of ordering is limited to sender/receiver party ID pairs.

If both **Pmode[].reliability.correlation** and **Pmode[].bundling.ordering.scope** are specified for a Pmode, their value MUST be identicial.

5.10.2 Processing Order of Message Units

In bundling multiple message units, message units in the `eb3:Messaging` container may not be arranged in the order in which they were submitted. In fact, a secondary message unit may have been submitted to the MSH before the primary message unit, but inserted following it in the `eb3:Messaging` container. The values of **Pmode[].bundling.ordering.policy** can be used to control delivery of units in a bundle:

- A value `undefined` means that the order of message units has no defined meaning. A receiving MSH MAY process and deliver in any order. It may even deliver them in parallel.
- A value of `documentorder` means that a receiving MSH MUST process and deliver message units in the order in the `eb3:Messaging` container.
- A value of `timestamp` means that a receiving MSH MUST process and deliver message units in the temporal order in the `eb3:Messaging` container. The receiving MSH MUST sort the message units based on the value of the `eb3:Timestamp` in `eb3:MessageInfo` and deliver the message units in temporal order, older units first. Any message units having the exact same time stamp value MUST be delivered in the order in which they occur in the `eb3:Messaging` container.

The **Pmode[].bundling.ordering.scope** parameter, if specified, allows the receiving MSH to partition the message units into subsets before processing them in the order specified by **Pmode[].bundling.ordering.policy**. The MSH MAY process each partition in parallel.

5.10.3 Processing Bundles with Errors

An MSH may encounter errors in the processing of one or more message units in a bundle. These errors may also be relevant to some other message units in the bundle, and to subsequent messages sent in other bundles or as standalone messages, but not to other messages units that are not related to any message unit in error. The **Pmode[].bundling.ordering.policy** and **Pmode[].bundling.ordering.scope** parameters allow an MSH to determine when such a dependency exists.

- The value `undefined` for **Pmode[].bundling.ordering.policy** indicates that there is no dependency between message units. An error in one message unit has no impact on any other message unit.
- The values `documentorder` and `timestamp for` **Pmode[].bundling.ordering.policy** indicate a dependency among message units. An MSH MUST NOT process subsequent message units when it encounters an error in a preceding (for `documentorder`) or older (for `timestamp`) message unit.

- The parameter **Pmode[].bundling.ordering.scope** allows the MSH to partition the message units into unrelated groups. Processing of one message unit partition, and any errors encountered in this, MUST NOT impact the processing of other partitions.

These requirements only affect the processing of units in a single bundle. They do not impact the processing of subsequent units sent in other bundles or as standalone messages.

To indicate that a message is not processed because of an error involving another related message unit in a bundle, an MSH SHOULD generate the following warning:

- Error ID: EBMS:0031
- Short Description: RelatedMessageFailed
- Severity: Failure
- Description: A message unit was not processed because a related message unit failed.

A separate error MUST be generated for that unit in error that indicates the nature of the error of that unit.

As an example, assume a bundle is sent containing 5 message units. The primary unit specifies the following Pmode parameters:

- `{eb3:CollaborationInfo/eb3:ConversationId`} as value for **Pmode[].bundling.ordering.scope**.
- `True` for **Pmode[].Security.SendReceipt.**
- `Response` for **Pmode[].Security.SendReceipt.ReplyPattern**.
- `True` for **Pmode[].ErrorHandling.Report.AsResponse**.
- `documentorder` for **Pmode[].bundling.ordering.policy.**
- An `http://..` address for **Pmode[].Protocol.Addres.**

Units 1, 3 and 4 are part of a conversation C1 and units 2 and 5 of conversation C2. The MSH successfully processes units 1 and 2, but encounters an error in processing unit 3. Since unit 4 is in the same partition as unit 3, it will not be processed. Unit 5 is processed successfully. A response message is generated and sent on the HTTP back-channel that consists of a bundle containing Receipts for messages 1, 2 and 5 and Errors for messages 3 and 4. The error for message 4 is of type EBMS:0031.

6 Large Message Splitting and Joining

6.1 Background and Context

In many industry sectors and communities in the public sector there is an ever-increasing need to exchange large data sets or files, with sizes ranging from hundreds of megabytes to gigabytes. Examples of this include:

- Large or very large transaction data sets, such as billing data in telecom industry (exchanged between MVNO and infrastructure providers, roaming partners or telecom service providers and corporate customers), or payment order sets in the financial services sector.
- Data sets from event monitoring applications.
- Data sent to off-site data warehouses for analysis.
- Backups and electronic archives.
- Image and other multimedia data in healthcare, media and other sectors.
- Geo-spatial data.
- Computer-Aided Design files in automotive, industrial goods, aerospace and defense industries.

The following problems may occur in large message exchanges:

- Transmitting large messages takes time. If the transfer fails and is restarted from scratch, a lot of data may have to be resent that was already transmitted.
- Network or software components may impose limits on message size and connection timeouts and may cause transfers to be aborted.
- Section 2.5.2 defines a store-and-forward model for intermediaries. When applied to large messages, the delays in transfer of these messages may be unacceptable.

This specification defines multiple new mechanisms to transfer large ebMS SOAP messages that extend the ebMS 3.0 Core Specification and are compatible with the ebMS bundling and intermediary features:

- Transport restart. This is provided for the HTTP protocol and the “push” MEP binding using the AS2 restart feature described in section 12.1.3 . This feature does not support “pull” messaging. It is well-suited to point-to-point, push messaging. It can also be used in multi-hop messaging to transmit (push using HTTP) large messages from an MSH to the next MSH.
- Transfer of a large message by splitting the message and transmitting it as a set of smaller SOAP messages. This is described in this chapter. This feature supports both push and pull messaging, and limits transfer delays when using store-and-forward intermediaries.

6.2 Message Splitting and Joining

This chapter specifies a protocol for transmitting a large SOAP-rooted, MIME `Multipart/Related` message as a set of SOAP message fragments. The protocol composes with other SOAP protocols for security, addressing, reliable messaging and with ebMS. An implementation of the protocol operates as a discrete module in a SOAP processing pipeline. The protocol defines how and when a sending MSH splits a message in fragments and how a receiving MSH reassembles the message from its fragments in an interoperable way. The protocol is similar to some HTTP 1.1 features, in particular to chunked transfer coding and compressed content coding, but is applied at the SOAP message level. It supports asynchronous messaging in multi-hop connections and supports both SOAP-with-attachments [SOAPATTACH]and Message Transmission Optimization Mechanism [MTOM].

While the protocol can be used with other Web Services profiles, the main goal is to support transfer of large ebMS messages. Its main benefits in that context are:

- Support for both push and pull message exchange (the AS2 restart feature of 12.1.3 is limited to push exchanges);
- Support for HTTP and SMTP transport protocols.
- Reliable messaging is leveraged to transmit (and, if needed, retransmit or eliminate duplicates of) fragments;
- The upper bound of the delay caused by a store-and-forward intermediary and the temporary storage space needed by such an intermediary are proportional to the fragment size, rather than to the message size.
- Fragments can be routed individually through a multi-hop network. Fragments may follow different paths from sender to recipient. If one intermediary fails and traffic is re-routed via an alternative route, only the undelivered fragments need to be resent.
- The protocol composes with the ebMS message bundling feature of chapter 5 .
- An optional compression option is provided.

6.2.1 Splitting/Joining Protocol

The splitting/joining protocol is an optional protocol that is intended to be used in a SOAP processing pipeline with other SOAP modules. The protocol consists of two related operations:

- As an extension to SOAP message *sending*, a message splitting operation splits a large message into multiple fragments, wrapping each fragment in a separate SOAP message with a `MessageFragment` header extension element. These messages are further processed individually in the SOAP pipeline, causing the sending MSH to emit multiple SOAP output messages for a single submitted message.
- An an extension to SOAP message *receiving*, presence of a `MessageFragment` header element is an indication that the content of the message is part of a larger message. Processing of the SOAP message is halted, until all related fragments are received. The fragmented message parts are re-assembled and joined into a single larger message. The receiving MSH then continues to process this larger message.

We will refer to the input to splitting (and the output of joining) as the “source message”, and to the output of splitting (and the input of joining) as the “result fragments”.

The following diagram illustrates the layering of SOAP services. Some services are higher-level than the splitting/joining protocol and some are lower-level services. Within a processing pipeline, the splitting protocol can be positioned relative to other SOAP protocols depending on specific requirements. For instance, if WS-Security is used as a higher level service to sign and encrypt messages, signatures of incoming messages can only be validated when all fragments are received and the larger message is reassembled. If WS-Security is used as the lower level service, each fragment can be processed as it comes in, but the number of security operations to perform increases.

When used in combination with higher level SOAP modules, these modules may require additional binding or configuration, such as adding specific SOAP headers, in addition to the `MessageFragment` header. Section 6.4.1 specifies such additional requirements for WS-Addressing, and section 6.4.2 specifies a default binding for ebXML Messaging version 3.0. Other Web Services protocols or profiles could similarly define such requirements as bindings.

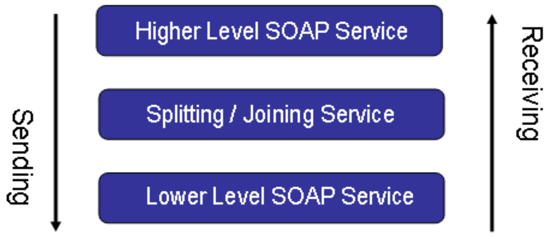

Figure 15: Splitting and joining stages in a SOAP pipeline

6.2.2 MessageFragment

Figure 16: MessageFragmentType shows the structure of the `MessageFragment` schema type definition.

The required attribute `href` identifies the MIME part containing the message fragment.

The attribute groups `S11atts` and `S12atts` define the SOAP 1.1 and 1.2 attributes. These express how SOAP processors process `MessageFragment` structures.

The following elements are in `MessageFragment`:

- `GroupId` identifies a set of related messages and specifies that these messages are fragments derived from the same single source message. The value of `GroupId` MUST be a globally unique string.
- The optional `MessageSize` element indicates the size (in bytes) of the message after reassembly. An MSH can use this to compute status information about a message that is being received, such as the percentage of data that has been received, and the amount of storage space needed to receive the message.
- `FragmentCount` indicates the number of fragments that a message has been split into. A receiving MSH can use this to determine if it has received all fragments in a group. At least one fragment must specify fragment count. This supports an implementation model where the splitter function works in streaming mode or at the end of a pipeline, and the number of fragments is not known until the last part of the message is processed.
- `FragmentNum` identifies a single fragment within a group. Its value is an integer, 1 ≤ `FragmentNum` ≤ `FragmentCount`.
- `MessageHeader` is a container with information related to the envelope of the source message. Its content is described in section 6.2.3 .
- `Action` is the value of the SOAP 1.1 `SOAPAction` header or the SOAP 1.2 `action` parameter.
- The optional `CompressionAlgorithm` and `UncompressedMessageSize` elements are defined in section 6.2.4 .

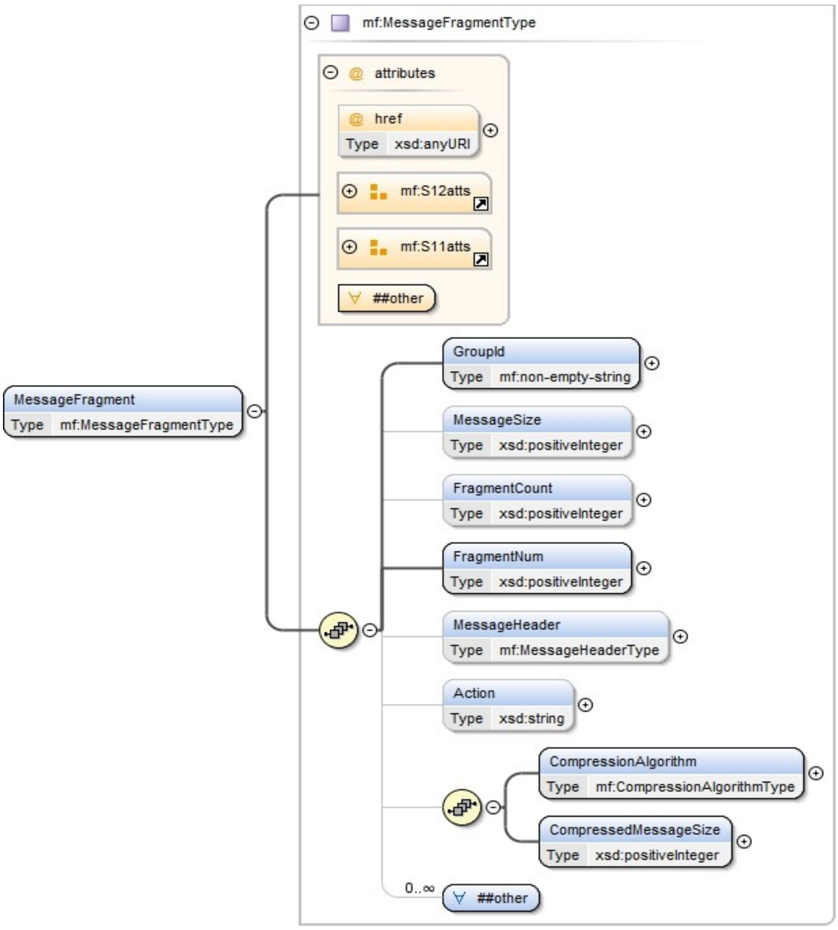

Figure 16: MessageFragmentType

6.2.3 Binding for Envelope Formats

The content of the `MessageHeader` element is displayed in Figure 17: Message Header. It contains the information from the source message header that needs to be preserved in order to correctly reassemble the MIME enveloped SOAP message in the receiving MSH.

The following elements are contained in `MessageHeader`:

- `Content-Type`: the content type of the source message envelope. The fixed value of this optional element is `Multipart/Related`.
- `Boundary`: the boundary separating the MIME parts in the source message envelope.
- `Type`: the value of the *type* parameter.
- `Start`: the value of the *start* parameter. This identifies the first MIME part in the MIME envelope that contains the SOAP envelope of the source message, without the “<” and “>” delimiters.
- `StartInfo`: the value of the *startinfo* parameter as used in MTOM.
- `Content-Description`: the optional description of the MIME envelope.

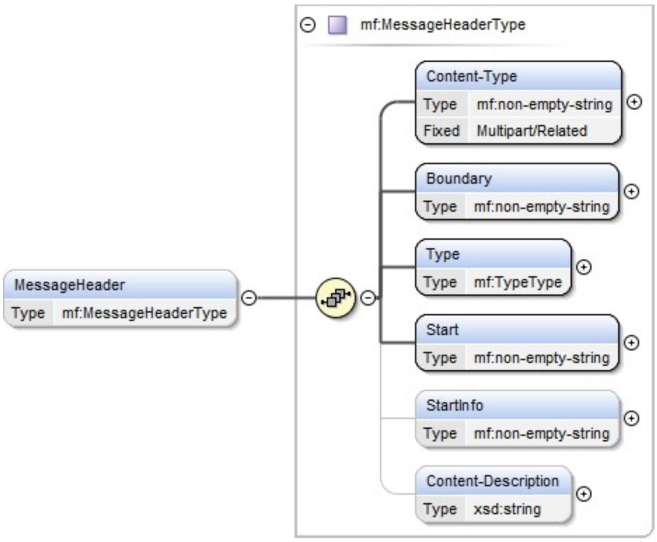

Figure 17: Message Header

To explain the semantics of the splitting and joining operations, we describe these operations as format transformations. The splitting operation in the sending MSH can be viewed as an operation on a MIME envelope that extracts values from the MIME header to build the `MessageFragment` element. The joining operation in the receiving MSH can be viewed as an operation that constructs a MIME envelope from a `MessageFragment` element. Note that SOAP processors may use optimized internal data structures for messages and may not serialize messages that are exchanged between SOAP modules for efficiency. This specification does not assume a particular processing or internal data structures and the transformation analogy does not require implementations to use MIME envelopes as interface formats.

As an example, we use the following example from the ebMS 3.0 Core Specification:

```
SOAPAction: leasing
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
        start="<car-data@cars.example.com>"

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <car-data@cars.example.com>

<?xml version='1.0' ?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/">
  <S11:Header>
     <eb:Messaging S11:mustUnderstand="1">
         …
           <eb:PayloadInfo>
              <eb:PartInfo href="cid:car-photo@cars.example.com" />
              <eb:PartInfo href="#carData" />
           </eb:PayloadInfo>
     </eb:Messaging>
  </S11:Header>

  <S11:Body>
     <t:Data id="carData" xmlns:t="http://cars.example.com">
        <t:Mileage>20000</t:Mileage>
        <t:OwnerPicture href="cid:picture-of-owner@cars.example.com"/>
     </t:Data>
  </S11:Body>
</S11:Envelope>

--MIME_boundary
```

```
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <car-photo@cars.example.com>

...binary TIFF image of the car...

--MIME_boundary—
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <picture-of-owner@cars.example.com>

...binary TIFF image of the car's owner...

--MIME_boundary—

```

Assuming the message will be split in two fragments, the first `MessageFragment` could look like the following:

```
        <mf:MessageFragment
            href="cid:3ec80b70-c6d0-4e39-89ee-
d45da8b82859@cars.example.com">
            <mf:GroupId>fa053eaf-47ed-4862-b70b-e5acbd94061a</mf:GroupId>
            <mf:FragmentCount>2</mf:FragmentCount>
            <mf:FragmentNum>1</mf:FragmentNum>
            <mf:MessageHeader>
                <mf:Content-Type>Multipart/Related</mf:Content-Type>
                <mf:Boundary>MIME_boundary</mf:Boundary>
                <mf:Type>text/xml</mf:Type>
                <mf:Start>car-data@cars.example.com</mf:Start>
            </MessageHeader>
            <mf:Action>leasing</mf:Action>
        </mf:MessageFragment>

```

6.2.4 Compression Feature

This protocol provides optional support for MIME envelope compression. Compression is applied to the source message (prior to splitting) and decompression is applied to the concatenated content of the assembled fragment messages (after joining). Therefore either all fragment messages contain compressed data or none.

- Use of compression is specified by presence of a `CompressionAlgorithm` element with value of type `CompressionAlgorithmType`. The values `gzip, compress, deflate and identity` MUST be interpreted as encoding transformations as defined in the section 3.5 of [HTTP11]. The supported compression algorithm is an agreement feature (section 6.5).
- The element `CompressedMessageSize` specifies the size (in bytes) of the compressed SOAP message.

The following summarizes the differences between the compression feature and other mechanisms to reduce file size:

- SOAP-with-attachments and Message Transmission Optimization Mechanism (MTOM) optimize transfer of `base64Binary` data as binary attachments when using the HTTP protocol. The compression feature applies to the entire MIME envelope, including SOAP and MIME part headers and any text or binary payload content.
- AS4 compression applies to an individual payload. The compression feature may be more effective in case of a message containing multiple payloads.
- The splitting/joining protocol requires each fragment to carry a SOAP header that contains the `MessageFragment` element and possibly additional SOAP headers. This increases the amount of data to be transmitted.

6.3 Behavior of the MSH

This section specifies the behavior of the sending MSH and the receiving MSH.

6.3.1 Behavior of the Sending MSH

The sending MSH operates the splitting function. Splitting is as an operation that:

- Uniquely identifies the group of related message fragments and sets the `GroupId` element accordingly. (Profiles MAY specify how the identification is done; section 6.4.2 describes this for ebMS 3.0).
- Removes the MIME header from the source message and extracts the relevant metadata needed to be able to construct the `MessageHeader` header and `Action` elements as described in section 6.2.3 .
- Optionally applies compression as specified in section 6.2.4 and 6.5 and sets the `CompressionAlgorithm` and `CompressedMessageSize` elements accordingly.
- Splits the remaining message in contiguous parts. Note that this is a splitting operation that considers the MIME envelope as a byte stream. It is unaware of the MIME structure of the source message. A single MIME part from the source message may start in one fragment, continue in another and end in a third part. A fragment may contain a complete MIME part, part of one, the end of one part and the beginning of the next, etc.
- Builds new SOAP messages, one for each fragment.
  - If the source message is a SOAP with attachment message, the SOAP message containing the fragment will also a SOAP with attachments message. An MTOM source message will result in an MTOM fragment message. The SOAP envelope versions in the source message and the SOAP envelope for each of the fragments MUST also match, so a SOAP 1.2 source message results in SOAP 1.2 SOAP fragment messages.
  - The `Content-Type` of the MIME part containing the SOAP envelope will be `text/xml` for SOAP with attachments messages and `application/xop+xml` for MTOM messages.
  - The MIME boundary and `Content-Id` of the SOAP root part in the SOAP message of the fragment MUST be unique and MUST differ from the corresponding values in the source message. The range of data from the source message that is included in the fragment MUST be inserted as a single MIME part, immediately following the part containing the SOAP envelope.
  - The `Content-Type` of the data part MUST be set to `application/octet-stream`.
  - The MSH MUST NOT insert any MIME part other than these two MIME parts.
  - The root SOAP Body element MUST have empty content.
  - The SOAP envelope MUST contain a `MessageFragment` header. If the SOAP message is split into just one fragment (i.e. is not really split at all), the values for `FragmentCount` and `FragmentNum` are both 1. If there are two fragments, `FragmentCount` is 2 and one fragment message has `FragmentNum` at 1 and the other at 2, and so on.
  - The MSH MUST provide a value for the `href` attribute of the `MessageFragment` in the SOAP envelop that identifies the data part using its content identifier.
  - Additional SOAP headers MAY be added, for instance to support message routing. This higher-level functionality is dependent on the other SOAP modules, such as WS-Addressing or ebMS (see sections 6.4.1 and 6.4.2).
- Continue, for each of these generated SOAP messages, processing them using the lower level SOAP modules.

As an example, here is a hypothetical generated first MIME part for the ebMS example message. It contains the first part of the source message. The entire SOAP envelope, and the beginning of the first TIFF image fit in the first fragment. Note that the MIME boundary of the source message is just text data in the fragment.

```
SOAPAction: leasing
Content-Type: Multipart/Related; boundary=37b648b1-44af-459f-a533-
bbf8ed6a93be; type=text/xml; start="<f19aff62-8983-47f5-bc36-
514741203fe2@cars.example.com>"

--37b648b1-44af-459f-a533-bbf8ed6a93be
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <f19aff62-8983-47f5-bc36-514741203fe2@cars.example.com>

<?xml version='1.0' ?>
<S11:Envelope
     xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:mf="http://docs.oasis-open.org/ebxml-msg/ns/v3.0/mf/2010/04/">
  <S11:Header>
        <mf:MessageFragment
            href="cid:3ec80b70-c6d0-4e39-89ee-
d45da8b82859@cars.example.com">
            <mf:GroupId>fa053eaf-47ed-4862-b70b-e5acbd94061a</mf:GroupId>
            <mf:FragmentCount>2</mf:FragmentCount>
            <mf:FragmentNum>1</mf:FragmentNum>
            <mf:MessageHeader>
                <mf:Content-Type>Multipart/Related</mf:Content-Type>
                <mf:Boundary>MIME_boundary</mf:Boundary>
                <mf:Type>text/xml</mf:Type>
                <mf:Start>car-data@cars.example.com</mf:Start>
            </mf:MessageHeader>
            <mf:Action>leasing</mf:Action>
        </mf:MessageFragment>
    <!-- OTHER HEADERS OMITTED, SEE SECTION 6.4.2  -->
  </SS:Header>
  <S11:Body />
</S11:Envelope>

--37b648b1-44af-459f-a533-bbf8ed6a93be
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <3ec80b70-c6d0-4e39-89ee-d45da8b82859@cars.example.com>

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <car-data@cars.example.com>

<?xml version='1.0' ?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/">
  <S11:Header>
     <eb:Messaging S11:mustUnderstand="1">
         …
           <eb:PayloadInfo>
              <eb:PartInfo href="cid:car-photo@cars.example.com" />
              <eb:PartInfo href="#carData" />
           </eb:PayloadInfo>
     </eb:Messaging>
  </S11:Header>

  <S11:Body>
     <t:Data id="carData" xmlns:t="http://cars.example.com">
        <t:Mileage>20000</t:Mileage>
        <t:OwnerPicture href="cid:picture-of-owner@cars.example.com"/>
     </t:Data>
  </S11:Body>
</S11:Envelope>

```

```
--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <car-photo@cars.example.com>

... BEGINNING of the data for the binary TIFF image of the car ...

--37b648b1-44af-459f-a533-bbf8ed6a93be
```

6.3.2 Behavior of Receiving MSH

A receiving MSH that receives SOAP message containing a `MessageFragment` element MUST process this as follows:

- Extract the value of `GroupId` and `FragmentNum`. Based on the value of `GroupId`, the MSH may classify the fragment message as the first fragment for a new message transfer, or a fragment in a group for which other fragments were previously received.
    - If the fragment is part of a group for which a previously received fragment was rejected, the fragment message is rejected.
    - If a fragment message with the same values for `GroupId` and `FragmentNum` was previously received, the message is rejected and all data from previously received fragments for the group is deleted.
    - If the `FragmentCount` is specified or known from previously received fragment and `FragmentNum` is greater than it, the message is rejected.
    - Otherwise, the message fragment data is extracted and stored, indexed on `GroupId` and `FragmentNum`. A counter counting the number of received fragments is (initialized at 0 if not yet set and) incremented. The storage in use to hold the fragment data related to the group is (initialized at 0 if not yet set and) incremented.
- If `FragmentCount` is specified then, if no previously received fragment specified a value for it and no previously received fragment for the group had a greater value for `FragmentNum`, the value is recorded for `GroupId`. Otherwise the message is rejected.
- If the size of the MIME data part exceeds the specified maximum, the message is rejected.
- If `CompressedMessageSize` is specified then, if no previously received fragment specified a value for this element, the value is recorded for `GroupId`. Otherwise, the message is rejected and an error generated. Based on the value of the element, the amount of data related to the group already received and persisted and the anticipated need for additional storage, the MSH MAY reserve storage space to store data from the fragment messages that are yet to be received. The MSH MAY be able to determine whether it has sufficient space to receive the complete message and MAY reject the message if this is not the case.
- This is done similarly for `MessageSize`.
- If the fragment message is accepted, the data content MIME part MUST be extracted and persisted. The SOAP header part serves no further purpose and MAY be deleted.
- If the message fragment that is received is the last fragment in a group, the message fragments are reassembled. If the message is compressed, it MUST be decompressed
- The corresponding values of the `eb3:UserMessage` header are compared and MUST match.
- The message is processed as if the message was received as a single large message, with the relevant MIME parameters taken from the `MessageFragment` group.

In all situations where a message is rejected, an error is generated and all data related to the `GroupId` is removed. The MSH MUST record the value for `GroupId` with status rejected and reject any subsequent fragments with this `GroupId`.

For security reasons the receiving MSH MAY identify fragment sets and messages on more values than just the value of `GroupId`. See section 6.7 for discussion.

6.4 Protocol Bindings

Higher-level protocols may require specific additional behaviour of a message splitting / joining processor. This section defines default bindings for WS-Addressing [WSADDRCORE] and ebMS 3.0 [EBMS3CORE].

6.4.1 Binding for WS-Addressing

When using this protocol in combination with WS-Addressing, the following WS-Addressing headers MUST be copied to the SOAP header of the SOAP fragment message. This includes:

- `wsa:To`, `wsa:Action`, `wsa:RelatesTo` headers.
- Any WS-Addressing headers targeted to the http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/nextmsh role.
- Any WS-Addressing headers targeted to other SOAP role that are used for routing purposes.

The `wsa:MessageId` MUST NOT be copied. If the source message contains a `wsa:MessageId`, distinct unique WS-Addressing Message IDs MUST be generated for and inserted in all fragment messages.

6.4.2 Binding for ebXML Messaging

This subsection specifies a default binding of ebMS 3.0 to the message splitting/joining protocol. The following is specified:

- Application of the splitting/joining protocol is controlled by Processing Mode parameters (see section 6.5), not (just) by message size.
- The `MessageFragment` element is targeted to the SOAP ultimate receiver and MUST be understood. Intermediaries do not split messages or join fragments.
- Splitting and Joining is a higher-level SOAP module than ebMS pull authorization; i.e. pull requests MUST be properly authorized, based on Pmode configuration, and return SOAP message fragments.
- When an ebMS message is split and separate fragment messages are sent, then the processing mode parameters for WS-Security and reliable messaging apply at the level of fragments, not at the level of the source message. Message fragments are individually secured and sent reliably. Message acknowledgments, retries and duplicate eliminiation apply at the level of message fragments.
- The content of the `eb3:Messaging` MUST be copied from the source message to the SOAP header carrying the fragment messages, with the following exceptions:
    - If more than one user message is present, only the first user message is used to construct the fragment envelope, with the following exceptions:
        - The `eb3:MessageId` MUST be reset to a new, unique value.
        - Any `eb3:PayloadInfo` elements are not copied.
    - Any `eb3:SignalMessage` elements are not copied.
    - Only properties specified (in **Pmode[].Splitting.RoutingProperties**) as needing to be copied are copied, with unmodified values. (Some properties may be used to route messages, or for selective pulling).

- The joining protocol reassembles fragments in groups based on sequence number. It is not needed to use InOrder delivery as the joining MSH will rearrange the data parts as appropriate.

- Messages containing `eb3:PullRequest` signals MUST never be split.

- An MSH SHOULD use the value of `eb3:MessageId` as the value of `GroupId` to support monitoring and message tracking.

- If the channel binding of the Pmode has the value **Pull** and the message is split into multiple fragments, then each of the fragments is pulled individually. The pull request is authorized for messages split in fragments using the same parameters as for messages that are not split. The order in which a sending MSH releases fragments in response to pull requests is undefined. A sending MSH MAY release the fragment messages in ascending order.

- The message splitting and joining protocol is constrained to asynchronous communication. Therefore the following constraints on binding of response messages apply:

  - The value for **Pmode.MEPBinding** MUST NOT be “sync”.

  - The value for **Pmode.ErrorHandling.Report.AsResponse** MUST NOT be “True”.

  - The value for **Pmode[1].Reliability.AtLeastOnce.ReplyPattern** MUST NOT be “Response”.

  - The value for **Pmode[1].Security.SendReceipt.ReplyPattern** MUST NOT be “Response”.

Splitting and bundling relate to other ebMS functionality as follows:

- Bundling and message splitting may seem to exclude each other. After all, unbundled user messages are smaller than bundled messages. However, there are situations where it may make sense to first bundle and then split. For example, if in some context a message fragment size of 500 KB is considered optimal, and there are six user messages to be sent, one of 700 KB and five messages of 25 KB each, then these could be bundled in one large message, then split into two fragments.

- The AS4 compression feature is defined in the AS4 profile. This AS4 feature applies compression to a single payload. The optional compression feature of the splitting/joining protocol applies to the complete MIME envelope. So it may be more effective in situations where the source message contains a large number of small payloads, in particular XML documents based on a common vocabulary.

To generate a receipt acknowledgment for non-repudiation, the receiving ebMS module needs to compute hash value for the payload parts (and sign them). This differs from regular ebMS processing, where the NRR module can reuse the computed digests from the WS-Security module (validated by that module).

The following diagram shows how splitting/joining fits in the SOAP protocol hierarchy with selected ebMS functional layers, security and reliability.

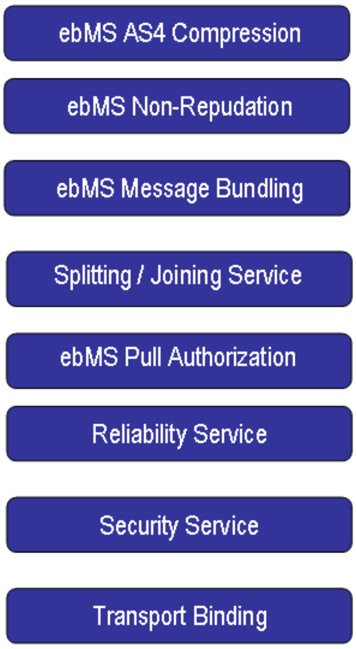

Figure 18: MSH functional layers

Other bindings to ebMS 3.0 than the binding specified in this section are conceivable and MAY be specified in other conformance profiles.

6.5 Configuration

The following processing mode parameter control the message splitting/joining protocol:

- **Pmode[].Splitting**: if present, indicates that messages sent using this Pmode can be split.
- **Pmode[].Splitting.FragmentSize**: maximum size of fragments.
- **Pmode[].Splitting.RoutingProperties**: a list of message properties that a sending MSH SHOULD copy to the fragment SOAP header.
- **Pmode[].Splitting.Compression**: If present, indicates that the message must be compressed before splitting.
- **Pmode[].Splitting.Compression.Algorithm**: Indicates which compression algorithm to use.
- **Pmode[].Splitting.JoinInterval**: Specifies the maximum time to expect and process additional fragments after the first fragment is received

6.6 Errors

An MSH MUST generate errors related to the splitting/joining of fragments in a number of situations. The splitting/joining protocol is typically applied as part of a higher-level standard or profile like ebMS, and any errors in the protocol are escalated to this higher level. The following table defines the errors and maps them to ebMS 3.0 errors.

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS:0040	BadFragmentGroup	failure		A fragment is received that relates to a group that was previously rejected.
EBMS:0041	DuplicateMessageSize	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS:0042	DuplicateFragmentCount	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS:0043	DuplicateMessageHeader	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBSMS:0044	DuplicateAction	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS:0045	DuplicateCompressionInfo	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for a compression element.
EBMS:0046	DuplicateFragment	failure		A fragment is received but a previously received fragment message had the same values for GroupId and FragmentNum
EBMS:0047	BadFragmentStructure	failure		The href attribute does not reference a valid MIME data part, MIME parts other than the fragment header and a data part are in the message. are added or the SOAP Body is not empty.
EBMS:0048	BadFragmentNum	failure		An incoming message fragment has a a value greater than the known FragmentCount.
EBMS:0049	BadFragmentCount	failure		A value is set for FragmentCount, but a previously received fragment had a greature value.
EBMS:0050	FragmentSizeExceeded	warning		The size of the data part in a fragment message is greater than **Pmode[].Splitting.FragmentSize**
EBMS:0051	ReceiveIntervalExceeded	failure		More time than **Pmode[].Splitting.JoinInterval** has passed since the first fragment was received but not all other fragments are received.
EBMS:0052	BadProperties	warning		Message properties were present in the fragment SOAP

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
				header that were not specified in **Pmode[].Splitting.RoutingProperties**
EBMS:0053	HeaderMismatch	failure		The eb3:Message header copied to the fragment header does not match the eb3:Message header in the reassembled source message.
EBMS:0054	OutOfStorageSpace	failure		Not enough disk space available to store all (expected) fragments of the group.
EBMS:0055	DecompressionError	failure	processing	An error occurred while decompressing the reassembled message.

6.7 Security Considerations

The join operation combines data received in different message fragments into a larger message, based on a shared value for `GroupId.` When the splitting and joining operations apply at a higher level in the SOAP stack than WS-Security, as in the binding for ebXML Messaging described in section 6.4.2 and message fragments received from different senders use the same `GroupId`, data from different senders will be merged. This is a vulnerability that can be addressed in the following way:

- Sender and Receiver SHOULD only configure **Pmode[].Splitting** for processing modes that provide secure transmission of the `GroupId` value, by using transport level encryption, trusted intermediaries and/or by encrypting the `MessageFragment` SOAP header.
- The sending MSH SHOULD generate unpredictable, random, sufficiently long values for `GroupId` and SHOULD prevent parties other than the receiving MSH to determine which values are used.
- The receiving MSH SHOULD use distinct value spaces for different Sender MSHs and index and correlate message fragments on a pair <`eb3:Messaging//eb3:From/eb3:PartyId`, `mf:MessageFragment/GroupId`> rather than on `GroupId` only.

Profiles of this specification MAY use WS-SecureConversation [WSSC13] to minimize the security overhead of processing fragment messages as separate SOAP messages.

7 Variants in Message Exchange Patterns Execution

This chapter defines variants in executing MEPs associated with a P-Mode, which require implementing new features in an MSH.

7.1 Selective message pulling

Selective pulling consists of pulling only from a subset of messages posted on an MPC, as defined by some common attribute value in their message header. Selective pulling is not intended as a general querying mechanism over message content. It only intends to support the most common cases of message selection, e.g. selecting messages that belong to the same conversation (i.e. with same `eb3:ConversationId value`) or messages that are responses to some previous requests (i.e. with a particular `eb3:RefToMessageId` value). The selection feature is limited to a set of such header elements, and is limited to exact value match (i.e. no sub-string comparison or relational operators).

In selective pulling, the `eb3:PullRequest` element has one or more child element(s) that can be any from a subset of those elements used under the `eb3:UserMessage` or `eb3:SignalMessage` elements.

For example, a Signal message containing the following `eb3:PullRequest` element will only pull a message (either User message or a Signal message such as a Receipt or and Error) that has an `eb3:MessageInfo/eb3:RefToMessageId` element with value equal to '11223344@initiator.example.com':

```
<eb3:SignalMessage>
  <eb3:MessageInfo>
    <eb3:Timestamp>2010-07-25T12:19:05</eb3:Timestamp>
    <eb3:MessageId>UUID-2@initiator.example.com</eb3:MessageId>
  </eb3:MessageInfo>
  <eb3:PullRequest mpc="http://msh.example.com/mpc123">

<eb3:RefToMessageId>11223344@initiator.example.com</eb3:RefToMessageId>
  </eb3:PullRequest>
</eb3:SignalMessage>
```

The elements that can be used as children of `eb3:PullRequest` are called *selection items*. A Pull signal containing selection items is called a *selective Pull signal*. Selection items are named after corresponding message header elements. Only a subset of header elements are eligible as selection items. Two categories of selection items are considered: (a) simple selection items: these are elements without children, (b) complex selection items: these are elements with children.

1. Simple selection items:
   - eb3:RefToMessageId element
   - `eb3:ConversationId` element
   - `eb3:AgreementRef` element
   - `eb3:Service` element
   - `eb3:Action` element

The selection semantics of such items is: messages from the targeted MPC will be pulled only if they contain a header element matching exactly the selection item. For example, the selection item: `<eb3:Service>QuoteToCollect</eb3:Service>` will not select a message containing: `<eb3:Service type="MyServiceTypes">QuoteToCollect</eb3:Service>` due to a mismatch on the `@type` attribute.

2. Complex selection items:

- eb3:From element element: this item may contain one or more eb3:PartyId elements and at most one eb3:Role element. The selection semantics is that only messages with eb3:PartyInfo/eb3:From containing the same elements (including attributes values if any) as those under this selection item - or a superset of these - will be pulled.
- eb3:To element element: this item may contain one or more eb3:PartyId elements, and at most one eb3:Role element. The selection semantics is that only messages with eb3:PartyInfo/e3b:To containing the same elements (including attributes values if any) as those under this selection item - or a superset of these - will be pulled.
- eb3:MessageProperties element: this item may contain one or more eb3:Property elements. The selection semantics is that only messages that have a property set eb3:MessageProperties containing the set of eb3:Property elements under this item, with specified values, will be pulled. Note that both @name attribute value and element value must be matched.

Any combination of above selection items (complex and simple) can be used as immediate children of eb3:PullRequest: such combination has the semantics of a boolean conjunction.

The MSH receiving a selective eb3:PullRequest MUST either:

(a) return a message assigned to the same @mpc that successfully matches the contained selection items as described in above selection semantics, or:

(b) return error EBMS:0006 (EmptyMessagePartitionChannel) if no message is currently assigned to this MPC , that matches the selection items in (a).

7.2 Alternate MEP bindings

Sometimes messaging partners want more flexibility than using just one MEP transport channel binding for message exchanges governed by a P-Mode. This section defines the "alternate MEP" feature allowing such flexibility, often by involving the pulling mechanism.

An example of such a flexibility requirement is when a Two-Way / Sync exchange cannot be honored by the responding MSH due to the application layer (Consumer of the request message) taking too long in producing the response. In this case, it is convenient to allow the initiating MSH to pull the response message later, effectively shifting the MEP of this exchange from a tentative Two-Way / Sync to an actual Two-Way / Push-and-Pull exchange. The latter is called an alternate (or "fall-back") MEP for the P-Mode governing the exchange, while the initially defined MEP (here the Two-Way / Sync) is the "preferred" MEP in the P-Mode.

This specification is restricting the alternate MEPs to only concern cases where:

- The preferred MEP is a Two-Way MEP.
- Alternates do not modify the way the Request message is transmitted but only modify the way the Response message is bound to the underlying channel.
- Not more than one alternate MEP binding is defined in a P-Mode, besides the preferred MEP binding.

In order to support this feature, new P-Mode parameters are defined:

- **PMode.MEPbinding.Alternate:** defines the alternate transport channel binding for the MEP defined in **PMode.MEP**. For example, http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pushAndPull   is an alternate channel binding for a Two-Way MEP where the preferred channel binding is: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/sync.
- **PMode[1].BusinessInfo.MPC.Alternate**: The value of this parameter is the identifier of the MPC (Message Partition Channel) to which the message sent using the alternate MEP is assigned. It maps to the attribute eb3:Messaging/eb3:UserMessage/@mpc. This

parameter is optional even when an **MEPbinding.Alternate** has been defined: in that case, the response is assigned to the same MPC regardless of its MEP binding (preferred MEP or alternate MEP).

- Other parameters already defined MUST be used for supporting the alternate MEP binding, when applicable. For example, when the alternate MEP binding introduces pulling, parameters such as **PMode[].Security.PModeAuthorize** and **PMode.Initiator.Authorization**, are to be used for governing the authorization of pulling in the alternate MEP binding.

In order to support the alternate MEP binding feature, both an initiating MSH and a responding MSH MUST understand these P-Mode parameters.

A Responding MSH MUST take the initiative of indicating its use of the alternate MEP binding, by sending an Error signal with code EBMS:0060 "ResponseUsingAlternateMEP" instead of a response message, when responding to a request message. For example, shifting from a Two-Way / Sync MEP binding to a Two-Way / Push-and-Pull MEP binding will be indicated by sending back synchronously an EBMS:0060 Error signal referring to this request, instead of the actual response to the request. The responding MSH MUST then transmit the actual response message on the alternate MPC (either push or pull, depending on the alternate MEP binding definition) when this response is available from its Producer application.

An Initiating MSH MUST accept responses over alternate MPCs when notifed with EBMS:0060 "ResponseUsingAlternateMEP" signal, and MUST take the initiative of pulling these responses in case the alternate MEP defines a Pull mode for the response. In such case, the Initiating MEP MAY pull the response using selective pulling based on the simple selective item `eb3:RefToMessageId,` using the value of `eb3:MessageId` in the request.

When using an alternate MEP with the same MPC as specified for the preferred MEP, the message header `eb3:Messaging` will be same regardless which MEP is used.

When using an alternate MEP to send a response message, the same reliable messaging features and security features apply as for the preferred MEP.

In case both MEP bindings use the same MPC and when reliable messaging for responses is required by the P-Mode, a Responding MSH SHOULD send a response over the same reliable messaging sequence regardless which MEP (preferred or alternate) is used for this P-Mode, In other words, the entire SOAP header of a response message should not vary whether the message is sent using preferred or alternate MEP over the same MPC.

11 Processing Mode Extensions

11.1 Overview

The ebMS 3.0 Core Specification [EBMS3CORE] defines a core set of processing mode parameters that were deemed essential to support the implementation of the ebMS 3.0 OASIS Standard. The parameters do not form an exhaustive set as profiles and extensions may require additional parameters to be agreed upon. This chapter extends this set of Pmode parameters with parameters needed to support the extended functionality defined in this specification and other parameters that ebMS 3.0 implementers have found to be useful. It references additional Pmode parameter defined in the AS4 profile [AS4]. It also describes any additional values or distinct semantics of parameters compared to the Core Specification. This section follows the grouping of parameters in the ebMS 3.0 Core Specification, where new parameters are added to existing groups.

11.2 Pmode[1].Addressing

The multihop protocol uses an Addressing group containing an Endpoint Reference group, defined in section 2.7.1

- **Addressing.Address**
- **Addressing.EPR.RoutingInput**
- **Addressing.EPR.RoutingInput.Initiator.Party**
- **Addressing.EPR.RoutingInput.Initiator.Role**
- **Addressing.EPR.RoutingInput.Responder.Party**
- **Addressing.EPR.RoutingInput.Responder.Role**
- **Addressing.EPR.RoutingInput.BusinessInfo.Service**
- **Addressing.EPR.RoutingInput.BusinessInfo.Action**
- **Addressing.EPR.RoutingInput.BusinessInfo.Properties**
- **Addressing.EPR.RoutingInput.BusinessInfo.MPC**
- **Addressing.Action**

11.3 Pmode.MEPBinding

This specification defines an extension to the MEPBinding Pmode to use HTTP pipelining.

11.3.1 HTTP Pipeline Capability

HTTP/1.1 [HTTP11] specifies that clients and servers may coordinate requests and responses by “pipelining” requests over a single open TCP connection subject to two basic constraints:

- A client must be able to send multiple requests before receiving HTTP responses for those requests.
- A server must return responses to requests in the same order that the requests have been received.

The capability to engage in a MEPbinding that uses pipelining is indicated by setting **PMode.MEPbinding.Pipelining** to “true”.

For MEPbindings involving HTTP 1.1, the default value for this parameter is “false.” HTTP 1.1 is currently the only protocol binding substrate supporting pipelining semantics, though other protocols may emerge with an option to support the basic constraints.

11.3.2 EbMS MEPs and Pipelining

The HTTP/1.1 protocol session always consists of a request to apply a specific method to a URI-specified resource. A response always includes a status code. In either case, the protocol data unit (“message”) can contain various headers and a message “entity,” which has a MIME structure.

ebMS has two P-Mode parameters, the **PMode.MEP** and the **PMode.MEPbinding**. The PMode.MEPbinding value indicates how the PMode.MEP is accomplished within an application transfer protocol, such as HTTP/1.1.

When used in combination with HTTP, the **MEPbinding**s reflect various kinds of entities that can appear within an HTTP/1.1 request/response pair. The **pull** MEPbinding , http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pull, creates the expectation that the request contains a Pull signal message and the HTTP response may contain an ebMS user message.

While pipelining might be used for any of the ebMS-specified **MEPbindings**, the HTTP/1.1 specification contains several cautions.

First, the specification advises “Clients SHOULD NOT pipeline requests using non-idempotent methods or non-idempotent sequences of methods (see section 9.1.2).” Most ebMS **MEPBindings** use HTTP POST method requests, which are not generally regarded as idempotent. However, normally each POST method request carries a distinct MIME entity. Exceptions would include retries or resending and several other situations, such as a Pull signal. A sequence of Pull signals (using the same MPC) would not be idempotent because normal operation would yield distinct results for the same Pull signal (unless the MPC is empty and they all return the error signal for an empty MPC.

For the non-idempotent cases, if they are pipelined, clients SHOULD wait for the return of a status code before posting the next message.

A more practical limitation on pipelining pertains to the **SYNC MEPbinding.** Because an ebMS user message in an HTTP response may involve obtaining data from internal applications, unpredictable latencies in response time are common. For this reason, pipelining support may be withdrawn for certain **MEPbinding**s because of end user deployment constraints, even though the capability for pipelining is present in the MSH.

11.3.3 ebMS Alternate MEPs

The P-Mode parameter indicating an MEP alternate is:

- **PMode.MEPbinding.Alternate:** defines the alternate transport channel binding for the MEP defined in **PMode.MEP**. It takes values similar to those in **PMode.MEP**. For example, `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pushAndPull` is an alternate channel binding for a Two-Way MEP where the preferred channel binding is: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/sync.`

12.1 Pmode[1].Protocol

This specification adds Pmode parameters for transport security, controlling the SOAP actor or role attributes, and large file handling based on protocol restarts.

12.1.1 Transport Security

In many deployments of ebMS 3.0, transport layer security will be used in addition to (or instead of) SOAP-based security mechanisms like WS-Security [WSS10],[WSS11]or WS-SecureConversation [WSSC13]. The following parameters have been added to express transport layer security agreements. They have been modeled after corresponding elements in [ebCPPA 3.0]. Conformance profiles MAY further constrain the acceptable values of these parameters.

- **Pmode[1].Protocol.Security.Protocol** This parameter identifies the security protocol used. If this parameter is not specified, no security protocol is used. The value "*TLS*" for this Pmode parameter MUST be interpreted as indicating the use of the IETF Transport Layer Security protocol and this is the RECOMMENDED transport layer.

- **Pmode[1].Protocol.Security.ProtocolVersion** This parameter specifies the minimum version of the security protocol to use and MUST only be specified when a security protocol has been specified. In combination with the value "*TLS*" for the previous parameter, the value "*1.2*" for this Pmode parameters indicates the use of the TLS 1.2 protocol [RFC5246].

- **Pmode[1].Protocol.Security.SecurityAlgorithm** This parameter specifies an agreed list of algorithm suites that may be used for key establishment, encryption and authentication. Note that some (versions of) security protocols require implementations to support particular cipher suites. For example, TLS 1.1 compliant implementations MUST support the *TLS_RSA_WITH_3DES_EDE_CBC_SHA* cipher suite and TLS 1.2 compliant implementations MUST support the *TLS_RSA_WITH_AES_128_CBC_SHA* cipher suite.

- **Pmode[1].Protocol.Security.Server.Certificate** This parameter specifies a server certificate to use. If this parameter is absent, server certificate authentication is not required.

- **Pmode[1].Protocol.Security.Client.Certificate** This parameter similarly specifies a client certificate to use. If this parameter is absent, client certificate authentication is not required.

- **Pmode[1].Protocol.Security.Server.AnchorCertificates** This parameter may be used to specify a list of trusted anchors. These anchors are used in the process of server certificate path validation. If a server certificate is used that does not chain to one of these trusted anchors, it is considered invalid.

- **Pmode[1].Protocol.Security.Client.AnchorCertificates** This parameter may be used for client certificate path validation in a similar way as the preceding parameter does for server certificates.

- **Pmode.[1].Protocol.Security.minimumKeySize** This parameter specifies the minimum acceptable size of the symmetric encryption key.

12.1.2 Controlling the SOAP Actor or Role

In a multi-hop context, all the **Pmode[1].Protocol** parameters MUST be defined separately for **init** and **resp** PModes (using the terminology defined in section 2.7.2) with the exception of **Pmode[1].Protocol.SOAPVersion** and the following new parameter:

- **Pmode[1].Protocol.AddActorOrRoleAttribute** {true/false} This parameter is defines whether or not the `actor` or `role` attribute is present in the `eb3:Messaging` element. If `true`, and if the value for the **Pmode[1].Protocol.SOAPVersion** parameter is "`1.1`" then the attribute `actor` is present. If `true`, and if the value for the **Pmode[1].Protocol.SOAPVersion** parameter is "`1.2`" then the attribute `role` is present. In both case the attribute has a fixed value of "`http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/nextmsh`".

12.1.3 Large File Handling using Protocol Restart

Some B2B message exchanges involve the transfer of large messages. In situations where a payload compression feature (as the one described in section 12.5) does not reduce this size sufficiently, and the message splitting/joining protocol described in section 6 is not available, a mechanism that allows trading partners to restart failed transfers from the point of failure is desirable.

Trading partners can configure a restart feature using the following parameters:

- **Pmode.[1].Protocol.Restart** *{True, False}* This parameter expresses whether or not a restart feature is to be used for the specified processing mode. The default value is *False*.
- **Pmode.[1].Protocol.Restart.Protocol** This parameter identifies the transport restart protocol to use. The value *as2-restart* identifies the the AS2 Restart protocol [AS2-RESTART]. In case the transport protocol is HTTP 1.1, this is the default restart protocol.
- **Pmode.[1].Protocol.Restart.Interval** This parameter expresses the maximum amount of time the recipient of a message SHOULD cache a temporary copy of the incomplete message, for a particular message transfer.

These parameters MUST be interpreted in the context of the transport protocol that is used and the MEP Binding. When using the [AS2-RESTART] protocol, **PMode.MEPBinding** MUST be set to the value "*push*".

12.2 Pmode[1].ErrorHandling

The following parameter allows initiators to pull errors related to messages they sent from intermediaries.

- **Pmode[1].ErrorHandling.Report.ReceiverErrors.ReplyPattern** configures the way receiver errors are transmitted and generalizes and deprecates the parameter **Pmode[1].ErrorHandling.Report.AsResponse** of [EBMS3CORE]. It has three possible values. The values *callback* and *response* are the equivalent of *false* and *true* for **Pmode[1].ErrorHandling.Report.AsResponse**. The new value *pull* indicates errors (which have an `ebint:RoutingInput` reference parameter to be routed) are to be pulled.

Additional routing information can be attached to errors using the parameters introduced in section 2.7.1

- **Pmode[1].Errorhandling.Report.SenderErrorsTo.Addressing.EPR**
- **PMode[1].Errorhandling.Report.ReceiverErrorsTo. Addressing.EPR**

The following parameter is defined in section 4.2.2 of [AS4]:

- **Pmode[1].ErrorHandling.Report.MissingReceiptNotifyProducer.** This parameter controls the behavior of a Sending MSH that fails to receive a Receipt.

12.3 Pmode[1].Reliability

12.3.1 Reliability Protocol

Whereas some ebMS 3.0 implementations may only support one reliability protocol, some products may support multiple protocols, or versions of protocols. An organization may use one reliability protocol with one trading partner or service and another protocol with another. The following parameter makes the choice of reliability protocol a configuration option.

The **Pmode[1].Reliability.Protocol** identifies the reliable messaging protocol, and the version of that protocol, that is used in a particular message exchange.

- The value http://docs.oasis-open.org/ws-rx/wsrm/200702 MUST be interpreted as identifying the WS-ReliableMessaging 1.1 [WSRM11] protocol or any backwards-compatible updates of it.
- The value http://docs.oasis-open.org/wsrm/2004/06/ws-reliability-1.1.xsd MUST be interpreted as identifying the WS-Reliability 1.1 [WSR11]protocol or any backwards-compatible updates of it.

Conformance profiles or implementations may limit the choice of reliability protocol or define different values to indicate other reliability protocols.

12.3.2 Reliability of the Pull Signal

Sections B.1.3 and B.2.3 of the ebMS 3.0 Core Specification specify that a Pull signal for a user message that is sent under an at-least-once delivery contract must itself be transmitted reliably. This supports point-to-point message pulling as well as multi-hop "end-to-end" pulling, where the `eb3:PullRequest` is forwarded to and processed by, the Sending MSH. It is not appropriate in situations where the request is processed by an intermediary that only provides temporary intermediate message storage. The reliable messaging model specified in section 2 is based on end-to-end message retransmission where ebMS intermediaries do not act as RMS or RMD.

The Boolean valued **Pmode[1].Reliability.AtLeastOnce.Contract.ReliablePull** parameter accommodates situations where an MSH may pull some messages directly from the Sending MSH and others from an intermediary. The default value of this parameter is *true*, meaning that Pull signals are sent reliably for any Pmode that has a value *true* for **Pmode[1].Reliability.AtLeastOnce.Contract**. The non-default value *false* indicates that the Pull request is not sent reliably. (For more discussion, see discussion in the 2.4.7.2 and 4.2 sections).

12.3.3 Transmitting Acknowledgements

In a multi-hop context, there is a requirement to retrieve acknowledgments that are returned asynchronously from intermediaries. The following parameters support this:

- **Pmode[1].Reliability.AtLeastOnce.ReplyPattern**. This parameter is defined in [EBMS3CORE] as having three values: "Response", "Callback" and "Poll". The value "Poll" is limited to WS-Reliability and indicates that acknowledgments are retrieved via `wsr:PollRequest` messages [WSR11]. This specification adds a fourth value "Pull", which specifies that acknowledgments are to be retrieved by ebMS 3.0 PullRequest signal messages. If this value is used, then one of the following two cases MUST apply:

1. A value is specified for **Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo.Addressing.** In this case the contents of the embedded reference parameter MUST be used to created a reference parameter.
2. If no value is specified for this parameter, then a reference parameter is inferred from the user message using the mechanism "Inferred RoutingInput for the reverse path" described in section 2.6.2 .

The reference parameter provides a value for the the `@mpc` attribute that identifies the channel that the acknowledgement can be pulled from.

- **Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo.Addressing.EPR** This parameter contains an EPR structure as discussed in section 2.7.1 . If present, its contents must be used to construct a routing input parameter that is attached to the acknowledgment message.

12.4 Pmode[1].Security

12.4.1 Receipts

- **Pmode[1].Security.SendReceipt.ReplyTo**. This parameter indicates the address to which the `eb3:SignalMessage` containing a requested `eb3:Receipt` is sent to, in situations where the **Pmode[1].Security.SendReceipt.ReplyPattern** parameter has a value different from "*response*". It serves a similar purpose as the **Pmode[1].Reliability.AtLeastOnce.Contract.AcksTo** for reliable messaging acknowledgments and **Pmode[1].Errorhandling.Report.ReceiverErrorsTo** for receiver errors.
- **Pmode[1].Security.SendReceipt.ReplyPattern**. In a multi-hop context, this parameter defined in [EBMS3CORE] may have a third value, "*pull*". This value indicates that receipt signals are pulled using ebMS 3 Pull messages. If this option is selected, an `ebint:RoutingInput` reference parameter MUST be added to the receipt, which specifies

the MPC from which the receipt can be pulled and routing information that supports I-Cloud routing. The **Pmode[1].Security.SendReceipt.Addressing.EPR** MUST be specified and the value of the routing input MUST be derived from it.

- **Pmode[1].Security.SendReceipt.Addressing.EPR**. This parameter can be used to specify routing information to be attached to receipt signals.

The following parameter is defined in section 4.2.1 of [AS4]:

- **Pmode[1].Security.SendReceipt.NonRepudiation** {true/false} This parameter constrains the content of `eb3:Receipt` to be a valid `ebbp:NonRepudiationInformation` element [EBBPSIG].

12.4.2 Secure Conversations

This specification does not fully specify the use of the WS-SecureConversation OASIS Standard [WSSC13] in combination with ebMS. However, initial support for routing of WS-SecureConversation messages is discussed in 4.2 . This requires the following additional parameters:

- **Pmode[1].Security.Conversation** {true/false} Specifies whether or not messages exchanged using this Pmode are part of secure conversations.
- **Pmode[1].Security.Conversation.Correlation** Defines which messages can be assigned to the same secure conversation, similarly to the **Pmode[1].Reliability.Correlation** parameter.

Conformance profiles may further specify the use of secure conversations.

12.5 Pmode[1].PayloadService

The AS4 profile defines a payload compression service:

- **PMode[1].PayloadService.Compression**: {true / false}

12.6 Pmode[1].ReceptionAwareness

Section 3.2 of the AS4 profile defines for additional Pmode parameters for reception awareness [AS4].

- **Pmode[1].ReceptionAwareness**: (true / false)
- **PMode[1].ReceptionAwareness.Replay**: (true / false)
- **Pmode[1].ReceptionAwareness.Replay.Parameters**.
- **PMode[1].ReceptionAwareness.DuplicateDetection**: (true / false)
- **Pmode[1].ReceptionAwareness.DetectDuplicates.Parameters**

12.7 Pmode[1].Bundling

The following parameters are defined in section 5.3 :

- **Pmode[].bundling.policy**
- **Pmode[].bundling.compatibility.pmodelist**
- **Pmode[].bundling.maxsize**
- **Pmode[].bundling.maxdelay**
- The following parameters are defined in section 5.10.1 :
- **Pmode[].bundling.ordering.policy**

- **Pmode[].bundling.ordering.scope**

12.8 Pmode[].Splitting

The following parameters are defined in section 6.5

- **Pmode[].Splitting**
- **Pmode[].Splitting.FragmentSize**
- **Pmode[].Splitting.RoutingProperties**.
- **Pmode[].Splitting.Compression**.
- **Pmode[].Splitting.Compression.Algorithm**.
- **Pmode[].Splitting.JoinInterval**

12.9 Pmode[].BusinessInfo

When defining an alternate MEP, the following parameter MAY be added to define an alternate MPC:

- **PMode[1].BusinessInfo.MPC.Alternate**: The value of this parameter is the identifier of the MPC (Message Partition Channel) to which the message sent using the alternate MEP is assigned. It maps to the attribute `eb3:Messaging/eb3:UserMessage/@mpc`.

14 Conformance

This conformance clause is defining four conformance profiles:

1. one conformance profile for the multi-hop part of this specification – called here the **simple multi-hop conformance profile**,
2. one conformance profile for the message bundling part of this specification – called here the **simple bundling conformance profile**,
3. one conformance profile for the extended messaging features - called here the **basic messaging extensions conformance profile**,
4. one conformance profile for the large message splitting and joining protocol – called here the **simple fragmented message transfer profile**,

as these four feature sets are largely independent and composable in various ways.

These four simple conformance profiles are defined here as addressing the most basic of the expected usages of this specification. They are not exclusive of other conformance profiles that may be defined separately outside this specification to accommodate the needs of some user communities. Such conformance profiles may require a different set of features to be implemented – either more features, or sometimes less if a particular context of use is expected that allows even simpler implementations.

In the absence of any claim to another externally-defined conformance profile, an implementation of this specification is expected to conform to either one or both of the conformance profiles defined here, as an interoperability baseline.

14.1 Simple Multi-hop Conformance Profile

The requirements for this profile are different, depending on the role an MSH is playing in a multi-hop environment.

MSH in Intermediary role:

An Intermediary MSH must comply with the following:

- must satisfy all strict (MUST) normative requirements in sections 2.4.5,
- must support the store-and-forward message forwarding model described in 2.5.2.
- must support one MEP bridging model (over the four described in 2.5.3): “push-on-push” .
- Must support a routing function as described in 2.5.5, that can process both `eb3:UserMessage` header and the `ebint:RoutingInput` reference parameter header block.
- Must conform to the error handling requirements (2.5.6) and must support at least the EBMS:0005 ConnectionFailure error and the new EBMS:0020 RoutingFailure.
- Must support WS-addressing as described in 2.6.3.

In addition, when a conforming MSH Intermediary implements additional features specified in this document, it must conform to all related requirements.

MSH in Endpoint role:

- Must support at least the Case 1 edge binding ("first-and-last-push") for one-way MEPs described in 2.4.7.1 (pushing messages from Sender), and at least the Case 1 edge binding ("Request-push-last-push and Reply-push-last-push") for two-way MEPs described in 2.4.8.1.
- Must support the Endpoint requirements in 2.6, but is not required to support the requirements for non-ebMS messages (case 3 in 2.6.1). In particular, it must be able to add the

`ebint:RoutingInput` header at least to ebMS Signal messages it generates. It must be able to infer the value of `ebint:RoutingInput` for response messages (2.6.2, item 4).

- Must implement all the PMode parameters that control the above features.

In addition, when a conforming MSH endpoint implements additional features specified in this document, it must conform to all related requirements. In particular:

- If an endpoint MSH supports WS-Addressing it must then comply with all requirements related to WS-Addressing, including implementing related PMode parameters such as “PMode.Addressing” parameters that control the use of WS-Addressing.
- If an endpoint MSH supports WS-ReliableMessaging, it must then comply with all requirements related to reliable messaging as well as related to the sending of non-ebMS messages (for RM signals), including implementing related PMode parameters.
- If bundling is in use, endpoints and intermediaries must comply with the strict requirements in section 3.9 (“Bundling for Multi-hop” section).
- In case a conforming MSH receives messages that exhibit multihop features beyond those required by this conformance profile, it SHOULD generate an EBMS:0008 error (FeatureNotSupported).
- All features referred to in this profile MUST be implemented in conformance to ebMS V3 core specification, and messages conforming to this profile MUST also conform to the ebMS V3 core specification.

14.2 Simple Bundling Conformance Profile:

A conforming MSH MUST comply with the following:

- It must satisfy all packaging and bundling rules in section 5.2 that are strict requirements (MUST or equivalent).
- It must implement at least the PMode parameter **PMode.bundling.policy** (section 5.3) with at least values “never” and “always”.
- It must understand and handle the BundlingError error message (EBMS:0030) when it receives one, although it is not required to generate one when in Sending role, and is only required to generate one in Receiving role when either one of **Pmode[].bundling.maxsize**, **Pmode[].bundling.compatibility** or **Pmode[].bundling.maxdelay** parameters is implemented and the corresponding error rule stated in 5.5 applies.
- A sending MSH must implement at least one of the two ways to control bundling as described in section 5.4 .
- It must satisfy the strict requirements for bundling responses as described in 5.8 .
- MUST implement all the PMode parameters that control the above features.
- MUST implement the value *undefined* for **Pmode[].bundling.ordering.policy**.

In addition, when a conforming MSH endpoint implements additional features beyond the Simple Bundling Conformance Profile, it must conform to all related requirements. In particular:

- The value Bundling delivery policies, if supported, must be controlled as described in section 3.10.
- In case a conforming MSH receives messages that exhibit bundling features beyond those required by this conformance profile, it SHOULD generate an EBMS:0008 error (FeatureNotSupported).
- All features referred to in this profile MUST be implemented in conformance to ebMS V3 core specification, and messages conforming to this profile MUST also conform to the ebMS V3 core specification.

14.3 Basic Messaging Extensions Conformance Profile:

This conformance profile concerns the "Variants in Message Exchange Pattern Execution" (Section 5) and establishes a minimal set of features from those specified in this section, that is suitable as an interoperable baseline of advanced features.

A conforming MSH MUST comply with the following:

- About the selective pulling capability, support for the `eb3:RefToMessageId` and `eb3:ConversationId` simple selection items, which are never supposed to be used together in the same Pull request.
- About alternate MEPs, support Two-Way / Push-and-Pull as alternate MEP to a Two-Way / Sync preferred MEP. This means: (a) for an initiating MSH to be able to accept responses in either mode for any exchange governed by such a P-Mode, and in particular to pull the response from the alternate MPC in case it received the EBMS:0060 "ResponseUsingAlternateMEP" signal instead of the synchronous response, (b) for a responding MSH to be able to dynamically re-assign responses to the alternate Pull MPC after sending back the EBMS:0060 "messageUsingAlternateMEP" signal as synchronous response to the request message.

14.4 Simple fragmented message transfer profile

This conformance profile concerns the Large Message Splitting and Joining functionality defined in section 6 .

- A SOAP procesor implementation is conformant with the simple fragmented message transfer profile if it satisfies all the MUST and REQUIRED level requirements defined in sections 6.2 and 6.3 .
- A WS-Addressing processor is conformant with the simple fragmented message transfer profile if it satisfies all the MUST and REQUIRED level requirements defined in sections 6.4.1 .
- As indicated in section 6.4.2 , multiple ebMS bindings for the splitting / joining protocol are conceivable. An ebMS MSH is conformant with the *simple fragmented message transfer profile* if it satisfies the MUST and REQUIRED level requirements defined in sections 6.4.2 .

Appendix A Multi-hop Routing Scenarios and Good Practices

In multi-hop environments, ebXML intermediaries provide a flexible mechanism for the routing of SOAP messages based on standardized SOAP header content. The ebMS 3.0 business document header offers a rich set of metadata elements with a standardized semantics that support a variety of messaging and routing scenarios, including document exchange and service invocation. This non-normative section discusses some routing scenarios and requirements enabled by the use of ebXML intermediaries.

A.1 Routing Scenarios

The ebMS 3.0 routing function defined by this profile supports messages carrying arbitrary payloads, including non XML data and encrypted data. It also supports routing non-ebMS messages using the `ebint:RoutingInput` WS-Addressing reference parameter. This section illustrates some scenarios that are supported by this profile:

- Routing based on business partner identity
- Routing based on business partner domains
- Routing based on requested service and action
- Defining separate logical environments for development, test, acceptance and production.

These scenarios are typical of many messaging environments and have been identified in some deployments of version 2.0 of ebXML messaging. All scenarios use routing based on pattern matching against SOAP envelope structures, rather than target URI, IP address or content of message payloads.

A.2 Routing Rules

To route messages, intermediaries need some configuration mechanism based on routing rules. Conceptually, a routing rule can be thought of as a message pattern, a destination and a set of configuration parameters that control the transmission to next MSH (via pull or push). A message pattern can be expressed using XPath expressions [XPATH]. A rule conflict resolution mechanism like the one defined in section 6.4 of [XSLT] could be used to select among multiple matching patterns.

This annex assumes three categories of message patterns:

- Patterns matching `eb3:UserMessage` content.
- Patterns matching `ebint:RoutingInput` structures
- Patterns used to forward `eb3:SignalMessage`s

The following example is an example of the first category:

```
//eb3:UserMessage[1]/eb3:PartyInfo/eb3:To/eb3:PartyId/text()
```

When applied to a SOAP message, this expression matches content in the first `eb3:UserMessage` element only and selects the destination business partner based on the PartyID value. The restriction to the first user message element avoids any routing ambiguity in situations where multiple `UserMessage` elements are “bundled” in a single SOAP envelope: all but the first user message structures are ignored by the routing function.

The second category of message patterns is needed because of the requirement to route messages other than ebMS user messages, such as ebMS response signals (receipts and errors) and non-ebMS messages like the sequence lifecycle management messages discussed in section 4.2 . These messages can be routed using the `ebint:RoutingInput` WS-Addressing routing parameter. An example of a pattern matching these messages using the same metadata as the previous pattern is:

```
//
ebint:RoutingInput/ebint:UserMessage/eb3:PartyInfo/eb3:To/eb3:PartyId/tex
t()
```

For any routing rules operating on `eb3:UserMessage`s, a rule operating on `ebint:UserMessage`s is needed to route these messages to the exact same destination.

The third category of routing pattern involves routing `eb3:SignalMessage`s in the “end-to-end pulling” case. In this scenario, the intermediary needs to connect to another ebMS node when it receives an ebMS message containing an `eb3:PullRequest`.

```
<eb3:SignalMessage>
   <eb3:MessageInfo>
     <eb3:Timestamp>2009-05-21T11:30:11.320Z</eb3:Timestamp>
     <eb3:MessageId>30c6eb92-6329-44c7-a4a3-
468d503c01f8@seller.com</eb3:MessageId>
   </eb3:MessageInfo>
   <eb3:PullRequest mpc="e5c31ef7-d750-4db8-b4dc-13a751d80b9a" />
</eb3:SignalMessage>
```

This third case of routing is similar to a regular pull request message except that there is no periodic or other scheduling of pull requests (the pull is triggered by an incoming pull request), that there is no authorization done by the intermediary (this is relayed transparently) and that the incoming request must wait for the related outgoing request to complete (no decoupling). This pattern only relies on the `mpc` attribute value. (Note that it is also possible to route pull signal messages using the previous pattern, using an appended `ebint:RoutingInput` header. If such a header is not present, the `mpc` attribute value is the only routing input.)

A.3 Business Partner Identification

It is a common requirement for electronic business messages to be routed based on the identification code for the intended recipient business partner. Examples of these include:

- EDI Value Added Networks (VANs) route messages based on partner identifiers in EDIFACT interchange header segments or ASX X12 Interchange Control Headers.
- Many messaging protocols have header elements to identify business partners using codes. An example are the `AS2-From` and `AS2-To` system identifiers of AS2 [RFC4130].

In ebXML, partner identification is expressed as a combination of a `PartyId` string, qualified by an optional `type` attribute. The content of the `PartyId type` can be retrieved from an incoming ebMS SOAP message using the following XPath expression:

```
//eb3:UserMessage[1]/eb3:PartyInfo/eb3:To/eb3:PartyId/@type
```

The following expression retrieves the actual partner identifier string for the `PartyId`

```
//eb3:UserMessage[1]/eb3:PartyInfo/eb3:To/eb3:PartyId/text()
```

An ebXML message containing the following destination information:

```
<eb3:To>
   <eb3:PartyId type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0002"

      >123456789</eb3:PartyId>
   <eb3:Role>Seller</eb3:Role>
</eb3:To>
```

matches the following XPath expression:

```
//eb3:UserMessage[1]/eb3:PartyInfo/eb3:To/eb3:PartyId[@type='urn:oasis:n
ames:tc:ebcore:partyid-type:iso6523:0002'][text()='123456789']
```

An intermediary could use this XPath expression to retrieve the transport configuration parameters for the next node from ebXML messages.

This example adopts the OASIS ebCore Party Id Type [PARTYIDTYPE] notation where `urn:oasis:names:tc:ebcore:partyid-type:iso6523:` prefix of a `PartyId type` attribute value indicate that a code list agency that is registered in ISO 6523. Value `0002` in the ISO 6523 registry is assigned to SIRENE, the business registry for France.

The combination of a “Push” channel binding for incoming messages with a “Pull” channel binding for outgoing messages and the use of `PartyId` for routing allows ebMS intermediaries to offer a *store-and-collect* functionality that replicates the “mailbox” functionality of EDI value-added networks and SMTP-based message exchanges. This enables the intermediary to support situations where *both* business partners have addressability and connection issues: the receiver can receive messages even if the sender is offline or not addressable, as long as the sender has stored the message on an intermediary accessible to both.

The routing techniques using XPath or similar pattern matching can also extend beyond the ebMS Intermediary functions: E.g. another routing practice could be to involve Payload elements, e.g. route based on some business document content. At the level of the intermediary information from attached business documents will often not be available due to end-to-end payload encryption. If that is the case, an approach is to bring-up these crucial payload elements in the header, as message properties. If the business data is not encrypted at the payload level but available in the SOAP Body, the type of pattern matching applied to SOAP headers here could be extended to apply to SOAP body content.

A.4 Business Partner Domains

A generalization from the previous scenario is a scenario where intermediaries are used to connect different communities that all use their own, distinct business identification schemas. Examples of these include cross-border trade and collaboration of government agencies across sectors.

As a first example, assume an organization in the Netherlands exchanges business documents with an organization in France. The organization in the Netherlands has a party identifier from the Association of Chambers of Commerce and Industry in the Netherlands, which has the value `0106` in ISO 6523. The party identification type for the organization in France could use the SIRENE identification (see A.3). A system of national intermediaries could be set up where each national intermediary provides secure routing to businesses in a single country, based on `PartyId` where the `type` is constrained to the national type. In addition to this, the intermediary would act as a relay to similar intermediaries in other countries.

This routing can be based only on the value for `type` and does not need to refer to any particular partner identification code. For example, the hypothetical intermediary in France could have a single routing rule to forward all messages sent to businesses in the Netherlands to an intermediary in the Netherlands based on the `0106` PartyId type value. That rule would use the following XPath expression:

```
//eb3:UserMessage[1]/eb3:PartyInfo/eb3:To/eb3:PartyId[@type=
'urn:oasis:names:tc:ebcore:partyid-type:iso6523:0106']
```

Similarly, the intermediary in the Netherlands would have a single rule to forward messages to businesses in France to its counterpart in France:

```
//eb3:UserMessage[1]/eb3:PartyInfo/eb3:To/eb3:PartyId[@type=
'urn:oasis:names:tc:ebcore:partyid-type:iso6523:0002']
```

A similar requirement is common in environments where multiple government sectors (e.g. healthcare, criminal justice, social security, immigration) have sectoral (private) networks and messaging infrastructures that are based on sector-specific identification schemas. For instance, the healthcare system could use the `urn:hl7ii` type to identify HL7 V3 instances [HL7ebMSv3]. Other sectors would use their own, distinct, organization identification mechanisms. A cross-sector routing mechanism supports collaboration among agencies across sector boundaries without requiring a single identification scheme.

A.5 Services

The ebXML document header also supports service-oriented messaging based on the `eb3:Service` and `eb3:Action` elements. An ebMS 3.0 intermediary can use these standard and required header elements to route request message to services providers and reverse route the response messages to the service consumers. Their values can be retrieved using the following XPath expressions:

```
//eb3:UserMessage[1]/eb3:CollaborationInfo/eb3:Service
//eb3:UserMessage[1]/eb3:CollaborationInfo/eb3:Action
```

As an example, the following pattern matches ebMS messages sent to a “Procurement” service.

```
//eb3:UserMessage[1]/eb3:CollaborationInfo/eb3:Service[text()='Procurem
ent']
```

In many larger environments there will be several (potential or competing) providers of a single particular service. This means that in practice routing rules are likely to require both a partner identifier (as described in Error: Reference source not found) and a service identifier.

```
//eb3:UserMessage[1]
[eb3:PartyInfo/eb3:To/eb3:PartyId[@type='urn:oasis:names:tc:ebcore:partyid
-type:iso6523:0002']
[text()='123456789']]/eb3:CollaborationInfo/eb3:Service[text()='Procureme
nt']
```

In large or distributed organizations, there may be multiple data centers hosting the business applications that provide distinct services. Each of these data centers could have its own ebXML message service handler endpoints. A separate rule would map messages related to this other service to a distinct next MSH.

```
//eb3:UserMessage[1]
[eb3:PartyInfo/eb3:To/eb3:PartyId[@type='urn:oasis:names:tc:ebcore:partyid
-type:iso6523:0002']
[text()='123456789']]/eb3:CollaborationInfo/eb3:Service[text()='Marketing
']
```

Only the last intermediary delivering messages to these MSHs needs to know which data center provides which service. When using intermediaries, services can be relocated from one data center and MSH to another, data centers can be reorganized and consolidated, services outsourced or in-sourced, without the business partners using services from those data center having to reconfigure their MSH configurations.

A.6 Separate Environments

Like other information systems, messaging systems typically follow a life cycle through various stages, such as development, test, acceptance and production. It is common practice in service management to have different environments for these stages so that versions of systems and services in various development stages are separated and can be developed, tested and deployed in parallel. In these situations it is important that messages from one environment never cross boundaries with other environments. For instance, a test message should never be confused with a production message.

There are multiple approaches to separating these environments. One approach is to use distinct (virtual) private networks, each containing endpoints and intermediaries for each environment. When using messaging intermediaries, each intermediary participates in at most one environment. Another approach is to partition the endpoints and intermediaries logically and to configure messaging intermediaries to keep the message traffic from or to systems in one particular logical environment separate from other environments.

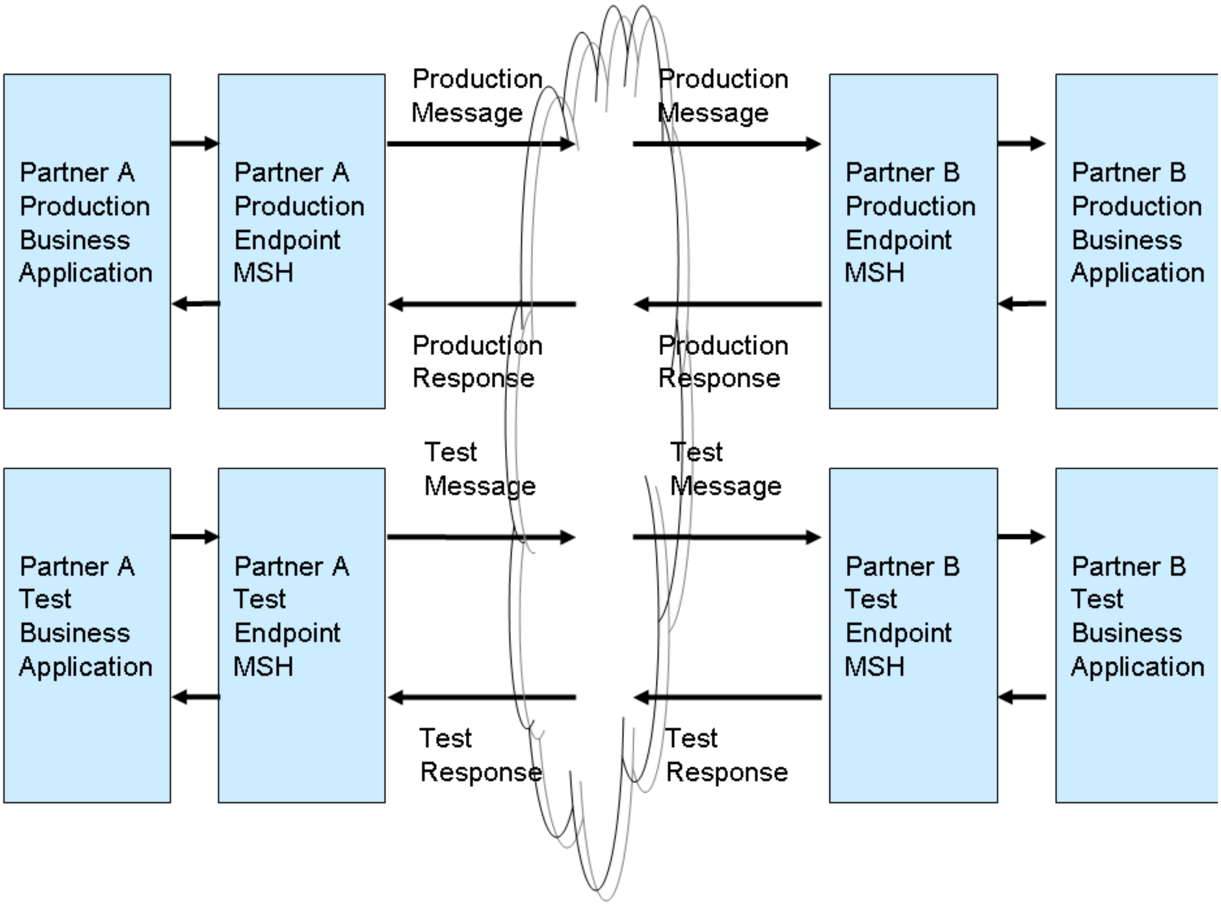
Production Message
Production Message
Partner A Production Business Application
Partner A Production Endpoint MSH
Partner B Production Endpoint MSH
Partner B Production Business Application
Production Response
Production Response
Test Message
Test Message
Partner A Test Business Application
Partner A Test Endpoint MSH
Partner B Test Endpoint MSH
Partner B Test Business Application
Test Response
Test Response

Figure 19 Separate environments for test and production

The routing function of an ebXML intermediary supports multiple approaches to meet this requirement. One approach is to use the ebMS concept of message partition channels to assign messages to a `Development`, `Test`, `Acceptance` and `Production` partition. An intermediary can route messages based on MPC using patterns like:

```
//eb3:UserMessage[1][@mpc='Production']
```

An alternative approach is to use the ebMS 3.0 feature of `MessageProperties` and have a `Property` to classify messages according to environment.

```
<eb3:MessageProperties>
      <eb3:Property name="Environment">Production</eb3:Property>
</eb3:MessageProperties>
```

This approach is more flexible as environments may be partitioned in more dimensions (e.g. for versions of services) and additional properties could be added to reflect this.

The name and values of these properties need to be standardized and used consistently in the community. A test MSH can be configured to always insert (or check for the presence of) this property and the correct value in any outgoing message and validate its correct use in incoming messages. Intermediaries can deploy routing rules that reference these properties, possibly in combination with the other message header elements discussed in this section or other properties, to route messages within the appropriate logical environment.

```
//eb3:UserMessage[1]/eb3:MessageProperties/eb3:Property[@name='Environm
ent'][text()='Production']
```

A.7 End-to-end Pulling

A separate category of messages to route are ebMS `PullRequest`s. An ebXML intermediary may handle `PullRequest`s either as requests to retrieve messages it received from other message handlers and is storing on behalf of these, or as request messages that need to be forwarded synchronously to a remote ebMS 3.0 server. The latter case is referred to as *end-to-end pulling* and involves a Receiving MSH that pulls messages from a remote Sender MSH via an Intermediary MSH. A routing rule supporting such end-to-end pulling could use a pattern containing the *mpc*, for example:

```
//eb3:SignalMessage/eb3:PullRequest[@mpc='e5c31ef7-d750-4db8-b4dc-
13a751d80b9a']
```

Note that an ebMS intermediary only considers routing messages if they are targeted by setting the value of the S12:role attribute to "http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/nextmsh".

Appendix B Refactored ebMS 3.0 Core XML Schema

As explained in section 2.5.5 , a refactored version of the normative schema for the ebXML Messaging version 3.0 Core allows more reuse of some element definitions from this schema by the `ebint:RoutingInput` and other XML schemas without affecting interoperability of ebMS 3.0 processors. This refactored schema is defined using [XMLSCHEMA-P1] and [XMLSCHEMA-P2] and is located at:

```
http://docs.oasis-open.org/ebxml-msg/... ebms-header-3_0-200704_refactored.xsd
```

The following copy is provided for reference:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
  xmlns:tns="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
  targetNamespace="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:appinfo>Schema for ebMS-3 XML Infoset</xsd:appinfo>
    <xsd:documentation xml:lang="en"> This schema defines the XML Infoset of
ebMS-3 headers.
      These headers are placed within the SOAP Header element of either a SOAP
1.1 or SOAP 1.2
      message. </xsd:documentation>
  </xsd:annotation>
  <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
    schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>
  <xsd:import namespace="http://www.w3.org/2003/05/soap-envelope"
    schemaLocation="http://www.w3.org/2003/05/soap-envelope"/>
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>

  <xsd:element name="Messaging" type="Messaging"/>
  <xsd:complexType name="Messaging">
    <xsd:annotation>
      <xsd:documentation xml:lang="en"> The eb:Messaging element is the top
element of ebMS-3
        headers, and it is placed within the SOAP Header element (either SOAP
1.1 or SOAP
        1.2). The eb:Messaging element may contain several instances of
eb:SignalMessage and
        eb:UserMessage elements. However in the core part of the ebMS-3
specification, only
        one instance of either eb:UserMessage or eb:SignalMessage must be
present. The
        second part of ebMS-3 specification may need to include multiple
instances of either
        eb:SignalMessage, eb:UserMessage or both. Therefore, this schema is
allowing
        multiple instances of eb:SignalMessage and eb:UserMessage elements
for part 2 of the
        ebMS-3 specification. Note that the eb:Messaging element cannot be
empty (at least
        one of eb:SignalMessage or eb:UserMessage element must present).
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="SignalMessage" type="SignalMessage" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element name="UserMessage" type="UserMessage" minOccurs="0"
maxOccurs="unbounded"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
```

```
        <xsd:attributeGroup ref="tns:headerExtension"/>
    </xsd:complexType>
    <xsd:complexType name="SignalMessage">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> In the core part of ebMS-3
specification, an eb:Signal
                Message is allowed to contain eb:MessageInfo and at most one Receipt
Signal, at most
                one eb:PullRequest element, and/or a series of eb:Error elements. In
part 2 of the
                ebMS-3 specification, new signals may be introduced, and for this
reason, an
                extensibility point is added here to the eb:SignalMessage element to
allow it to
                contain any elements. </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element ref="MessageInfo"/>
            <xsd:element name="PullRequest" type="PullRequest" minOccurs="0"/>
            <xsd:element name="Receipt" type="Receipt" minOccurs="0"/>
            <xsd:element name="Error" type="Error" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Error">
        <xsd:sequence>
            <xsd:element name="Description" type="tns:Description" minOccurs="0"/>
            <xsd:element name="ErrorDetail" type="xsd:token" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="category" type="xsd:token" use="optional"/>
        <xsd:attribute name="refToMessageInError" type="xsd:token" use="optional"/>
        <xsd:attribute name="errorCode" type="xsd:token" use="required"/>
        <xsd:attribute name="origin" type="xsd:token" use="optional"/>
        <xsd:attribute name="severity" type="xsd:token" use="required"/>
        <xsd:attribute name="shortDescription" type="xsd:token" use="optional"/>
    </xsd:complexType>
    <xsd:complexType name="PullRequest">
        <xsd:sequence>
            <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="pullAttributes"/>
    </xsd:complexType>
    <xsd:complexType name="Receipt">
        <xsd:sequence>
            <xsd:any namespace="##other" processContents="lax"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="UserMessage">
        <xsd:sequence>
            <xsd:element ref="MessageInfo"/>
            <xsd:element ref="PartyInfo"/>
            <xsd:element ref="CollaborationInfo"/>
            <xsd:element ref="MessageProperties" minOccurs="0"/>
            <xsd:element ref="PayloadInfo" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="mpc" type="xsd:anyURI" use="optional"/>
    </xsd:complexType>
    <xsd:element name="MessageInfo" type="MessageInfo"/>
    <xsd:complexType name="MessageInfo">
        <xsd:sequence>
            <xsd:element name="Timestamp" type="xsd:dateTime"/>
            <xsd:element name="MessageId" type="tns:non-empty-string"/>
            <xsd:element name="RefToMessageId" type="tns:non-empty-string"
minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="PartyInfo" type="PartyInfo"/>
```

```
   <xsd:complexType name="PartyInfo">
      <xsd:sequence>
         <xsd:element name="From" type="tns:From"/>
         <xsd:element name="To" type="tns:To"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="PartyId">
      <xsd:simpleContent>
         <xsd:extension base="tns:non-empty-string">
            <xsd:attribute name="type" type="tns:non-empty-string"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="From">
      <xsd:sequence>
         <xsd:element name="PartyId" type="tns:PartyId" maxOccurs="unbounded"/>
         <xsd:element name="Role" type="tns:non-empty-string"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="To">
      <xsd:sequence>
         <xsd:element name="PartyId" type="tns:PartyId" maxOccurs="unbounded"/>
         <xsd:element name="Role" type="tns:non-empty-string"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="CollaborationInfo" type="CollaborationInfo"/>
   <xsd:complexType name="CollaborationInfo">
      <xsd:sequence>
         <xsd:element name="AgreementRef" type="tns:AgreementRef" minOccurs="0"/>
         <xsd:element name="Service" type="tns:Service"/>
         <xsd:element name="Action" type="xsd:token"/>
         <xsd:element name="ConversationId" type="xsd:token"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Service">
      <xsd:simpleContent>
         <xsd:extension base="tns:non-empty-string">
            <xsd:attribute name="type" type="tns:non-empty-string"
use="optional"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="AgreementRef">
      <xsd:simpleContent>
         <xsd:extension base="tns:non-empty-string">
            <xsd:attribute name="type" type="tns:non-empty-string"
use="optional"/>
            <xsd:attribute name="pmode" type="tns:non-empty-string"
use="optional"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:element name="PayloadInfo" type="PayloadInfo"/>
   <xsd:complexType name="PayloadInfo">
      <xsd:sequence>
         <xsd:element name="PartInfo" type="tns:PartInfo" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="PartInfo">
      <xsd:sequence>
         <xsd:element name="Schema" type="tns:Schema" minOccurs="0"/>
         <xsd:element name="Description" type="tns:Description" minOccurs="0"/>
         <xsd:element name="PartProperties" type="tns:PartProperties"
minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="href" type="xsd:token"/>
   </xsd:complexType>
   <xsd:complexType name="Schema">
      <xsd:attribute name="location" type="xsd:anyURI" use="required"/>
      <xsd:attribute name="version" type="tns:non-empty-string" use="optional"/>
      <xsd:attribute name="namespace" type="tns:non-empty-string" use="optional"/>
```

```
    </xsd:complexType>
    <xsd:complexType name="Property">
        <xsd:simpleContent>
            <xsd:extension base="tns:non-empty-string">
                <xsd:attribute name="name" type="tns:non-empty-string"
use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="PartProperties">
        <xsd:sequence>
            <xsd:element name="Property" type="tns:Property" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="MessageProperties" type="MessageProperties"/>
    <xsd:complexType name="MessageProperties">
        <xsd:sequence>
            <xsd:element name="Property" type="Property" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:attributeGroup name="headerExtension">
        <xsd:attribute name="id" type="xsd:ID" use="optional"/>
        <xsd:attribute ref="S11:mustUnderstand" use="optional">
            <xsd:annotation>
                <xsd:documentation>If SOAP 1.1 is being used, this attribute is
required,
                    other SOAP 1.1 attributes are allowed and SOAP 1.2 attributes are
prohibited.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute ref="S11:encodingStyle"/>
        <xsd:attribute ref="S11:actor"/>
        <xsd:attribute ref="S12:mustUnderstand" use="optional">
            <xsd:annotation>
                <xsd:documentation>If SOAP 1.2 is being used, this attribute is
required,
                    other SOAP 1.2 attributes are allowed and SOAP 1.1 attributes are
prohibited.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute ref="S12:encodingStyle"/>
        <xsd:attribute ref="S12:relay"/>
        <xsd:attribute ref="S12:role"/>
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:attributeGroup>
    <xsd:attributeGroup name="pullAttributes">
        <xsd:attribute name="mpc" type="xsd:anyURI" use="optional"/>
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:attributeGroup>
    <xsd:complexType name="Description">
        <xsd:simpleContent>
            <xsd:extension base="tns:non-empty-string">
                <xsd:attribute ref="xml:lang" use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:simpleType name="non-empty-string">
        <xsd:restriction base="xsd:string">
            <xsd:minLength value="1"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

Appendix C Reference Parameter

The normative schema that is defined for the WS-Addressing reference parameter using [XMLSCHEMA-P1] and [XMLSCHEMA-P2] is located at:

```
http://docs.oasis-open.org/ebxml-msg/...
```

The following copy is provided for reference:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://docs.oasis-open.org/ebxml-
msg/ns/ebms/v3.0/multihop/200902/"
    xmlns:ebint="http://docs.oasis-open.org/ebxml-
msg/ns/ebms/v3.0/multihop/200902/"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    targetNamespace="http://docs.oasis-open.org/ebxml-
msg/ns/ebms/v3.0/multihop/200902/">

    <xs:import namespace="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/"
        schemaLocation="ebms-header-3_0-200704_refactored.xsd"/>

    <xs:import namespace="http://www.w3.org/2005/08/addressing"
        schemaLocation="http://www.w3.org/2002/ws/addr/ns/ws-addr"/>

    <xs:import namespace="http://www.w3.org/2003/05/soap-envelope"
        schemaLocation="http://www.w3.org/2003/05/soap-envelope"/>

    <xs:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
        schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>

    <xs:element name="RoutingInput" type="RoutingInput"/>
    <xs:complexType name="RoutingInput">
        <xs:sequence>
            <xs:element name="UserMessage">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="eb3:MessageInfo" minOccurs="0"/>
                        <xs:element ref="eb3:PartyInfo"/>
                        <xs:element ref="eb3:CollaborationInfo"/>
                        <xs:element ref="eb3:MessageProperties"
minOccurs="0"/>
                        <xs:element ref="eb3:PayloadInfo" minOccurs="0"/>
                    </xs:sequence>
                    <xs:attribute name="mpc" type="xs:anyURI"
use="optional"/>
                </xs:complexType>
            </xs:element>
            <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="optional"/>
        <xs:attribute ref="wsa:IsReferenceParameter" fixed="true"/>
        <xs:attributeGroup ref="S12atts"/>
        <xs:attributeGroup ref="S11atts"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>

    <xs:attributeGroup name="S12atts">
        <xs:attribute ref="S12:mustUnderstand" use="optional">
            <xs:annotation>
```

```
                <xs:documentation> if SOAP 1.2 is being used, this
attribute is required, other
                    attributes in the S12atts group are allowed and
attributes in the S11atts group
                    are prohibited.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute ref="S12:encodingStyle"/>
        <xs:attribute ref="S12:relay"/>
        <xs:attribute ref="S12:role"
            fixed="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"/>
    </xs:attributeGroup>

    <xs:attributeGroup name="S11atts">
        <xs:attribute ref="S11:mustUnderstand" use="optional">
            <xs:annotation>
                <xs:documentation> if SOAP 1.1 is being used, this
attribute is required, other
                    attributes in the S11atts group are allowed and
attributes in the S12atts group
                    are prohibited.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute ref="S11:encodingStyle"/>
        <xs:attribute ref="S11:actor"
            fixed="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"/>
    </xs:attributeGroup>
</xs:schema>
```

Appendix D Flow Diagrams for Multi-hop

D.1 Reliable Sequence Establishment

NOTE: the following diagram shows sequence establishment when using the WS-ReliableMessaging standard. In WS-Reliability, sequence establishment does not require a distinct procedure.

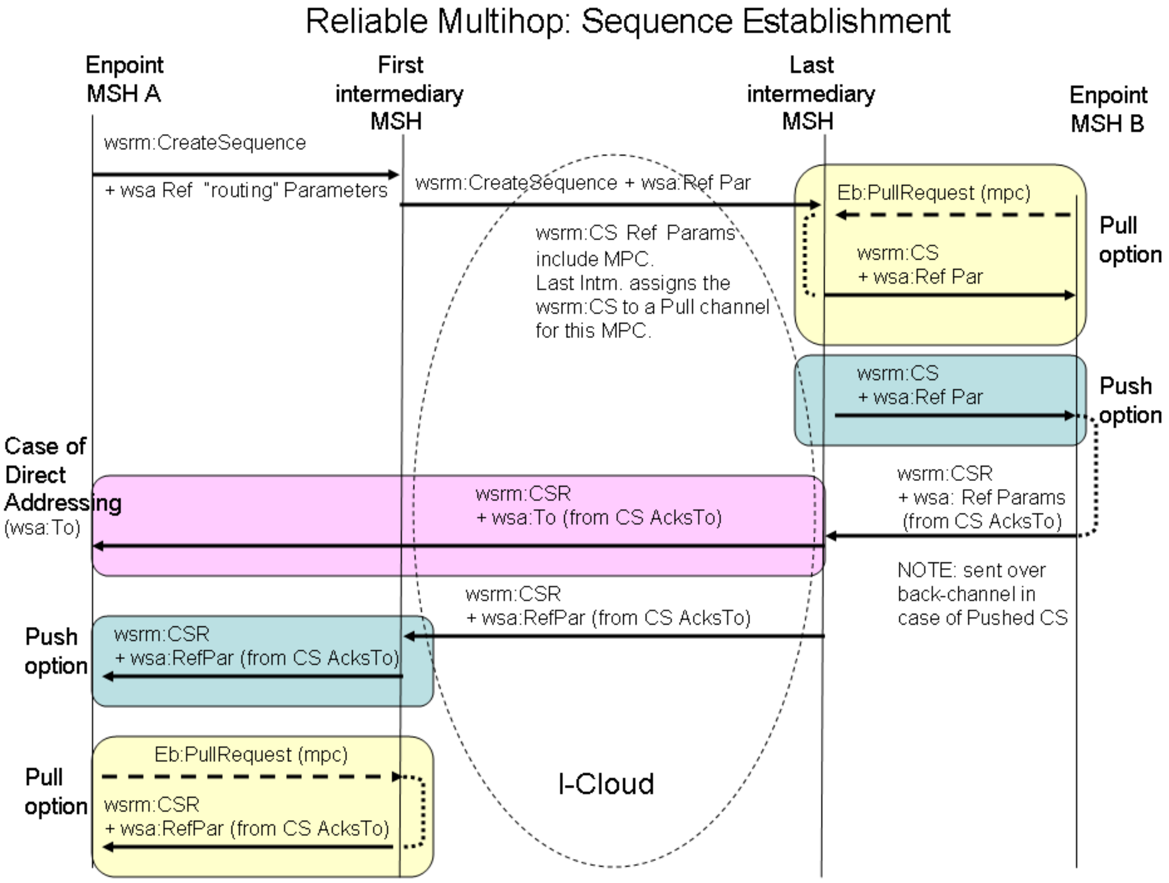
Reliable Multihop: Sequence Establishment
Enpoint MSH A
First intermediary MSH
Last intermediary MSH
Enpoint MSH B
wsrm:CreateSequence
+ wsa Ref "routing" Parameters
wsrm:CreateSequence + wsa:Ref Par
Eb:PullRequest (mpc)
wsrm:CS
+ wsa:Ref Par
Pull option
wsrm:CS Ref Params include MPC. Last Intm. assigns the wsrm:CS to a Pull channel for this MPC.
wsrm:CS
+ wsa:Ref Par
Push option
Case of Direct Addressing (wsa:To)
wsrm:CSR
+ wsa:To (from CS AcksTo)
wsrm:CSR
+ wsa: Ref Params (from CS AcksTo)
NOTE: sent over back-channel in case of Pushed CS
wsrm:CSR
+ wsa:RefPar (from CS AcksTo)
Push option
wsrm:CSR
+ wsa:RefPar (from CS AcksTo)
Pull option
Eb:PullRequest (mpc)
wsrm:CSR
+ wsa:RefPar (from CS AcksTo)
I-Cloud

Figure 20: Reliable Multi-hop: Sequence Establishment

The following steps are taken:

- **Step 1**: The Sending party submits a User message to its endpoint MSH A. The MSH resolves which PMode / CPA is associated with this message so that it knows its processing mode (which level of security, reliability, MEP…). In this case, the PMode requires that its related messages need be sent reliably, i.e. that an RM sequence needs be initiated for these.

- **Step 2**: The Sending endpoint MSH A determines where to send this message – here to an ebMS Intermediary – by getting the Intermediary URL from its PMode / CPA configuration

- **Step 3**: MSH A is initiating a `wsrm:CreateSequence` message to the Intermediary (unless an RM sequence already exists for this PMode / CPA). Because the PMode also contains a WS-Addressing EPR for the destination, including the Reference Parameter `ebint:RoutingInput` that replicates a significant subset of the ebMS header data associated with this PMode (data that will be common to all User Messages sent for this PMode), the MSH piggybacks this Reference Parameter on the `wsrm:CreateSequence` message for routing purpose.

- NOTE: because the destination URL is unknown (dynamically resolved by I-Cloud routing), the destination EPR is set to the I-Cloud URI: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/icloud, which has no special meaning in case of a SOAP Request. The header `wsa:To` is present in the wsrm:CS message and contains the I-Cloud URI.

- **Step 4:** The First Intermediary receives the `wsrm:CreateSequence` message, and forwards it using a routing function that uses ebMS header data (regardless whether this data is wrapped into an ebMS header or into the `ebint:RoutingInput` wsa Ref Parameters headers). In this case the routing data is in wsa Reference Parameters.

- **Step 5**: At the end of the routing path, the Last Intermediary gets the `wsrm:CreateSequence` message. Two options must be supported depending on the MEP required by the destination endpoint (MSH B):

  1. **Push MEP**: The Last Intermediary keeps routing the wsrm:CS in a push mode to the destination endpoint (MSH B). The content of `wsrm:AcksTo` will determine how the wsrm:CSR is sent back.

  2. **Pull MEP**: The Last Intermediary is aware that the reference parameter associated with the wsrm:CS calls for a Pull mode. Among these parameters, is a mention of the MPC where messages for this sequence will be pulled from. The Intermediary MAY piggyback on the wsrm:CS a dummy eb3:Header with a non-effective Service value (`http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service)`. The intermediary MUST assign the wsrm:CS message to the related MPC so that the message becomes subject to pulling using `eb3:PullRequest` targeted to this MPC.

- ***EDITOR NOTE:*** *an alternative could have used wsmc:MakeConnection to pull such signals. However, the use of MC would not be able to use the most standard cases (pulling based on sequence ID, and pulling based on `wsa:Address`) due to the multi-hop context. Consequently MC extensibility points would have to be used, to pull based on MPC. Because of this level of customization there is little advantage in using the wsmc standard, and it becomes simpler and more reliable to rely on a unique pulling mechanism – `eb3:PullRequest` – for all messages related to a sequence.*

- **Step 6**: The RM module in MSH B takes knowledge of the `wsrm:AcksTo` element contained in the wsrm:CS message. The `wsrm:AcksTo` element value indicates the EPR where the wsrm:CSR must be sent back. Indeed, in the ebMS multi-hop model, RM lifecycle response messages such as CSR must be sent to the same destination as RM acknowledgment. The `wsrm:AcksTo` element is itself an EPR that sufficiently identifies the initial Sending MSH so that the I-Cloud can route the wsrm:CSR back to MSH A. The following cases need to be supported, depending on how the wsrm:CS was transmitted:

  - **Pushed wsrm:CS:** The wsrm:CSR is sent back based on the `wsrm:AcksTo` content. If it were an anonymous URI, the CSR is sent over the backchannel of the wsrm:CS, from the RM module of MSH B to the Last Intermediary. In all cases, the `ebint:RoutingInput` Reference Parameter, if present in the AcksTo EPR, is added to the wsrm:CSR message. The routing of wsrm:CSR is based on this reference parameter. In a special case where the AcksTo gives the URL of the destination (MSH A) and this destination can directly be resolved without ebMS-level routing, the RM module of MSH B can directly send it back to MSH A.

  - **Pulled wsrm:CS:** The wsrm:CSR is sent over a new HTTP connection to the I-Cloud. The `ebint:RoutingInput` Reference Parameter in the AcksTo EPR is added to the wsrm:CSR for routing.

  - NOTE: In both cases the wsrm:CSR MAY be sent to a node of the I-Cloud other than the last Intermediary involved in routing the wsrm:CS message. This depends on how the AcksTo EPR is to be resolved by the I-Cloud.

- **Step 7**: In case the routing of wsrm:CSR involves the initial First Intermediary, the latter gets the wsrm:CSR with sequence ID. Here, the same procedure as for Step 5 takes place for transmitting the wsrm:CSR to MSH A, allowing for both Push and Pull options.

In the above process, it must be noted that:

- the endpoint MSHs have the ability to insert and parse WS-addressing EPRs.
- The ebMS intermediary in contact with the destination endpoint MSH, must act in accordance of the PMode unit that governs its edge-hop, especially in case of message pulling from the endpoint.

D.2 Routing RM Acknowledgments

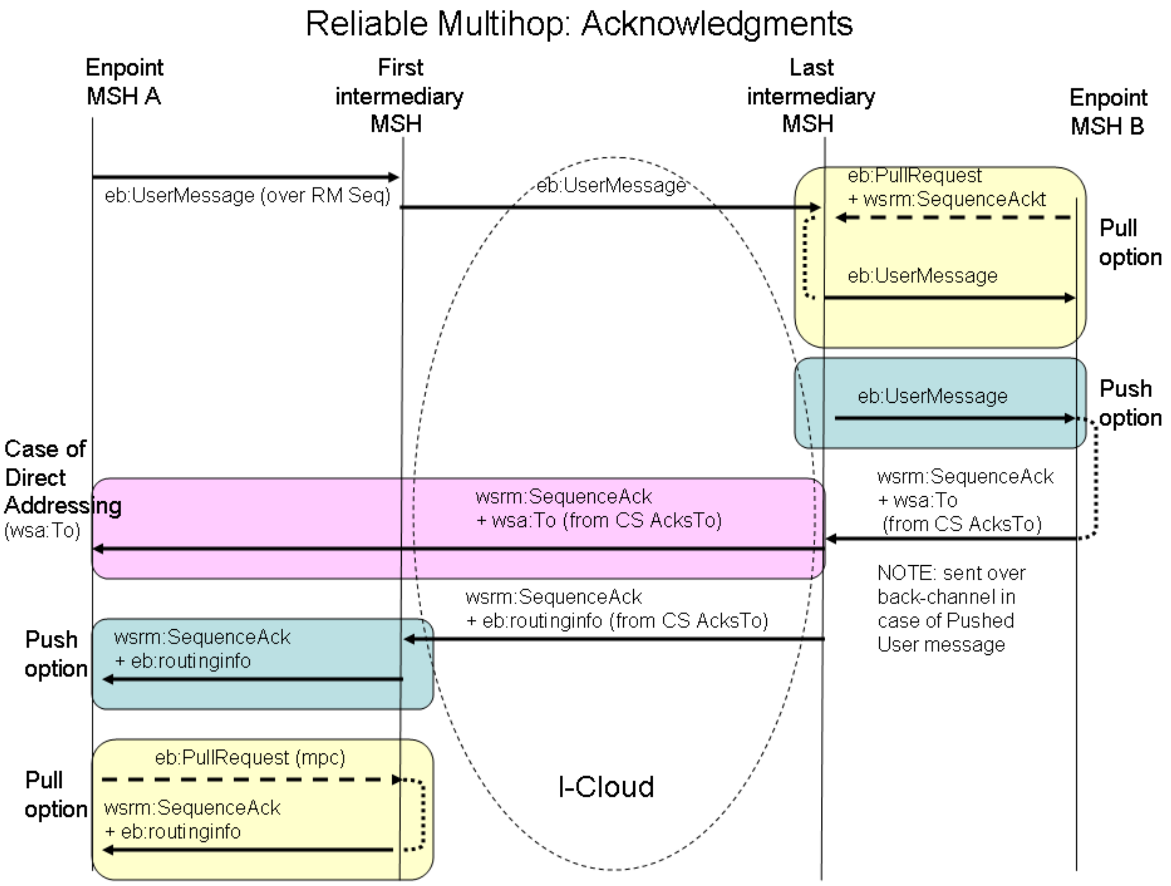
Reliable Multihop: Acknowledgments
Enpoint MSH A
First intermediary MSH
Last intermediary MSH
Enpoint MSH B
eb:UserMessage (over RM Seq)
eb:UserMessage
eb:PullRequest + wsrm:SequenceAckt
eb:UserMessage
Pull option
eb:UserMessage
Push option
Case of Direct Addressing (wsa:To)
wsrm:SequenceAck + wsa:To (from CS AcksTo)
wsrm:SequenceAck + wsa:To (from CS AcksTo)
NOTE: sent over back-channel in case of Pushed User message
wsrm:SequenceAck + eb:routinginfo (from CS AcksTo)
Push option
wsrm:SequenceAck + eb:routinginfo
Pull option
eb:PullRequest (mpc)
wsrm:SequenceAck + eb:routinginfo
I-Cloud

Figure 21: Reliable Multihop: Acknowledgments

- **Step 1**: The Sending party submits a User message to its endpoint MSH A. The MSH resolves which PMode / CPA is associated with this message so that it knows its processing mode (P-Mode) In this case, the P-Mode is already associated with a Reliable Messaging (RM) sequence.
- **Step 2**: The Sending endpoint MSH A determines where to send this message – here to an ebMS Intermediary
- **Step 3:** The First Intermediary receives the User message, and forwards it using a routing function that uses ebMS header data.
- **Step 4**: At the end of the routing path, the Last Intermediary gets the User message. Two options must be supported depending on the MEP required by the destination endpoint (MSH B):
    1. **Push MEP**: The Last Intermediary keeps routing the User message in a push mode to the destination endpoint (MSH B).

2. **Pull MEP**: The Last Intermediary is aware that the P-Mode associated with the User message calls for a Pull mode. The message is assigned to a Pull channel (MPC).

- **Step 5**: The RM module in MSH B sends back a `wsrm:SequenceAcknowledgment` based on the content of `wsrm:AcksTo`. It will be routed in the same way as the wsrm:CSR during the sequence establishment. In case where the User message was pulled from the last Intermediary, and if the resolution of `wsrm:AcksTo` is compatible with using this last Intermediary for routing acknowledgments, then the `wsrm:SequenceAcknowledgment` header may be piggybacked on the `eb3:PullRequest` message.

- **Step 6**: In case the routing of `wsrm:SequenceAcknowledgment` involves the initial First Intermediary, the `wsrm:SequenceAcknowledgment` message is subject to the same Push / Pull alternative as in Step 4. In case of a Pull configuration, the Intermediary will assign the `wsrm:SequenceAcknowledgment` to a Pull channel. The acknowledgment will be associated with an `eb3:Error` of type warning (EBMS:0006) unless it is piggybacked with another eb message for this channel.

D.3 Routing User Messages (One-Way MEP)

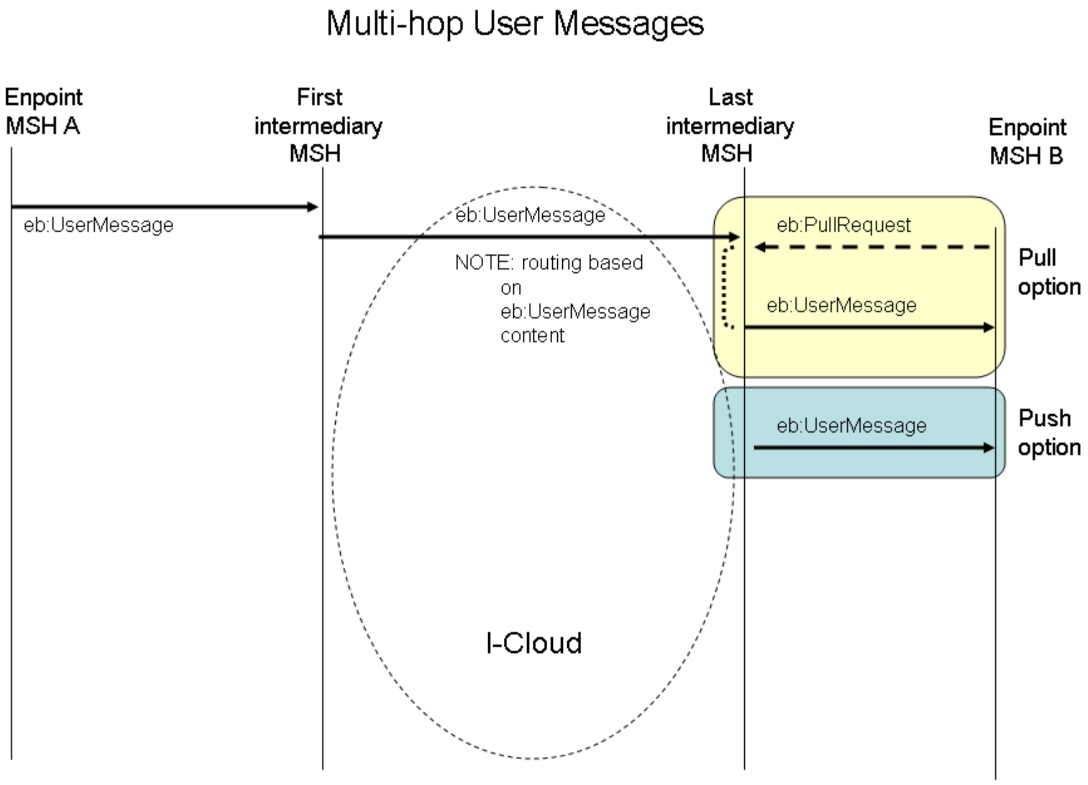

Figure 22: Routing User Messages (One Way MEP)

This case only concerns User messages that are pushed by the Sending MSH. This flow has been described in sufficient details in the previous sections. Routing is expected to be based on the UserMessage content. However the presence of a RoutingInput reference parameter is not excluded, in which case the latter takes precedence.

The routing across the I-Cloud may involve both pushing and pulling. Both push and pull cases are illustrated on the Receiving side.

D.4 Routing ebMS Reply User Messages (Two-way MEP)

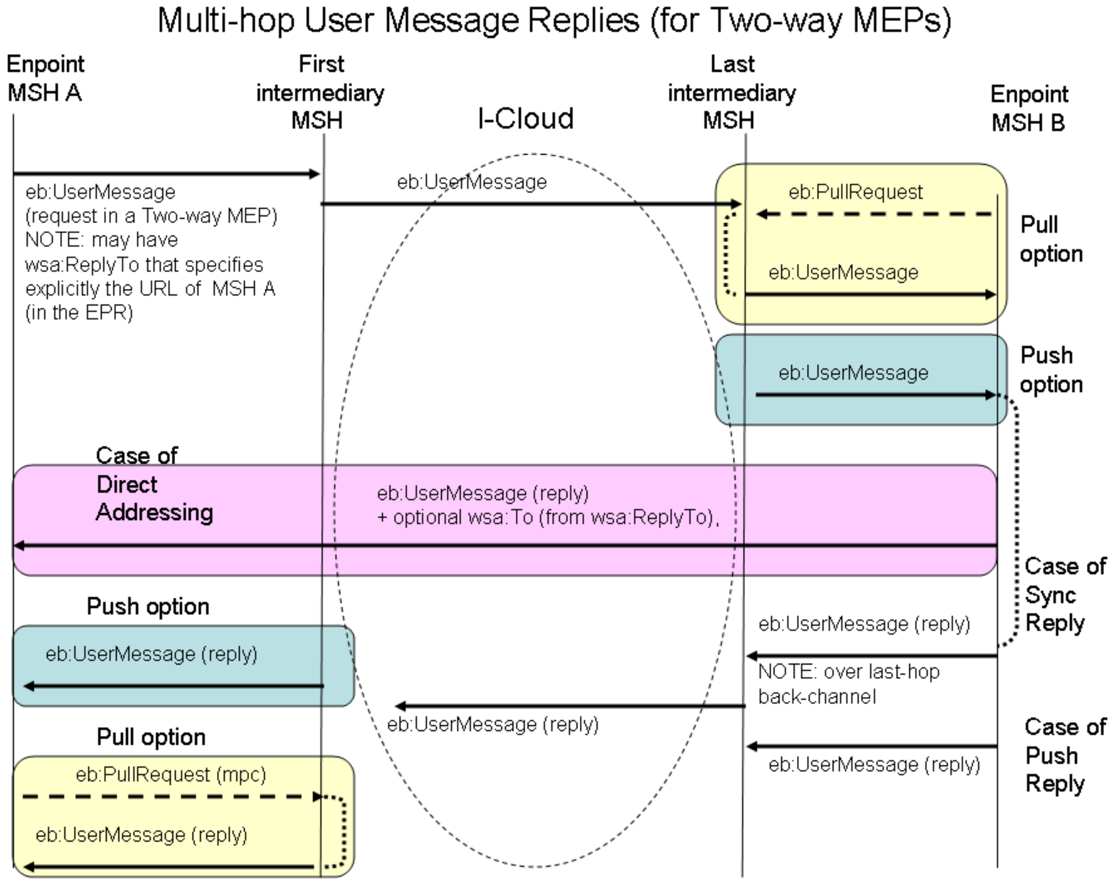

Figure 23: Routing User Messages (Two Way MEP)

The above case illustrates a Two-Way exchange with the “request” message being pushed by the Sending MSH. The diagram illustrates the different ways the response (“reply” message in the Two-Way) can be sent back:

1. Direct addressing, by-passing the I-Cloud routing in case the response address is explicitly provided (e.g. by `wsa:ReplyTo`) and can be resolved without ebMS-level routing.
2. Reply message sent back over the back-channel of the “request” message (case where the P-Mode.MEPbinding is “Sync”, for a request-response underlying transport such as HTTP).
3. Reply message sent back as callback (case where the P-Mode.MEPbinding is “Push-and-Push”, e.g. over a new HTTP connection).

This flow has been described in sufficient details in the previous sections. Routing is expected to be based on the UserMessage content, for both request and reply messages. However the presence of a RoutingInput reference parameter is not excluded, in which case the latter takes precedence.The routing across the I-Cloud may involve both pushing and pulling. On the receiving side of the reply message as well as of the request message, both push and pull options are illustrated.

D.5 Routing ebMS Response Signals (Receipts, Errors)

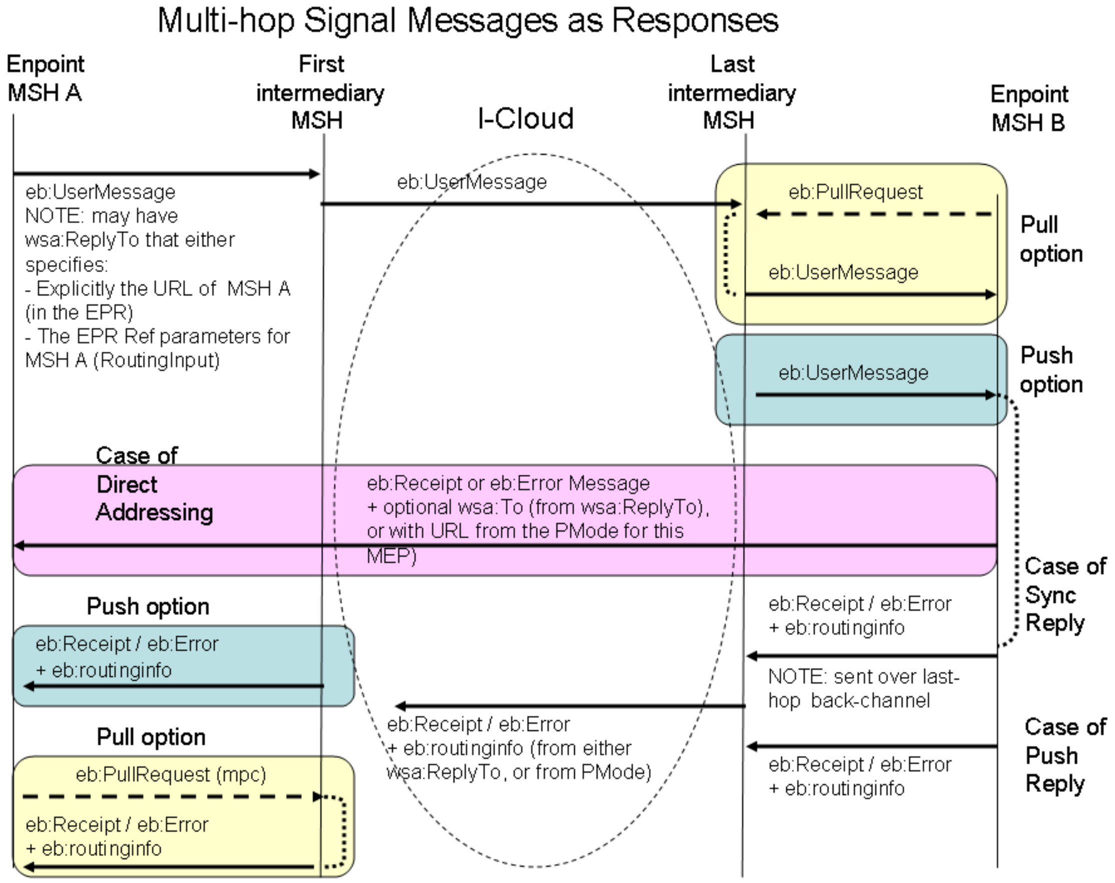
Multi-hop Signal Messages as Responses
Enpoint MSH A
First intermediary MSH
I-Cloud
Last intermediary MSH
Enpoint MSH B
eb:UserMessage
NOTE: may have wsa:ReplyTo that either specifies:
- Explicitly the URL of MSH A (in the EPR)
- The EPR Ref parameters for MSH A (RoutingInput)
eb:UserMessage
eb:PullRequest
eb:UserMessage
Pull option
eb:UserMessage
Push option
Case of Direct Addressing
eb:Receipt or eb:Error Message + optional wsa:To (from wsa:ReplyTo), or with URL from the PMode for this MEP)
Case of Sync Reply
eb:Receipt / eb:Error + eb:routinginfo
NOTE: sent over last-hop back-channel
Push option
eb:Receipt / eb:Error + eb:routinginfo
eb:Receipt / eb:Error + eb:routinginfo (from either wsa:ReplyTo, or from PMode)
Case of Push Reply
eb:Receipt / eb:Error + eb:routinginfo
Pull option
eb:PullRequest (mpc)
eb:Receipt / eb:Error + eb:routinginfo

Figure 24: Routing ebMS Response Signals

The routing of ebMS signals that are responding to previous ebMS messages (i.e. `eb3:Receipt`s, `eb3:Error`s) has been described in section .1.7.3 (“Routing support for Response Messages”). The options are similar to the way “reply” user messages are sent back, except that the routing input must be provided as a WS-Addressing reference parameter header block.

NOTE: `eb3:Error` signals may also be sent back by Intermediaries (EBMS:0020 (RoutingFailure),) which has not been illustrated here. The routing of such Intermediary errors back to the message originator would need to be supported either by the routing function in a specific way, or by dynamic routing input provided in `wsa:ReplyTo` header of the failing message.

Appendix E Sample Message Exchange: One Way Exchange with Asynchronous Receipt

This annex as well as the next two annexes contain some sample message exchanges across intermediaries. They highlight the issues related to intermediaries, such as the use of the `ebint:RoutingInput` reference parameter and WS-Addressing headers. The first example contains a `wsse:Security` header, but the timestamp, security token and signature values of the security header are omitted and digest values may be incorrect. The internal structure of the security header and all payload content is omitted in the other examples.

The first example is a One-Way push MEP across a single intermediary, followed by a `Receipt` sent back via the same intermediary, on separate asynchronous HTTP connections. The assumption is that both endpoints are addressable from the intermediary. The intermediary pushes all messages to next hop URLs using a routing function based on `eb3:To/eb3:PartyId`. These URLs are the URLs of the endpoints, as this topology is limited to a single intermediary.

E.1 Message from Buyer to Intermediary

The first message in this exchange is a regular ebMS 3.0 user message. The only visible sign of communication via an intermediary is that the message is posted to http://intermediary.example.com:4081/reroute, the URL of an intermediary, instead of the URL of the business partner. If the example had used TLS, the TLS server certificate would similarly be the intermediary TLS certificate.

WS-Security is used to time stamp the SOAP message, sign the `eb3:Messaging` element, the attached payload document and the empty `S12:Body`.

```
POST /reroute HTTP/1.1
Host: intermediary.example.com:4081
Content-Type: multipart/Related; type=application/soap+xml; action=''; charset="utf-
8"; boundary=f1fad5ca-f6b1-4c1b-ba46-099321af7cbe; start="<d201cab1-198e-49b1-8988-
f55161de3b57@buyer.example.com>"

--f1fad5ca-f6b1-4c1b-ba46-099321af7cbe
Content-Type: application/soap+xml; charset=utf-8
Content-Transfer-Encoding: 8bit
Content-ID: <d201cab1-198e-49b1-8988-f55161de3b57@buyer.example.com>

<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
    <S12:Header>
        <eb3:Messaging S12:mustUnderstand="true"
            id="_5cb44655-5720-4cf4-a772-19cd480b0ad4"
            S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh">
            <eb3:UserMessage mpc="e5c31ef7-d750-4db8-b4dc-13a751d80b9a">
                <eb3:MessageInfo>
                    <eb3:Timestamp>2009-05-21T10:49:28.886Z</eb3:Timestamp>
                    <eb3:MessageId>orders123@buyer.example.com</eb3:MessageId>
                </eb3:MessageInfo>
                <eb3:PartyInfo>
                    <eb3:From>
                        <eb3:PartyId
                            type="urn:oasis:names:tc:ebcore:partyid-
type:iso6523:0002"
                            >123456789</eb3:PartyId>
                        <eb3:Role>Buyer</eb3:Role>
                    </eb3:From>
                    <eb3:To>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0106"
                            >192837465</eb3:PartyId>
                        <eb3:Role>Seller</eb3:Role>
                    </eb3:To>
                </eb3:PartyInfo>
                <eb3:CollaborationInfo>
                    <eb3:Service>Sales</eb3:Service>
                    <eb3:Action>ProcessPurchaseOrder</eb3:Action>
                    <eb3:ConversationId>ecae53d4-7473-45a6-ad70-
61970dd7c4b0</eb3:ConversationId>
                </eb3:CollaborationInfo>
                <eb3:PayloadInfo>
                    <eb3:PartInfo href="cid:a1d7fdf5-d67e-403a-ad92-
3b9deff25d43@buyer.example.com"
                    />
                </eb3:PayloadInfo>
            </eb3:UserMessage>
        </eb3:Messaging>
        <wsse:Security S12:mustUnderStand="true">
            <wsu:Timestamp wsu:Id="_476193ac-1584-48ac-9074-143a8fc13523">
                <!-- details omitted -->
            </wsu:Timestamp>
            <wsse:BinarySecurityToken wsu:Id="_43336c8b-38bd-470d-94c1-
165ab96b57a5">
                <!-- details omitted -->
            </wsse:BinarySecurityToken>
            <ds:Signature>
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#_5cb44655-5720-4cf4-a772-19cd480b0ad4">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>KshAH7QFFAw2sV5LQBOUOSSrCaI=</ds:DigestValue
>
                    </ds:Reference>
                    <ds:Reference URI="#_f8aa8b55-b31c-4364-94d0-3615ca65aa40">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>l2HxWizaP7d43f4hATCD+O7it1c=</ds:DigestValue
>
                    </ds:Reference>
                    <ds:Reference URI="cid:a1d7fdf5-d67e-403a-ad92-
3b9deff25d43@buyer.example.com">
                        <ds:Transforms>
                            <ds:Transform
                                Algorithm="http://docs.oasis-open.org/wss/oasis-wss-
SwAProfile-1.1#Attachment-Content-Signature-Transform"
                            />
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>iWNSv2W6SxbOYZliPzZDcXAxrwI=</ds:DigestValue
>
                    </ds:Reference>
                    <ds:Reference URI="#_476193ac-1584-48ac-9074-143a8fc13523">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
```

```
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>PreCqm0ESZqmITjf1qzrLFuOEYg=</ds:DigestValue
>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    <!-- details omitted -->
                </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#_43336c8b-38bd-470d-94c1-165ab96b57a5"
                            ValueType="http://docs.oasisopen.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3"
                        />
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="_f8aa8b55-b31c-4364-94d0-3615ca65aa40"/>
</S12:Envelope>

--f1fad5ca-f6b1-4c1b-ba46-099321af7cbe
       Content-Type: text/plain
Content-Transfer-Encoding: 7bit
Content-ID: <a1d7fdf5-d67e-403a-ad92-3b9deff25d43@buyer.example.com>
Content-Length: 368

UNB+UNOA:2+ABSENDER901234567890123456789OABCDE:CD-A:WEITERLEITNG-A+EMPFAENGER:CD-
E:WEITERLEITNG-E+940812:0235+T940812023504A++0026-Anwendung'
UNH+M940812023504A+ORDERS:D:93A:UN:EAN007'
BGM+220+5211229'
DTM+002:940815'
NAD+SU+4004353000000::9'
NAD+BY+4306517005214::9'
LIN+1++4004353054099:EN'
QTY+21:9'
UNS+S'
UNT+51+M940812023504A'
UNZ+3+T940812023504A'
--f1fad5ca-f6b1-4c1b-ba46-099321af7cbe

```

E.2 Message from Intermediary to Seller

The intermediary forwards the message without any changes to the SOAP-with-Attachment structure. It only changes the HTTP command.

```
POST /msh HTTP/1.1
Host: buyer.example.com:5030
```

The remainder of the MIME structure is forwarded without modification. (Within the I-Cloud intermediaries could even use SMTP instead of HTTP).

E.3 Message from Seller to Intermediary

Based on P-mode, Seller sends an `eb3:SignalMessage` containing an `eb3:Receipt` for the message displayed in E.1 . The receipt has `ds:Reference`s for all structures signed by the WS-Security header in the received message. To be routed through the I-Cloud, an `ebint:RoutingInput` element is added. In this response message, WS-Security is used to sign the `ebint:RoutingInput` and `eb3:Messaging` structures, the empty `S12:Body` element and the WS-Addressing headers. The following diagram only shows the SOAP envelope.

```
<S12:Envelope
    xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:ebint="http://docs.oasis-open.org/ebxml-
msg/ns/ebms/v3.0/multihop/200902/"
    xmlns:ebbp="http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#" >
    <S12:Header>
        <wsa:To wsu:Id="_25ca74aa-0a9f-454d-a5fe-5231ba8304ed"
            S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"
            S12:mustUnderstand="true"
            >http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/icloud</wsa:To>
        <wsa:Action wsu:Id="_42924487-4099-4600-a0e7-4156022320a6"
            >http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/oneWay.receipt</wsa:Action>
        <ebint:RoutingInput wsa:IsReferenceParameter="true"
            id="_ccd050c7-a1a5-4c31-8c01-e3c2534609ab" S12:mustUnderstand="true"
            S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh">
            <ebint:UserMessage mpc="e5c31ef7-d750-4db8-b4dc-13a751d80b9a.receipt">
                <eb3:PartyInfo>
                    <eb3:From>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0106"
                            >192837465</eb3:PartyId>
                        <eb3:Role>Seller</eb3:Role>
                    </eb3:From>
                    <eb3:To>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0002"
                            >123456789</eb3:PartyId>
                        <eb3:Role>Buyer</eb3:Role>
                    </eb3:To>
                </eb3:PartyInfo>
                <eb3:CollaborationInfo>
                    <eb3:Service>Sales</eb3:Service>
                    <eb3:Action>ProcessPurchaseOrder.receipt</eb3:Action>
                    <eb3:ConversationId>ecae53d4-7473-45a6-ad70-
61970dd7c4b0</eb3:ConversationId>
                </eb3:CollaborationInfo>
            </ebint:UserMessage>
        </ebint:RoutingInput>
        <eb3:Messaging S12:mustUnderstand="true" id="_393bb2e2-df86-4b4f-b682-
f7a684316b3d">
            <eb3:SignalMessage>
                <eb3:MessageInfo>
                    <eb3:Timestamp>2009-05-22T14:33:11.735Z</eb3:Timestamp>
                    <eb3:MessageId>orderreceipt@seller.example.com</eb3:MessageId>
                    <eb3:RefToMessageId>orders123@buyer.example.com</eb3:RefToMessa
geId>
                </eb3:MessageInfo>
                <eb3:Receipt>
                    <ebbp:NonRepudiationInformation>
                        <ebbp:MessagePartNRInformation>
                            <ds:Reference URI="#_5cb44655-5720-4cf4-a772-
19cd480b0ad4">
                                <ds:Transforms>
                                    <ds:Transform
                                        Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                                </ds:Transforms>
                                <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                                <ds:DigestValue>KshAH7QFFAw2sV5LQBOUOSSrCaI=</ds:Di
gestValue>
                            </ds:Reference>
                        </ebbp:MessagePartNRInformation>
                        <ebbp:MessagePartNRInformation>
                            <ds:Reference URI="#_f8aa8b55-b31c-4364-94d0-
3615ca65aa40">
                                <ds:Transforms>
                                    <ds:Transform
                                        Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                                </ds:Transforms>
                                <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                                <ds:DigestValue>l2HxWizaP7d43f4hATCD+O7it1c=</ds:Di
gestValue>
                            </ds:Reference>
                        </ebbp:MessagePartNRInformation>
                        <ebbp:MessagePartNRInformation>
                            <ds:Reference
                                URI="cid:a1d7fdf5-d67e-403a-ad92-
3b9deff25d43@buyer.example.com">
                                <ds:Transforms>
                                    <ds:Transform
                                        Algorithm="http://docs.oasis-
open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform"
                                    />
                                </ds:Transforms>
                                <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                                <ds:DigestValue>iWNSv2W6SxbOYZliPzZDcXAxrwI=</ds:Di
gestValue>
                            </ds:Reference>
                        </ebbp:MessagePartNRInformation>
                    </ebbp:NonRepudiationInformation>
                </eb3:Receipt>
            </eb3:SignalMessage>
        </eb3:Messaging>
        <wsse:Security S12:mustUnderStand="true">
            <wsu:Timestamp wsu:Id="_870532b4-2a25-4959-8d5e-029c05f5f6ee">
                <!-- details omitted -->
            </wsu:Timestamp>
            <wsse:BinarySecurityToken wsu:Id="_dc8f9e0f-529f-47eb-b78b-
30ebff44f820">
                <!-- details omitted -->
            </wsse:BinarySecurityToken>
            <ds:Signature>
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#_25ca74aa-0a9f-454d-a5fe-5231ba8304ed">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>ZwNxc91tmwMBFrvVJZRTlyQUseQ=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_42924487-4099-4600-a0e7-4156022320a6">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>XlAi4OIAYdZJftFWacQAVpRDGeg=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_ccd050c7-a1a5-4c31-8c01-e3c2534609ab">
                        <ds:Transforms>
```

```
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>h0vN6yVOXZwsaLoeLVitUdYoKlM=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_393bb2e2-df86-4b4f-b682-f7a684316b3d">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>vYPsc9BmFT8hNrlruSqDNlycZhg=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_861ff127-f930-4934-8ad0-5b91bd6dbc1e">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>3/NPWtNNuYzfWaCfV2oBdYnEDbg=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_870532b4-2a25-4959-8d5e-029c05f5f6ee">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>PreCqm0ESZqmITjf1qzrLFuOEYg=</ds:DigestValu
e>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    <!-- details omitted -->
                </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#_dc8f9e0f-529f-47eb-b78b-
30ebff44f820"
                            ValueType="http://docs.oasisopen.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3"
                        />
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="_861ff127-f930-4934-8ad0-5b91bd6dbc1e"/>
</S12:Envelope>
```

The ebint:RoutingInput reference parameter is filled using the values derived from the eb3:UserMessage structure using the recommended default values.

E.4 Receipt from Intermediary to Buyer

Again, the intermediary only rewrites the HTTP header:

```
POST /msh HTTP/1.1
Host: buyer.example.com
```

E.5 Variant: Routing Error

A variant of this scenario is a situation where the intermediary is unable to forward the message from Buyer to Seller from E.1 and responds with a routing error message:

```
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
    xmlns:ebint="http://docs.oasis-open.org/ebxml-
msg/ns/ebms/v3.0/multihop/200902/" >
    <S12:Header>
        <wsa:To wsu:Id="_79bfe70e-f87f-4b3c-acb6-a19ddc0f9ad5"
            S12:mustUnderstand="true"
            S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"
        >http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/icloud</wsa:To>
        <wsa:Action wsu:Id="_b6760f73-9fa5-48e8-acc0-1fb1784d1a68"
            >http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/oneWay.error
        </wsa:Action>
        <ebint:RoutingInput id="_43336c8b-38bd-470d-94c1-165ab96b57a5"
            S12:mustUnderstand="true"
            S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh">
            <ebint:UserMessage>
                <eb3:PartyInfo>
                    <eb3:From>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0106"
                            >192837465</eb3:PartyId>
                        <eb3:Role>Seller</eb3:Role>
                    </eb3:From>
                    <eb3:To>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0002"
                            >123456789</eb3:PartyId>
                        <eb3:Role>Buyer</eb3:Role>
                    </eb3:To>
                </eb3:PartyInfo>
                <eb3:CollaborationInfo>
                    <eb3:Service>Sales</eb3:Service>
                    <eb3:Action>ProcessPurchaseOrder.error</eb3:Action>
                    <eb3:ConversationId>ecae53d4-7473-45a6-ad70-
61970dd7c4b0</eb3:ConversationId>
                </eb3:CollaborationInfo>
            </ebint:UserMessage>
        </ebint:RoutingInput>
        <eb3:Messaging id="_25ca74aa-0a9f-454d-a5fe-5231ba8304ed"
S12:mustUnderstand="true">
            <eb3:SignalMessage>
                <eb3:MessageInfo>
                    <eb3:Timestamp>2009-05-21T10:49:29.251Z</eb3:Timestamp>
                    <eb3:MessageId>fc9d9e2a-b396-4a9c-89e9-
5f55d7deefa0@intermediary.example.com</eb3:MessageId>
                    <eb3:RefToMessageId>orders123@buyer.example.com</eb3:RefToMessa
geId>
                </eb3:MessageInfo>
                <eb3:Error origin="ebMS" category="InternalProcess"
errorCode="EBMS:0020"
                    shortDescription="RoutingFailure" severity="failure"
                    refToMessageInError="orders123@buyer.example.com">
                    <eb3:Description xml:lang="en"
```

```
>Unable to forward message: no matching routing rule found at
intermediary.example.com for message orders123@buyer.example.com</eb3:Description>
                </eb3:Error>
            </eb3:SignalMessage>
        </eb3:Messaging>
        <wsse:Security  S12:mustUnderStand="true">
            <!-- Omitted -->
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="_30dadeb1-3428-4d1e-92bf-90b28a0adbdc"/>
</S12:Envelope>
```

Appendix F Sample Message Exchange: One Way Exchange using WS-ReliableMessaging

This example is again a one way push message, but this time the exchange has to be reliable and use the WS-ReliableMessaging 1.1 protocol for reliability. This protocol requires the initialization of a reliable sequence using the WS-ReliableMessaging sequence management messages. To route these messages across the I-Cloud, an `ebint:RoutingInput` structure is added. This example is about establishing a sequence with asynchronous acknowledgment messages, and also uses an asynchronous exchange of lifecycle messages.

F.1 CreateSequence

A sending MSH is about to send a user message containing an invoice on behalf of a partner acting as a Seller to the Buyer who previously placed an order. The MSH detects that, for this particular message Pmode, a reliable sequence is needed. As none is available for re-use, it sets one up. An `ebint:RoutingInput` is added and constructed using the relevant P-mode parameters. One other occurrence of the `ebint:RoutingInput` is present in the `wsrm:AcksTo` element to support reverse routing by the recipient of the `CreateSequenceResponse` response message and of the acknowledgments that will use the sequence to be established.

```
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
    xmlns:wsrmp="http://docs.oasis-open.org/ws-rx/wsrmp/200702"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:ebint="http://docs.oasis-open.org/ebxml-
msg/ns/ebms/v3.0/multihop/200902/"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
>
    <S12:Header>
        <wsa:MessageID wsu:Id="_590ae39c-1e12-47ea-8697-fa2be482ff23"
            >http://seller.example.com/b6ca4390-6694-419f-a2c5-
c9fa161ba0aa.wsrm.cs</wsa:MessageID>
        <wsa:Action wsu:Id="_973a9035-f878-41d8-8958-c9fe20ffa820"
            >http://docs.oasis-open.org/ws-
rx/wsrm/200702/CreateSequence</wsa:Action>
        <wsa:To wsu:Id="_d5c147b8-c610-4629-85ee-8c9bfd5f49c0"
S12:mustUnderstand="true"
            S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"
            >http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/icloud</wsa:To>
        <ebint:RoutingInput wsa:IsReferenceParameter="true"
S12:mustUnderstand="true"
            id="_5d657462-b175-4657-86bf-90a231a5f495"
            S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh">
            <ebint:UserMessage mpc="595066c0-77b6-4c0e-95db-3015e335095e">
                <eb3:PartyInfo>
                    <eb3:From>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0106"
                            >192837465</eb3:PartyId>
                        <eb3:Role>Seller</eb3:Role>
                    </eb3:From>
                    <eb3:To>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0002"
                            >123456789</eb3:PartyId>
                        <eb3:Role>Buyer</eb3:Role>
                    </eb3:To>
                </eb3:PartyInfo>
                <eb3:CollaborationInfo>
                    <eb3:Service>Sales</eb3:Service>
                    <eb3:Action>RequestForPayment</eb3:Action>
                    <eb3:ConversationId>b6ca4390-6694-419f-a2c5-
c9fa161ba0aa</eb3:ConversationId>
                </eb3:CollaborationInfo>
            </ebint:UserMessage>
        </ebint:RoutingInput>
        <wsse:Security S12:mustUnderStand="true">
            <wsu:Timestamp wsu:Id="_7222f3fb-018e-4952-9511-841bbc883279">
                <!-- details omitted -->
            </wsu:Timestamp>
            <wsse:BinarySecurityToken wsu:Id="_76d8adbc-0f19-4af3-80e1-
b48e25bdf02c">
                <!-- details omitted -->
            </wsse:BinarySecurityToken>
            <ds:Signature>
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#_590ae39c-1e12-47ea-8697-fa2be482ff23">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>Epfm13Uu/HIfINBz9HvvQ5pNu0E=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_973a9035-f878-41d8-8958-c9fe20ffa820">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>a/kLYjouK9ShHnQgiZz7X3/ofbk=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_d5c147b8-c610-4629-85ee-8c9bfd5f49c0">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>RgDdQIprqVHgTnjwhUAvreoy73w=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_5d657462-b175-4657-86bf-90a231a5f495">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>E71/QH3ev3JnhD4deNCr2Z76S7o=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_f8aa8b55-b31c-4364-94d0-3615ca65aa40">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>PreCqm0ESZqmITjf1qzrLFuOEYg=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_7222f3fb-018e-4952-9511-841bbc883279">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>PreCqm0ESZqmITjf1qzrLFuOEYg=</ds:DigestValu
e>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    <!-- details omitted -->
                </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#_76d8adbc-0f19-4af3-80e1-
b48e25bdf02c"
                            ValueType="http://docs.oasisopen.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3"
                        />
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="_f8aa8b55-b31c-4364-94d0-3615ca65aa40">
        <wsrm:CreateSequence>
            <wsrm:AcksTo>
                <wsa:Address
                    S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"
                    >http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/icloud</wsa:Address>
                <wsa:ReferenceParameters>
                    <ebint:RoutingInput
                        S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh">
                        <ebint:UserMessage mpc="595066c0-77b6-4c0e-95db-
3015e335095e.response">
                            <eb3:PartyInfo>
                                <eb3:From>
                                    <eb3:PartyId
                            type="urn:oasis:names:tc:ebcore:partyid-
type:iso6523:0002"
                                        >123456789</eb3:PartyId>
                                    <eb3:Role>Buyer</eb3:Role>
                                </eb3:From>
                                <eb3:To>
                                    <eb3:PartyId
                            type="urn:oasis:names:tc:ebcore:partyid-
type:iso6523:0106"
                                        >192837465</eb3:PartyId>
                                    <eb3:Role>Seller</eb3:Role>
                                </eb3:To>
                            </eb3:PartyInfo>
                            <eb3:CollaborationInfo>
                                <eb3:Service>Sales</eb3:Service>
                                <eb3:Action>RequestForPayment.response</eb3:Action>
                                <eb3:ConversationId>b6ca4390-6694-419f-a2c5-
c9fa161ba0aa</eb3:ConversationId>
                            </eb3:CollaborationInfo>
                        </ebint:UserMessage>
                    </ebint:RoutingInput>
                </wsa:ReferenceParameters>
            </wsrm:AcksTo>
            <wsrmp:DeliveryAssurance>
                <wsrmp:AtLeastOnce/>
                <wsrmp:InOrder/>
            </wsrmp:DeliveryAssurance>
        </wsrm:CreateSequence>
    </S12:Body>
</S12:Envelope>

```

F.2 CreateSequenceResponse

Buyer’s MSH receives the request via the I-Cloud and produces the following response, using a `ebint:RoutingInput` reference parameter derived from the `AcksTo` structure in the `CreateSequence` message:

```
<?xml version="1.0" encoding="UTF-8"?>
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
 secext-1.0.xsd"
    xmlns:ebint="http://docs.oasis-open.org/ebxml-msg/ns/ebms/v3.0/multihop/200902/"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
 utility-1.0.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 >
    <S12:Header>
        <wsa:MessageID wsu:Id="_de488ec7-e533-486e-ab16-f48889a3675a"
            >http://buyer.example.com/9393b326-e9f3-4969-94bb-
 afc525413370.wsrm.csr</wsa:MessageID>
        <wsa:To wsu:Id="_256766f8-19f8-4ee1-9be6-e00c05a79c9c"
            S12:mustUnderstand="true"
            S12:role="http://docs.oasis-open.org/ebxml-
 msg/ebms/v3.0/ns/part2/200811/nextmsh"
            >http://docs.oasis-open.org/ebxml-
 msg/ebms/v3.0/ns/part2/200811/icloud</wsa:To>
        <wsa:Action wsu:Id="_476193ac-1584-48ac-9074-143a8fc13523"
            >http://docs.oasis-open.org/ws-
 rx/wsrm/200702/CreateSequenceResponse</wsa:Action>
        <wsa:RelatesTo wsu:Id="_60e8a742-50cd-4eb4-8a7c-e49a08f26372"
            RelationshipType="http://www.w3.org/2005/08/addressing/reply"
            >http://seller.example.com/b6ca4390-6694-419f-a2c5-
 c9fa161ba0aa.wsrm.cs</wsa:RelatesTo>
        <ebint:RoutingInput wsa:IsReferenceParameter="true"
            id="_95aa453c-e68e-4365-8c3e-09c76d889e03"
            S12:role="http://docs.oasis-open.org/ebxml-
 msg/ebms/v3.0/ns/part2/200811/nextmsh"
            S12:mustUnderstand="true">
            <ebint:UserMessage mpc="595066c0-77b6-4c0e-95db-3015e335095e.response">
                <eb3:PartyInfo>
                    <eb3:From>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0002"
                            >123456789</eb3:PartyId>
                        <eb3:Role>Buyer</eb3:Role>
                    </eb3:From>
                    <eb3:To>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0106"
                            >192837465</eb3:PartyId>
                        <eb3:Role>Seller</eb3:Role>
                    </eb3:To>
                </eb3:PartyInfo>
                <eb3:CollaborationInfo>
                    <eb3:Service>Sales</eb3:Service>
                    <eb3:Action>RequestForPayment.response</eb3:Action>
                    <eb3:ConversationId>b6ca4390-6694-419f-a2c5-
c9fa161ba0aa</eb3:ConversationId>
                </eb3:CollaborationInfo>
            </ebint:UserMessage>
        </ebint:RoutingInput>
        <wsse:Security S12:mustUnderStand="true">
            <wsu:Timestamp wsu:Id="_5cb44655-5720-4cf4-a772-19cd480b0ad4">
                <!-- details omitted -->
            </wsu:Timestamp>
            <wsse:BinarySecurityToken wsu:Id="_41666e7f-facb-4f3e-86b0-
f6ba246e5c6a">
                <!-- details omitted -->
            </wsse:BinarySecurityToken>
            <ds:Signature>
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#_de488ec7-e533-486e-ab16-f48889a3675a">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>bgUQRNjzfnYSAh9mkq0C5+G0xaY=</ds:DigestValue
>
                    </ds:Reference>
                    <ds:Reference URI="#_256766f8-19f8-4ee1-9be6-e00c05a79c9c">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>CNfEqkxRjnXM8P/hbfe4vzsuEBs=</ds:DigestValue
>
                    </ds:Reference>
                    <ds:Reference URI="#_476193ac-1584-48ac-9074-143a8fc13523">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>kNNQkuLbjuAZaqIVlovDrIUFiLw=</ds:DigestValue
>
                    </ds:Reference>
                    <ds:Reference URI="#_60e8a742-50cd-4eb4-8a7c-e49a08f26372">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>6FWE1+iUD9UnJTbnqn/BPT9fnZw=</ds:DigestValue
>
                    </ds:Reference>
                    <ds:Reference URI="#_95aa453c-e68e-4365-8c3e-09c76d889e03">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>+AktU8cQ9k152maEKVQRZJfmLiQ=</ds:DigestValue
>
                    </ds:Reference>
                    <ds:Reference URI="#_7863066f-44bb-49d0-a7f2-8edc045b4601">
                        <ds:Transforms>
```

```
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
 exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>WLKJJXX66QrUwSv0TVA+dP/EMt0=</ds:DigestValue
 >
                    </ds:Reference>
                    <ds:Reference URI="_5cb44655-5720-4cf4-a772-19cd480b0ad4">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
 exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>PreCqm0ESZqmITjf1qzrLFuOEYg=</ds:DigestValue
 >
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    <!-- details omitted -->
                </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#_41666e7f-facb-4f3e-86b0-f6ba246e5c6a"
                            ValueType="http://docs.oasisopen.org/wss/2004/01/oasis-
 200401-wss-x509-token-profile-1.0#X509v3"
                        />
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="_7863066f-44bb-49d0-a7f2-8edc045b4601">
        <wsrm:CreateSequenceResponse>
            <wsrm:Identifier>64331cf8-8a64-4e36-bf8e-9cbee3cf9384</wsrm:Identifier>
        </wsrm:CreateSequenceResponse>
    </S12:Body>
</S12:Envelope>
```

F.3 UserMessage

Seller can now use the sequence to send messages. The third message sent is also asking for an acknowledgment.

```
<?xml version="1.0" encoding="UTF-8"?>
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
    xmlns:ebint="http://docs.oasis-open.org/ebxml-
msg/ns/ebms/v3.0/multihop/200902/"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 >
    <S12:Header>
        <eb3:Messaging
            S12:mustUnderstand="true"
            id="_476193ac-1584-48ac-9074-143a8fc13523">
            <eb3:UserMessage mpc="595066c0-77b6-4c0e-95db-3015e335095e">
                <eb3:MessageInfo>
                    <eb3:Timestamp>2009-06-01T13:21:55.0001Z</eb3:Timestamp>
                    <eb3:MessageId>invoice123@seller.example.com</eb3:MessageId>
                </eb3:MessageInfo>
                <eb3:PartyInfo>
                    <eb3:From>
```

```
                        <eb3:PartyId
                            type="urn:oasis:names:tc:ebcore:partyid-
type:iso6523:0106"
                            >192837465</eb3:PartyId>
                        <eb3:Role>Seller</eb3:Role>
                    </eb3:From>
                    <eb3:To>
                        <eb3:PartyId
                            type="urn:oasis:names:tc:ebcore:partyid-
type:iso6523:0002"
                            >123456789</eb3:PartyId>
                        <eb3:Role>Buyer</eb3:Role>
                    </eb3:To>
                </eb3:PartyInfo>
                <eb3:CollaborationInfo>
                    <eb3:Service>Sales</eb3:Service>
                    <eb3:Action>RequestForPayment</eb3:Action>
                    <eb3:ConversationId>595066c0-77b6-4c0e-95db-
3015e335095e</eb3:ConversationId>
                </eb3:CollaborationInfo>
                <eb3:PayloadInfo>
                    <eb3:PartInfo
                        href="cid:e3cb0fcd-6c0a-4609-875f-
5bacfb0d0764@seller.example.com"
                    />
                </eb3:PayloadInfo>
            </eb3:UserMessage>
        </eb3:Messaging>
        <wsrm:Sequence wsu:Id="_50111bf6-cec9-4f65-9b14-d8a67473bfec"
            S12:mustUnderstand='true' >
            <wsrm:Identifier>64331cf8-8a64-4e36-bf8e-9cbee3cf9384</wsrm:Identifier>
            <wsrm:MessageNumber>3</wsrm:MessageNumber>
        </wsrm:Sequence>
        <wsrm:AckRequested wsu:Id="_ce8b7adc-8299-4709-8786-7b6e47d11bfe">
            <wsrm:Identifier>64331cf8-8a64-4e36-bf8e-9cbee3cf9384</wsrm:Identifier>
        </wsrm:AckRequested>
        <wsse:Security S12:mustUnderStand="true">
            <wsu:Timestamp wsu:Id='_3f3a6960-0da8-4cfe-a558-0429da9f10f1'>
                <!-- details omitted -->
            </wsu:Timestamp>
            <wsse:BinarySecurityToken wsu:Id="_c116daa8-2c55-4021-8090-
9aba31306924">
                <!-- details omitted -->
            </wsse:BinarySecurityToken>
            <ds:Signature>
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
                        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
                        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#_476193ac-1584-48ac-9074-143a8fc13523">
                        <ds:Transforms>
                            <ds:Transform
                                Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
                            Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>G81aGuOd+yex89mdYP4yfy7zeE0=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_50111bf6-cec9-4f65-9b14-d8a67473bfec">
                        <ds:Transforms>
                            <ds:Transform
                                Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
                            Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>m/dQpoocYc5mDy7I5Q7BGtmCM6g=</ds:DigestValu
e>
```

```
                    </ds:Reference>
                    <ds:Reference URI="#_ce8b7adc-8299-4709-8786-7b6e47d11bfe">
                        <ds:Transforms>
                            <ds:Transform
                                Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
                            Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>BZv/Y2V6CBO5bML96vBJDB79kBk=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_60e8a742-50cd-4eb4-8a7c-e49a08f26372">
                        <ds:Transforms>
                            <ds:Transform
                                Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
                            Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>LgUqGoN49jOMC8v8JAFIwFIexhg=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference
                        URI="cid:e3cb0fcd-6c0a-4609-875f-
5bacfb0d0764@seller.example.com">
                        <ds:Transforms>
                            <ds:Transform
                                Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
                            Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>Mc1tKNVmA5m/j426qwk07Nr5Ba4=</ds:DigestValu
e>
                    </ds:Reference>
                    <ds:Reference URI="#_3f3a6960-0da8-4cfe-a558-0429da9f10f1">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>PreCqm0ESZqmITjf1qzrLFuOEYg=</ds:DigestValu
e>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    <!-- details omitted -->
                </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#_c116daa8-2c55-4021-8090-
9aba31306924"
                            ValueType="http://docs.oasisopen.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3"
                        />
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="_60e8a742-50cd-4eb4-8a7c-e49a08f26372"/>
</S12:Envelope>
```

F.4 Acknowledgment

The following acknowledgment is returned. It contains a `ebint:RoutingInput` that was specified as the `wsrm:AcksTo` in the message that established the sequence.

```
<?xml version="1.0" encoding="UTF-8"?>
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
    xmlns:ebint="http://docs.oasis-open.org/ebxml-
msg/ns/ebms/v3.0/multihop/200902/"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <S12:Header>
        <wsa:To wsu:Id="_95aa453c-e68e-4365-8c3e-09c76d889e03"
            S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"
            S12:mustUnderstand="true"
            >http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/icloud</wsa:To>
        <wsa:Action wsu:Id="_7863066f-44bb-49d0-a7f2-8edc045b4601"
            >http://docs.oasis-open.org/ws-
rx/wsrm/200702/SequenceAcknowledgement</wsa:Action>
        <ebint:RoutingInput wsa:IsReferenceParameter="true"
            S12:role="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/nextmsh"
            id="_a83cb2ba-956c-4b46-b6d3-cd8edd98b32e" S12:mustUnderstand="true">
            <ebint:UserMessage mpc="595066c0-77b6-4c0e-95db-3015e335095e.response">
                <eb3:PartyInfo>
                    <eb3:From>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0002"
                            >123456789</eb3:PartyId>
                        <eb3:Role>Buyer</eb3:Role>
                    </eb3:From>
                    <eb3:To>
                        <eb3:PartyId
                         type="urn:oasis:names:tc:ebcore:partyid-type:iso6523:0106"
                            >192837465</eb3:PartyId>
                        <eb3:Role>Seller</eb3:Role>
                    </eb3:To>
                </eb3:PartyInfo>
                <eb3:CollaborationInfo>
                    <eb3:Service>Sales</eb3:Service>
                    <eb3:Action>RequestForPayment.response</eb3:Action>
                    <eb3:ConversationId>b6ca4390-6694-419f-a2c5-
c9fa161ba0aa</eb3:ConversationId>
                </eb3:CollaborationInfo>
            </ebint:UserMessage>
        </ebint:RoutingInput>
        <wsrm:SequenceAcknowledgement wsu:Id="_02b17eed-be4f-4483-af26-
55a3e2412bb3"
            S12:ackRequested="true" S12:mustUnderstand="true">
            <wsrm:Identifier>64331cf8-8a64-4e36-bf8e-9cbee3cf9384</wsrm:Identifier>
            <wsrm:AcknowledgementRange Upper="1" Lower="1"/>
            <wsrm:AcknowledgementRange Upper="3" Lower="3"/>
        </wsrm:SequenceAcknowledgement>
        <wsse:Security S12:mustUnderStand="true">
            <!-- Omitted -->
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="_98825416-7b3d-4aba-85c5-089fd22f07d4"/>
</S12:Envelope>
```

Appendix G Sample Message Exchange: Store-and-Collect Intermediary

The functionality described in scenario Appendix E can be modified to support a (third-party hosted) “mailbox” type solution by changing the routing function of the intermediary. If Buyer is not addressable, it can only retrieve the asynchronous `Receipt` if the intermediary stores the response signal message and makes it available for pulling. One key difference with scenario Appendix E is that Buyer needs to actively poll the intermediary for messages on the relevant MPC. The content of the user message and the receipt are the same as in the previous scenario. The signal can be pulled from the Intermediary using an `eb3:PullRequest` signal message. No `S12:role="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/nextmsh"` attribute has to be added as the pull request accesses the intermediary directly and no forwarding of the pull request via intermediaries occurs. The intermediary also acts in the “ebms” role.

```
<?xml version="1.0" encoding="UTF-8"?>
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:ebint="http://docs.oasis-open.org/ebxml-
msg/ns/ebms/v3.0/multihop/200902/"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <S12:Header>
        <eb3:Messaging S12:mustUnderstand="true"
            id="_b2943cc2-0af3-49d8-ba09-1e3922219711">
            <eb3:SignalMessage>
                <eb3:MessageInfo>
                    <eb3:Timestamp>2009-05-21T11:30:11.320Z</eb3:Timestamp>
                    <eb3:MessageId>30c6eb92-6329-44c7-a4a3-
468d503c01f8@buyer.example.com</eb3:MessageId>
                </eb3:MessageInfo>
                <eb3:PullRequest mpc="e5c31ef7-d750-4db8-b4dc-
13a751d80b9a.receipt"/>
            </eb3:SignalMessage>
        </eb3:Messaging>
        <wsse:Security S12:role="ebms"
            S12:mustUnderstand="true"
            wsu:Id="_c885e5b3-934d-4f06-ab9c-d8b044d800d6">
            <wsse:UsernameToken>
                <!-- This must be valid at the Intermediary -->
                <wsse:Username>buyer</wsse:Username>
                <wsse:Password
                    Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordDigest"
                    >r0jfXM1XaIK1/IlwycaHZTZvDpc=</wsse:Password>
                <wsse:Nonce>h0vN6yVOXZwsaLoeLVitUdYoKlM=</wsse:Nonce>
                <wsu:Created>2009-05-21T11:30:11.320Z</wsu:Created>
            </wsse:UsernameToken>
        </wsse:Security>
        <wsse:Security
            S12:mustUnderStand="true">
            <!-- Omitted -->
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="_f9e2b212-c001-4648-a6bd-ef24d1dc813e"/>
</S12:Envelope>
```

Appendix H New ebMS Error Types

This appendix summarizes the new ebMS error types defined in this specification. Section 2.5.6 discusses the following errors:

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS:0020	RoutingFailure	failure	Processing	An Intermediary MSH was unable to route an ebMS message and stopped processing the message.
EBMS:0021	MPCCapacityExceeded	failure	Processing	An entry in the routing function is matched that assigns the message to an MPC for pulling, but the intermediary MSH is unable to store the message with this MPC
EBMS:0022	MessagePersistenceTimeout	failure	Processing	An intermediary MSH has assigned the message to an MPC for pulling and has successfully stored it. However the intermediary set a limit on the time it was prepared to wait for the message to be pulled, and that limit has been reached.
EBMS:0023	MessageExpired	warning	Processing	An MSH has determined that the message is expired and will not attempt to forward or deliver it.

Section 5.5 and 5.10.3 introduce the following two errors:

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS:0030	BundlingError	failure	Content	The structure of a received bundle is not in accordance with the bundling rules.
EBMS:0031	RelatedMessageFailed	failure	Processing	A message unit in a bundle was not processed because a related message unit in the bundle caused an error.

Section 6.6 defines the errors EBMS:0040 to EBMS:0055.

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS:0040	BadFragmentGroup	failure		A fragment is received that relates to a group that was previously rejected.
EBMS:0041	DuplicateMessageSize	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS:0042	DuplicateFragmentCount	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS:0043	DuplicateMessageHeader	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBSMS:0044	DuplicateAction	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS:0045	DuplicateCompressionInfo	failure		A fragment is received but more than one fragment message in a group of fragments specifies a value for a compression element.
EBMS:0046	DuplicateFragment	failure		A fragment is received but a previously received fragment message had the same values for GroupId and FragmentNum
EBMS:0047	BadFragmentStructure	failure		The href attribute does not reference a valid MIME data part, MIME parts other than the fragment header and a data part are in the message. are added or the SOAP Body is not empty.
EBMS:0048	BadFragmentNum	failure		An incoming message fragment has a a value greater than the known FragmentCount.
EBMS:0049	BadFragmentCount	failure		A value is set for FragmentCount, but a previously received fragment had a greature value.
EBMS:0050	FragmentSizeExceeded	warning		The size of the data part in a fragment message is greater than **Pmode[].Splitting.FragmentSize**
EBMS:0051	ReceiveIntervalExceeded	failure		More time than **Pmode[].Splitting.JoinInterval** has passed since the first fragment was received but not all other fragments are received.
EBMS:0052	BadProperties	warning		Message properties were present in the fragment SOAP

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
				header that were not specified in **Pmode[].Splitting.RoutingProperties**
EBMS:0053	HeaderMismatch	failure		The eb3:Message header copied to the fragment header does not match the eb3:Message header in the reassembled source message.
EBMS:0054	OutOfStorageSpace	failure		Not enough disk space available to store all (expected) fragments of the group.
EBMS:0055	DecompressionError	failure	processing	An error occurred while decompressing the reassembled message.

Section 7.2 defines the following error

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS:0060	ResponseUsing-AlternateMEP	Warning	Processing	A responding MSH indicates that it applies the alternate MEP binding to the response message.

Appendix I MessageFragment Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:mf="http://docs.oasis-open.org/ebxml-msg/ns/v3.0/mf/2010/04/"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    targetNamespace="http://docs.oasis-open.org/ebxml-
msg/ns/v3.0/mf/2010/04/"
    >

    <xsd:import namespace="http://www.w3.org/2003/05/soap-envelope"
        schemaLocation="http://www.w3.org/2003/05/soap-envelope"/>

    <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
        schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>

    <xsd:element name="MessageFragment" type="mf:MessageFragmentType"/>

    <xsd:complexType name="MessageFragmentType">
        <xsd:sequence>
            <xsd:element name="GroupId" type="mf:non-empty-string">
                <xsd:annotation>
                    <xsd:documentation>The identifier of the group of
fragments that a message is
                        split into. </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="MessageSize" type="xsd:positiveInteger"
minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>The size of the message as the sum
of the size of all
                        fragments (excluding headers), i.e. before
splitting or after reassembly.
                        This needs to be specified exactly once for a
group. </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="FragmentCount" type="xsd:positiveInteger"
minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>The number of items that are part of
the group. This needs to
                        be specified exactly once for a group.
</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="FragmentNum" type="xsd:positiveInteger">
                <xsd:annotation>
                    <xsd:documentation>The identifier of an individual
message in a sequence. 1 &lt;
                        FragmentCount &lt; </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="MessageHeader" type="mf:MessageHeaderType"
minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>The header fields and parameters of
the fragmented message.
                        This needs to be specified exactly once for a
group. </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="Action" type="xsd:string" minOccurs="0">
```

```
                <xsd:annotation>
                    <xsd:documentation> This needs to be specified exactly 
once for a group. When
                        using SOAP 1.1, this value corresponds to the 
SOAPAction header. When using
                        SOAP 1.2, this value corresponds to the action 
parameter. When not set, this
                        value of the SOAPaction header or action parameter 
is assumed to be the
                        empty string. </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:sequence minOccurs="0" maxOccurs="1">
                <xsd:element name="CompressionAlgorithm" 
type="mf:CompressionAlgorithmType"/>
                <xsd:element name="CompressedMessageSize" 
type="xsd:positiveInteger"/>
            </xsd:sequence>
            <xsd:any namespace="##other" processContents="lax" 
minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="href" type="xsd:anyURI"/>
        <xsd:attributeGroup ref="mf:S12atts"/>
        <xsd:attributeGroup ref="mf:S11atts"/>
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:complexType>

    <xsd:complexType name="MessageHeaderType">
        <xsd:sequence>
            <xsd:element name="Content-Type" type="mf:non-empty-string" 
fixed="Multipart/Related"/>
            <xsd:element name="Boundary" type="mf:non-empty-string">
                <xsd:annotation>
                    <xsd:documentation>The MIME boundary separating the 
MIME parts in the MIME
                        envelop. </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="Type" type="mf:TypeType"/>
            <xsd:element name="Start" type="mf:non-empty-string"/>
            <xsd:element name="StartInfo" type="mf:non-empty-string" 
minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>For MTOM XOP </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="Content-Description" type="xsd:string" 
minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="non-empty-string">
        <xsd:restriction base="xsd:string">
            <xsd:minLength value="1"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="TypeType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="application/xop+xml">
                <xsd:annotation>
                    <xsd:documentation>XOP Package as defined in
                        http://www.w3.org/TR/2005/REC-xop10-20050125/ 
</xsd:documentation>
                </xsd:annotation>
            </xsd:enumeration>
            <xsd:enumeration value="text/xml">
                <xsd:annotation>
                    <xsd:documentation>SOAP with attachments 
</xsd:documentation>
                </xsd:annotation>
            </xsd:enumeration>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="HTTPCompressionAlgorithmType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="gzip"/>
            <xsd:enumeration value="compress"/>
            <xsd:enumeration value="deflate"/>
            <xsd:enumeration value="identity"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="CompressionAlgorithmType">
        <xsd:union  memberTypes="mf:non-empty-string
mf:HTTPCompressionAlgorithmType"/>
    </xsd:simpleType>

    <xsd:attributeGroup name="S12atts">
        <xsd:attribute ref="S12:mustUnderstand" use="optional">
            <xsd:annotation>
                <xsd:documentation> if SOAP 1.2 is being used, this
attribute is required, other
                    attributes in the S12atts group are allowed and
attributes in the S11atts group
                    are prohibited.</xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute ref="S12:encodingStyle"/>
        <xsd:attribute ref="S12:relay"/>
        <xsd:attribute ref="S12:role"/>
    </xsd:attributeGroup>

    <xsd:attributeGroup name="S11atts">
        <xsd:attribute ref="S11:mustUnderstand" use="optional">
            <xsd:annotation>
                <xsd:documentation> if SOAP 1.1 is being used, this
attribute is required, other
                    attributes in the S11atts group are allowed and
attributes in the S12atts group
                    are prohibited.</xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute ref="S11:encodingStyle"/>
        <xsd:attribute ref="S11:actor"/>
    </xsd:attributeGroup>

</xsd:schema>
```

Appendix J Acknowledgments

The editors would also like to acknowledge the contributions of the OASIS ebXML Messaging Services TC, whose members at the time of publication were:

- Timothy Bennet, Drummond Group Inc.
- Weisin Chong, Cisco Systems, Inc.
- Jacques Durand, Fujitsu Limited (Secretary).
- Pim van der Eijk, Sonnenglanz Consulting.
- Richard Emery, Axway Software
- Sander Fieten, Individual member.
- Kazunori Iwasa, Fujitsu Limited.
- Ian Jones, British Telecommunications Plc (Chair)
- Dale Moberg, Axway.
- Ernst Jan van Nigtevecht, Sonnenglanz Consulting.
- Makesh Rao, Cisco Systems, Inc.
- Dilip Sarma, Sybase.
- John Voss, Cisco Systems, Inc.
- David Webber, Individual member.

- The following former TC member contributed to chapter 2 of this specification and this contribution is gratefully acknowledged:

- Rajashekar Kailar, Centers for Disease Control and Prevention.

Appendix K Revision History

Rev	Date	By Whom	What
CD 1	06/30/10	J. Durand / P. van der Eijk	CD 1 draft for PR