INTERNATIONAL STANDARD

ISO 15000-1

First edition
2021-02

Electronic business eXtensible Markup Language (ebXML) —

Part 1: Messaging service core specification

Reference number
ISO 15000-1:2021(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by the OASIS ebXML (Electronic business eXtensible Markup Language) Messaging Services Committee (as “OASIS ebXML Messaging Services Version 3.0”) and drafted in accordance with its editorial rules. It was assigned to Technical Committee ISO/TC 154, *Processes, data elements and documents in commerce, industry and administration* and adopted under the "fast-track procedure".

This first edition incorporates content from ISO/TS 15000-2:2004 and ISO/TS 15000-1:2004, which have been withdrawn.

The main changes compared to ISO/TS 15000-2:2004 and ISO/TS 15000-1:2004 are as follows:

- The original ISO/TS 15000-2:2004 specification for the ebXML Messaging Service (ebMS) has been updated and refactored into multiple parts, including this document, the “Core” specification for ebMS, resubmitted to become this document.
- A separate “AS4 Profile of ebMS 3.0 Version 1.0” is submitted separately to ISO/TC 154. It provides a select limited profile of the ebMS3 specification sufficient for Web Services business-to-business messaging applications over the HTTP transport protocol, and is to become ISO 15000-2.
- This document defines the basic (and some optional) features necessary for reliable electronic messaging and the transactional interactions that support such messaging.
- This document noted the availability of several newer methodologies, represented by normative references and informative references included here, that were not available as of the 2004 version.

A list of all parts in the ISO 15000 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document describes a communication-protocol neutral method for exchanging electronic business messages. It defines specific enveloping constructs supporting reliable, secure delivery of business information. Furthermore, the specification defines a flexible enveloping technique, permitting messages to contain payloads of any format type. This versatility ensures that legacy electronic business systems employing traditional syntaxes (i.e. UN/EDIFACT, ASC X12, or HL7) can leverage the advantages of the ebXML infrastructure along with users of emerging technologies.

The prime objective of the ebXML messaging service (ebMS) is to facilitate the exchange of electronic business messages within an XML framework that leverages common Internet standards, without making any assumption on the integration and consumption model these messages will follow on the back-end. These messages may be consumed in different ways that are out of scope of this document: they may bind to a legacy application, to a service, be queued, enter a message workflow process, be expected by an already-running business process, be batched for delayed processing, be routed over an Enterprise Service Bus (ESB) before reaching their consumer application, or be dispatched based on header data or payload data, etc.

It is becoming critical for broad adoption among all partners – large or small - of a supply-chain, to handle differences in message flow capacity, intermittent connectivity, lack of static IP addresses or firewall restrictions. Such new capabilities played an important role in the motivation that led to ebMS 3.0, along with the need to integrate and profile the emerging SOAP-based QoS-supporting standards. The message header profiling that provided, in ebMS 2.0, a standard business-level header, has also been extended to better address the diversity of back-end binding models, as well as the emerging trend in business activity monitoring, the eBusiness side of which a message handler should be able to support.

The ebXML messaging framework is not a restrictive one: business messages, identified as the 'payloads' of ebXML messages, are not limited to XML documents. Traditional EDI formats may also be transported by ebMS. These payloads can take any digital form–XML, ASC X12, HL7, AIAG E5, database tables, binary image files, etc. Multiple payloads, possibly of different MIME types, can be transported in a single ebMS message. An objective of ebXML Messaging protocol is to be capable of being carried over any available transfer protocol. This version of the specification provides bindings to HTTP and SMTP, but other protocols to which SOAP may bind can also be used. The choice of an XML framework rather reflects confidence in a growing XML-based Web infrastructure and development tools infrastructure, the components of which can be leveraged and reused by developers.

Electronic business eXtensible Markup Language (ebXML) —

Part 1: Messaging service core specification

1. Scope

This document provides a communication-protocol neutral method for exchanging electronic business messages. It defines specific enveloping constructs supporting reliable, secure delivery of business information. Furthermore, this document defines a flexible enveloping technique, permitting messages to contain payloads of any format type.

It specifies each of the following:

- Messaging model
- Message pulling and partitioning
- Processing modes
- Message packaging
- Error handling
- Security module
- Reliable messaging module

This document is applicable to all types of organizations (e.g., commercial enterprises, government agencies, not-for-profit organizations) that exchange documents or data electronically using messaging.

2. Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

INTERNET ASSIGNED NAMES AUTHORITY (IANA). *MIME Media Types*, Available from http://www.iana.org/assignments/media-types/.

INTERNET ENGINEERING TASK FORCE (IETF). RFC 2045. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, 1996. Edited by N Freed, et al. Available from http://www.ietf.org/rfc/rfc2045.txt.

INTERNET ENGINEERING TASK FORCE (IETF). RFC 2279. *UTF-8, a transformation format of ISO 10646*, 1998. Edited by F. Yergeau. Available from http://www.ietf.org/rfc/rfc2279.txt.

INTERNET ENGINEERING TASK FORCE (IETF). RFC 2387. *The MIME Multipart/Related Content-type*, 1998. Edited by E. Levinson, Available from http://www.ietf.org/rfc/rfc2387.txt.

INTERNET ENGINEERING TASK FORCE (IETF). RFC 2392. *Content-ID and Message-ID Uniform Resource Locators*, 1998. Edited by E. Levinson. Available from http://www.ietf.org/rfc/rfc2392.txt.

INTERNET ENGINEERING TASK FORCE (IETF). RFC 2396. *Content-ID and Message-ID Uniform Resource Locators*, 1998. Edited by T. Berners-Lee, et al. Available from http://www.ietf.org/rfc/rfc2396.txt.

INTERNET ENGINEERING TASK FORCE (IETF). RFC 2616. *Hypertext Transfer Protocol -- HTTP/1.1*, 1999. Edited by R. Fielding, et al. Available from http://www.ietf.org/rfc/rfc2616.txt.

INTERNET ENGINEERING TASK FORCE (IETF). RFC 2822. *Internet Message Format*, 2001. Edited by P. Resnick. Available from http://www.ietf.org/rfc/rfc2822.txt.

INTERNET ENGINEERING TASK FORCE (IETF). RFC 2821. *Simple Mail Transfer Protocol*, 2001. Edited by J. Klensin. Available from http://www.ietf.org/rfc/rfc2821.txt.

OASIS. *WS-Reliability 1.1*, 2004. Edited by Kazunori Iwasa, et al. Available from http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf.

OASIS. *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1*, 2007. Edited by D. Davis, et al. Available from http://docs.oasis-open.org/ws-rx/wsrm/v1.1/wsrm.pdf.

OASIS. *Web Services Reliable Messaging Policy (WS-RM Policy) Version 1.1*, 2007. Edited by D. Davis, et al. Available from http://docs.oasis-open.org/ws-rx/wsrmp/v1.1/wsrmp.pdf.

OASIS.*Web Services Security: SOAP Message Security 1.0*, 2004. Edited by Anthony Nadalin, et al Available from http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.

OASIS.*Web Services Security UsernameToken Profile 1.0*, 2004. Edited by P. Hallam-Baker, et al. Available from http://docs.oasis-open.org/wss/2004/01/.

OASIS.*Web Services Security X.509 Certificate Token Profile*, 2004. Edited by P. Hallam-Baker, et al Available from http://docs.oasis-open.org/wss/2004/01/.

ASIS.*Web Services Security: SOAP Message Security 1.1*, 2005. Edited by Anthony Nadalin, et al Available from http://docs.oasis-open.org/wss/v1.1/.

OASIS.*Web Services Security UsernameToken Profile 1.1*, 2006. Edited by A. Nadalin, et al. Available from http://docs.oasis-open.org/wss/v1.1/.

OASIS.*Web Services Security X.509 Certificate Token Profile 1.1*, 2006. Edited by A. Nadalin, et al. Available from http://docs.oasis-open.org/wss/v1.1/.

WEB SERVICES INTEROPERABILITY ORGANIZATION. *Attachments Profile Version 1.0*, 2004. Edited by Chris Ferris, et al. Available from http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html.

WEB SERVICES INTEROPERABILITY ORGANIZATION. *Basic Security Profile Version 1.0*, 2005. Edited by Abbie Barbir, et al. Available from http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html.

WORLD WIDE WEB CONSORTIUM. *Simple Object Access Protocol (SOAP) 1.1*, 2000. Edited by D. Box, et al. Available from http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

WORLD WIDE WEB CONSORTIUM. SOAP Version 1.2 Part 1: Messaging Framework, 2003. Edited by M. Gudgin, et al Available from http://www.w3.org/TR/soap12-part1/.

WORLD WIDE WEB CONSORTIUM. *SOAP Messages with Attachments*, 2000. Edited by J. Barton, et al. Available from http://www.w3.org/TR/SOAP-attachments.

WORLD WIDE WEB CONSORTIUM.*Extensible Markup Language (XML) 1.0 (Third Edition)*, 2004. Edited by Tim Bray, et al. Available from http://www.w3.org/TR/2004/REC-xml-20040204/.

WORLD WIDE WEB CONSORTIUM. *XML Schema Part 1: Structures Second Edition*, 2004. Edited by Henry S. Thompson, et al. Available from http://www.w3.org/TR/xmlschema-1/.

WORLD WIDE WEB CONSORTIUM. *XML-Signature Syntax and Processing*, 2002. Edited by Donald Eastlake, et al. Available from http://www.w3.org/TR/xmldsig-core/.

WORLD WIDE WEB CONSORTIUM. *XML Encryption Syntax and Processing*, 2002. Edited by D. Eastlake, et al. Available from http://www.w3.org/TR/xmlenc-core/.

WORLD WIDE WEB CONSORTIUM. *Namespaces in XML*, 1999. Edited by Tim Bray, et al. Available from http://www.w3.org/TR/REC-xml-names/.

3. Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
ebMS error generation
Operation of creating an *ebMS error* (3.14) based on some failure or warning condition

3.2
ebMS error reporting
Operation of communicating an *ebMS error* (3.14) to some other entity

3.3
ebMS message
SOAP message (3.29) that contains SOAP header(s) qualified with the ebMS namespace and that conforms to this document

3.4
ebMS message exchange pattern
ebMS MEP
Choreography of *ebMS user messages* (3.11) which are all related through the use of the referencing feature (`eb:RefToMessageId`)

3.5
ebMS MEP transport channel binding
Binding of an *ebMS message exchange pattern* (3.4) that defines how the ebMS MEP maps abstractly to the channels allowed by an underlying transport protocol

3.6
ebMS MEP transport protocol binding
Binding of an *ebMS message exchange pattern* (3.4) that defines further how an *ebMS MEP transport channel binding* (3.5) is implemented over a specific underlying transport protocol

3.7
ebMS message unit
Logical unit of data that is a subset of an *ebMS Message* (3.3)

3.8
ebMS messaging service handler
ebMS MSH
Entity that is able to generate or process *ebMS messages* (3.3) that conform to this document

3.9
ebMS user message unit
ebMS message unit (3.7) that is represented by the XML infoset `eb:Messaging/eb:UserMessage`, together with any referenced payload items

Note 1 to entry: This is the part of the ebMS message that is submitted by a producer (via the "submit" operation) and that is subject to delivery to a consumer.

3.10
ebMS signal message unit
ebMS message unit (3.7) that is represented by the XML infoset `eb:Messaging/eb:SignalMessage`

Note 1 to entry: Its role is to activate a specific function in the receiving MSH. It is not intended to be delivered to a message consumer.

3.11
ebMS user message
ebMS message (3.3) that contains an *ebMS user message unit* (3.9)

3.12
ebMS signal message
ebMS message (3.3) that contains an *ebMS signal message unit* (3.10)

3.13
ebMS pull signal message
ebMS signal message (3.12) that contains an `eb:PullRequest` element

3.14
ebMS error
Particular case of *Error* (3.15) that is generated by the ebMS module in conformity with this document

3.15
Error
Object representing some failure or warning condition that originates in one of the defined modules (ebMS module, reliability module, security module)

3.16
Escalated ebMS error
ebMS error (3.14) that originates in a module other than the ebMS Module (i.e. security module, or reliability module)

3.17
Fault
Object representing some failure or warning condition that originates in SOAP processing

Note 1 to entry. It shall be generated and processed according to the W3C SOAP 1.1 or SOAP 1.2 specifications.

3.18
Message consumer
Entity that interacts with a receiving *ebMS MSH* (3.8) (i.e. an MSH in the receiving role) to consume data from a received *ebMS user message* (3.11)

3.19
Message partition channel
Mechanism to partition the flow of messages from a sending MSH to a receiving MSH into several flows that can be controlled separately and consumed differently

3.20
Message producer
Entity that interacts with a sending *ebMS MSH* (3.8) (i.e. an MSH in the sending role) to initiate the sending of an *ebMS user message* (3.11)

3.21
Message-in-error
Flawed message causing an *Error* (3.15) of some kind

3.22
One-Way message exchange pattern
one-way MEP
Exchange of a single *ebMS user message unit* (3.9) unrelated to other user messages

3.23
Processing mode
P-Mode
Contextual information that governs the processing of a particular *ebMS message* (3.3)

Note 1 to entry: A P-Mode is not provided on a per-message basis, but is common to a set of messages exchanged between or among parties. It may be interpreted as configuration data for a deployed MSH.

3.24
Processing mode operation set
Set of all *P-Modes* (3.23) that are supported by an *ebMS MSH* (3.8) during operation

3.25
Pull channel binding
ebMS MEP transport channel binding (3.5) in which the transfer of an *ebMS message* (3.3) is initiated by the receiver

3.26
Push channel binding
ebMS MEP transport channel binding (3.5) in which the transfer of an *ebMS message* (3.3) is initiated by the sender

3.27
Reliability error
Error (3.15) generated by the reliability module

3.28
Security error
Error (3.15) generated by the security module

3.29
SOAP message
Message that conforms to the SOAP 1.1 or SOAP 1.2 W3C specifications

3.30
Two-way message exchange pattern
two-way MEP
Exchange of two *ebMS user message units* (3.9) in opposite directions, in which the first one to occur is labeled "request" and the second one "reply".

4. Relevant messaging concepts

4.1. Web services and their role in an eBusiness messaging framework

A major design choice in ebMS 3 is the specification of the MSH and its associated processing rules using Web Services (WS) standards. The intent is to make use of other relevant Web Services specifications that fulfill certain messaging requirements, and build upon that base by adding what is necessary for a complete and coherent eBusiness messaging service. ebMS 3 brings this all together into a single, coherent framework.

In order to achieve this, message security and reliability requirements are met through the use of other Web Services standards and their implementations. The message SOAP body has been freed for business payload. The ebMS header is just a SOAP extension among others. As a result, ebMS 3 is significantly more compliant than ebMS 2 with the SOAP processing model, and apt at composing Web services standards that are defined as SOAP extensions. Compliance of ebMS 3 implementations with the latest version of WS-I profiles - once approved as final material by the organization - will be addressed in the definition of conformance profiles that are adjunct to this document (see Annex G). A compatibility mapping from ebMS 2 to ebMS 3 is provided in Annex F.

Compliance with Web services standards does not remove the rationale behind an Internet-based messaging middleware. Often, document-centric eBusiness and eGovernment exchanges need to clearly dissociate messaging functions from the way these messages are consumed on the back-end. Such consumption may take place according to various models. The use of SOAP message header elements that represent standard business metadata (user or company ID, business conversation, business service and action, etc.), is a key feature for supporting a decoupled binding with back-end business processes. At the same time, experience has demonstrated that the messaging layer needs to be more supportive of business transactions: messages are parts of basic choreographies that map to higher-level business exchanges between partners. To this end, ebMS 3 supports a notion of message exchange pattern (MEP) the properties of which (reliability, security, binding to underlying transport, error handling, and other quality of service aspects such as timing, etc.) are controlled in a contract-based manner by the message producer and consumer layers.

4.2. Caveats and assumptions

The target audience for this document is the community of software developers who will implement the ebXML messaging service.

It is assumed the user has an understanding of communications protocols, MIME, XML, SOAP, SOAP Messages with Attachments and security technologies.

All examples are informative. If inconsistencies exist between the specification and the examples, the specification supersedes the examples.

Implementers are strongly advised to read and understand the OASIS Collaboration Protocol Profile & Agreement specification (CPPA) and its implications prior to implementation.

This document presents some alternatives regarding underlying specifications (e.g. SOAP 1.1/1.2, WSS1.0/1.1, and Web Services specifications that support the reliability function). This does not imply that a conforming implementation supports all of them, nor that it is free to support any option. The definition of conformance profiles - out of scope for this document, and to be described in an adjunct OASIS document - will complement this document by asserting which option(s) shall be supported in order to claim support for a particular conformance profile. Conformance to compatible profiles is a prerequisite to interoperability. See Annex G for more details on conformance profiles.

4.3. XML notation

When describing concrete XML schemas and information items, this document uses a convention in which each XML element or attribute is identified using abbreviated XPath 1.0 notation (e.g., `/x:MyHeader/x:SomeProperty/@attribute`).

4.4. Namespace prefixes

Table 1 maps various prefixes that appear in XML examples to their intended corresponding namespaces.

Table 1: Namespace Prefixes

Prefix	*Namespace*
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
Ds	http://www.w3.org/2000/09/xmldsig#
eb, eb3	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
eb2	http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
enc	http://www.w3.org/2001/04/xmlenc#
wsr	http://docs.oasis-open.org/wsrm/2004/06/ws-reliability-1.1.xsd

Prefix	*Namespace*
wsrx	http://docs.oasis-open.org/ws-rx/wsrm/200702
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
ebbpsig	http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0

4.5. Example domains

Hostnames used in the examples are fictitious and conform to RFC 2606. The *example.org* domain is intended to refer generically to a relevant industry standards organization, while the *example.com* domain represents a participant in a message exchange (whether commercial, government, or other entity).

5. Messaging model

5.1. Model components

5.1.1. Components of the model

The ebMS messaging model assumes the following components:

- **ebMS MSH (Messaging Service Handler)**: An entity that is able to generate or process messages that conform to this document, and to act in at least one of two ebMS roles defined in 5.1.2: sending and receiving. In terms of SOAP processing, an MSH is either a SOAP processor or a chain of SOAP processors. In either case, an MSH shall be able to understand the `eb:Messaging` header (qualified with the ebMS namespace).

- **Producer (or Message producer)**: An entity that interacts with a sending MSH (i.e. an MSH in the sending role) to initiate the sending of a user message. Some examples are: an application, a queuing system, another SOAP processor (though not another MSH).

- **Consumer (or Message consumer)**: An entity that interacts with a receiving MSH (i.e. an MSH in the Receiving role) to consume data from a received user message. Some examples are: an application, a queuing system, another SOAP processor.

Figure 1 shows the entities and operations involved in a message exchange.

NOTE 1: In all figures, the arrows do not represent control flow, i.e. they do not represent a component invoking an operation on another component. They only represent data transfer under the control of an operation which may be implemented in either component.

NOTE 2: Producer and consumer are always MSH endpoints, and *Submit* and *Deliver* operations occur at the endpoints only once per message lifetime. Any actions performed by an intermediary will be defined in different terms.

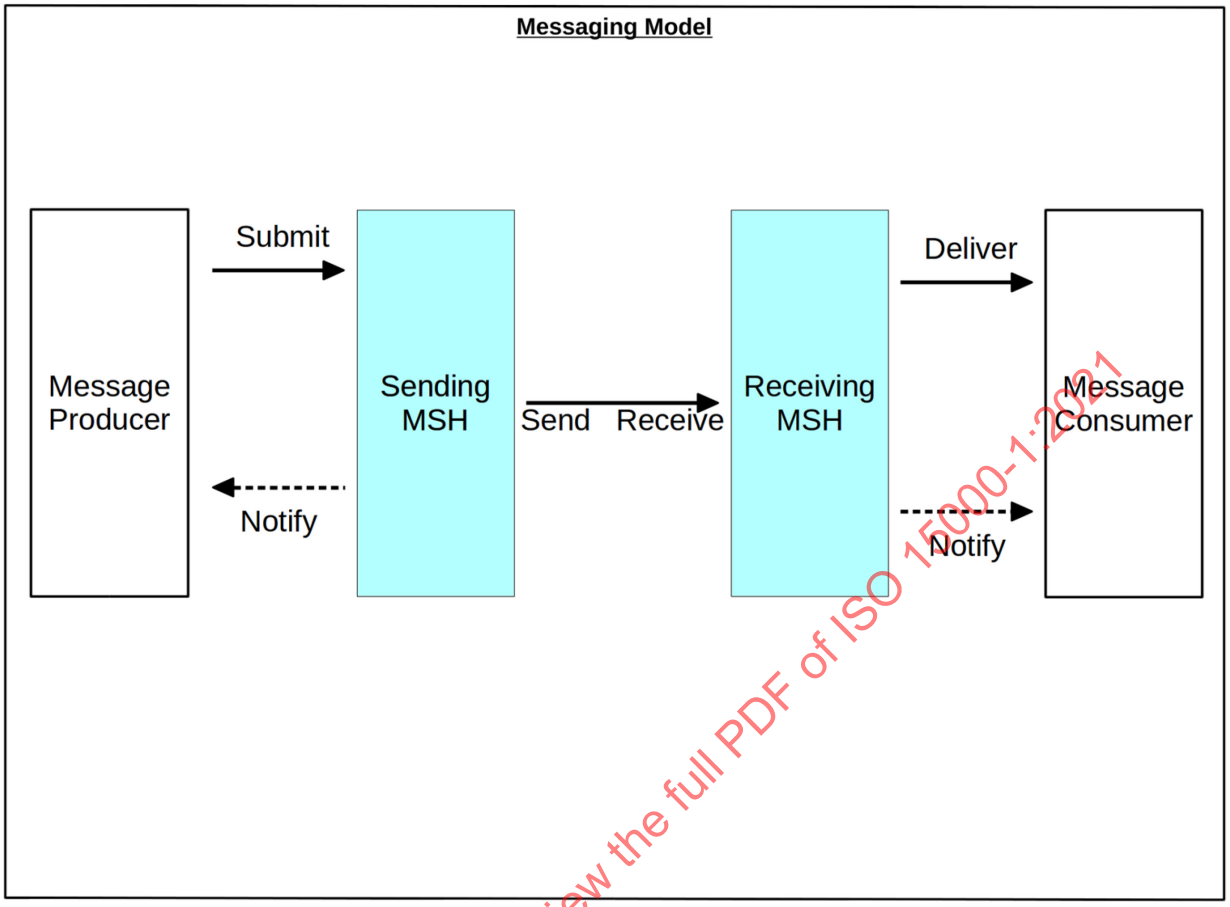

Figure 1: Entities of the Messaging Model and Their Interactions

5.1.2. Messaging roles

The Messaging Model assumes the following roles for an MSH:

- **Sending**: When an MSH acts in the *sending* role, it performs the functions associated with generating an ebMS user message and sending this message to another MSH. The abstract operations *Submit*, *Send* and *Notify* are supported by this role. (Note that even in a sending role, an MSH may be required to receive and process some types of signal messages, depending on the conformance profile in use.)
- **Receiving**: An MSH acting in the *receiving* role performs the functions associated with the receiving and processing of an ebMS user message. The abstract operations *Receive*, *Deliver* and *Notify* are supported by this role. (Note that even in a receiving role, an MSH may be required to generate and send ebMS signal messages related to the reception of messages, such as error messages or pull request signals.)

The transmission of an ebMS user message requires a pair of sending and receiving MSHs. Note that these roles are defined as only relevant to ebMS user messages, as are the abstract operations provided in 5.1.3.

5.1.3. Abstract messaging operations

An ebMS MSH supports the following abstract operations, depending on which role it is operating in:

- **Submit:** This operation transfers enough data from the producer to the sending MSH to generate an ebMS user message unit.
- **Deliver:** This operation makes data of a previously received (via *Receive* operation) ebMS user message unit available to the consumer.
- **Notify:** This operation notifies either a producer or a consumer about the status of a previously submitted or received ebMS user message unit, or about general MSH status.
- **Send**: This operation initiates the transfer of an ebMS user message from the sending MSH to the receiving MSH, after all headers intended for the receiving MSH have been added (including security and/or reliability, as required).
- **Receive**: This operation completes the transfer of an ebMS user message from the sending MSH to the receiving MSH. A successful reception means that a contained user message unit is now available for further processing by the receiving MSH.

5.2. Message exchange patterns

5.2.1. Rationale

Two communicating partners may agree to conduct business transactions as message sequences that follow well defined patterns, or message exchange patterns (MEP). Enforcing these patterns is usually done above the messaging layer. However it has proved useful to support some aspects of such MEPs in the messaging layer. In particular:

- The correlation between messages, when expressed directly via a referencing mechanism that appears in the message header, allows for efficient monitoring and enforcement of MEPs.
- As an MSH has to bind messages to the transport protocol, these binding requirements may be better expressed and controlled at MEP level. For example, different messages of the same MEP (such as a request and a response) may be required to bind differently to the transport.

An ebMS MEP represents the part of such exchange patterns that is controlled and implemented by an MSH, thus making an abstraction of the business semantics. Although the notion of MEP was not explicitly supported by ebMS 2.0, it can be noted that it provided some informal support for MEPs, such as message referencing (`eb2:RefToMessageId`) and the `eb2:SyncReply` element that controls the use of the back-channel of the underlying protocol. In the following, the acronym "MEP" implicitly means ebMS MEP, unless otherwise qualified.

The goal of this document is to introduce a model for ebMS MEPs, rather than a formal representation of them. This model is the basis for partners agreeing to which MEPs their exchanges will conform. Such agreements are manifested in processing modes, or P-Modes, the representation of which is outside the scope of this document. The P-Mode also defines which message profile is associated with which MEP, and the role it plays in this MEP. Processing modes are described in detail in Clause 7.

5.2.2. General definition

An **ebMS MEP** defines a typical choreography of ebMS user messages which are all related through the use of the referencing feature (`eb:RefToMessageId`). Each message of an MEP instance refers to a previous message of the same instance, unless it is the first one to occur. Messages are associated with a label (e.g. "request", "reply") that precisely identifies their direction between the parties involved and their role in the choreography.

NOTE: Because `eb:RefToMessageId` more accurately defines a referencing between user message units than between user messages (SOAP messages), MEPs are preferably defined here as exchanges of message units, rather than of ebMS Messages.

Two MEPs are defined in this document, not exclusive of others:

- The **One-Way MEP** which governs the exchange of a single user message unit unrelated to other user messages. Its label is "one-way" and is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay`.

- The **Two-Way MEP** which governs the exchange of two user message units in opposite directions, the first one to occur is labeled "request", the other one "reply". In an actual instance, the "reply" shall reference the "request" using `eb:RefToMessageId`. This MEP is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/twoWay`.

The MEP definitions are primarily concerned with the transfer of ebMS user message units. Instances of such MEPs may involve or cause the transfer of additional messages or the piggy-backing of additional elements (e.g. ebMS signal messages or units such as errors, receipts, pull requests, and low-level acknowledgments when using reliability), but these are not taken into account in the MEP definition. Instead, the different ways these additions can be associated with the MEPs defined here, are considered as part of the execution mode of the MEP, which is controlled by some agreement/configuration external to the MEP definition (see P-Modes in Clause 7). Some extra messages (signal messages) may also be mandated by the binding of an ebMS MEP (see channel-binding), but are not relevant to the ebMS MEP definition itself.

MEP definitions in this document are restricted to exchanges between two MSHs.

5.2.3. MEP bindings

The previous definition of ebMS MEP is quite abstract, and ignores any binding consideration to the transport protocol. This is intentional, so that application-level MEPs can be mapped to ebMS MEPs independently from the transport protocol to be used. In addition to agreeing on MEP usage, the following notions of MEP bindings should be subject to agreements between partners:

- An **ebMS MEP Transport Channel Binding** defines how the MEP maps to the channels allowed by the underlying transport protocol, while making an abstraction of this underlying transport. In case of a two-way transport, the transport channel binding defines whether each message of the MEP maps to the fore-channel (or first leg) or back-channel (second leg). It also tells if an ebMS signal is needed to initiate the transfer - e.g. by pulling - and which one. Annex E shows possible options for combining headers supporting reliable messaging as well as error reporting, when binding basic ebMS MEPs to a two-way protocol such as HTTP. Annex E also shows how these combinations can be controlled with P-Mode parameters.

- An **ebMS MEP Transport Protocol Binding** defines further how an MEP transport channel binding is implemented over a specific underlying transport protocol such as HTTP or SMTP. For example, an HTTP transport protocol binding will define the usage of HTTP headers and methods for each message. A transport protocol binding usually relies on standard SOAP bindings when these exist.

A transport channel binding is a critical complement to an MEP, to be agreed on in order for partners to interoperate. The rationale in using different transport channel bindings for an ebMS MEP is to accommodate different connectivity constraints (e.g. firewall restrictions, intermittent availability, non-static IP address) by dictating how each message transfer is initiated over the underlying protocol. Because such connectivity constraints usually exist independently from the details of the transport protocol, the transport channel binding is the right level to address them. The transport channel bindings identified in this document are:

- **Push**: maps an MEP user message to the 1st leg of an underlying two-way transport protocol, or of a one-way protocol. This binding is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push`.

- **Pull**: maps an MEP user message to the second leg of an underlying two-way transport protocol, as a result of an ebMS Pull Signal sent over the first leg. This binding is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pull`.

- **Sync**: maps an exchange of two user messages respectively to the first and second legs of a two-way underlying transport protocol. This binding is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/sync`.

NOTE 1: An underlying transport protocol qualifies as "two-way" if (a) it guarantees a transport channel for transferring the response of every message (request) initiated by an MSH, back to this MSH without need for explicit addressing information in SOAP headers, and regardless of connectivity restrictions such as inability to accept incoming new connections; and (b) it provides to the MSH initiator of the exchange, some means for correlating the response with the request, without relying on the SOAP header. For example, HTTP qualifies as two-way, but SMTP and FTP do not (although FTP has a notion of session, it does not inherently support the coupling of (b)). The channel offered in (a) is also called "back-channel" in this document.

NOTE 2: "Pull" and "sync" cannot be used with a one-way underlying protocol.

NOTE 3: Communicating parties should agree on a transport channel binding: a sending MSH will treat a message submitted for pulling differently from a message submitted for pushing.

An MEP that is associated with a particular transport channel binding is also called a transport-channel-bound MEP. A transport-channel-bound MEP is identified by a pair <MEP name / transport-channel-binding name>. For example, a two-way ebMS MEP that executes over a single request-response exchange of the underlying transport (e.g. HTTP), is called a **Two-Way/Sync** MEP.

A channel-bound MEP has an **initiating MSH**, or **initiator**, which is the one that triggers the execution of the MEP. The other MSH is called the **responding MSH**, or **responder**. These MSH roles do not change for the duration of the MEP, regardless of the number of messages exchanged and of their direction. Due to endpoint addressing or availability restrictions, some MSHs may be required to act only as initiator, and never as responder.

On the wire, the only method by which messages from the same MEP instance are associated is through a referencing link (`eb:RefToMessageId`). This referencing is decided above the MSH layer (by the producer entity). A receiving MSH relies on both this referencing and the interpretation of the P-Mode for associating a message with a specific MEP and for validating this association.

5.2.4. Relationship to SOAP MEPs

In theory, the transport-channel-bindings previously defined could be expressed in terms of SOAP MEPs instead of channels of the underlying transport protocol. However, the notion of SOAP MEP has only been introduced with SOAP 1.2, and would need to be extended to SOAP 1.1.

Also, only the SOAP Request-Response MEP and Response MEP have been formally defined, as of the time this document was written. A SOAP One-way MEP could also be defined, but how such an MEP may or may not bind to a two-way underlying protocol is yet to be determined.

Expressing the transport-channel-binding in terms of SOAP MEPs is only helpful if there is a published, non-ambiguous, standard way for these to map to the underlying protocol(s). This is currently only the case for some SOAP MEPs and some transport protocols. Consequently, this document has chosen to express its transport-channel-bindings directly in terms of how to use the channels of the transport protocol, abstracting such a transport as either "one-way" or "two-way".

5.2.5. The One-Way/Push MEP

This transport-channel-bound MEP involves the transfer of a single ebMS user message unit (label: "one-way").

To conform to this MEP, the ebMS user message unit that is exchanged shall not relate to any other user message unit (no `eb:RefToMessageId` element). Figure 2 illustrates the exchange pattern and MSH operations involved in this MEP.

In case the One-Way/Push MEP is performed over a Two-way underlying transport protocol, the response message may carry an ebMS signal message, such as an error message, or other SOAP headers. Such an option is controlled by the P-Mode (see Clause 7). However, the response message shall not carry an ebMS user message that refers to the request message. If the P-Mode allows faults to be reported on the two-way protocol's back-channel, the MEP can be qualified as a **robust** MEP, but is still an ebMS One-Way/Push MEP.

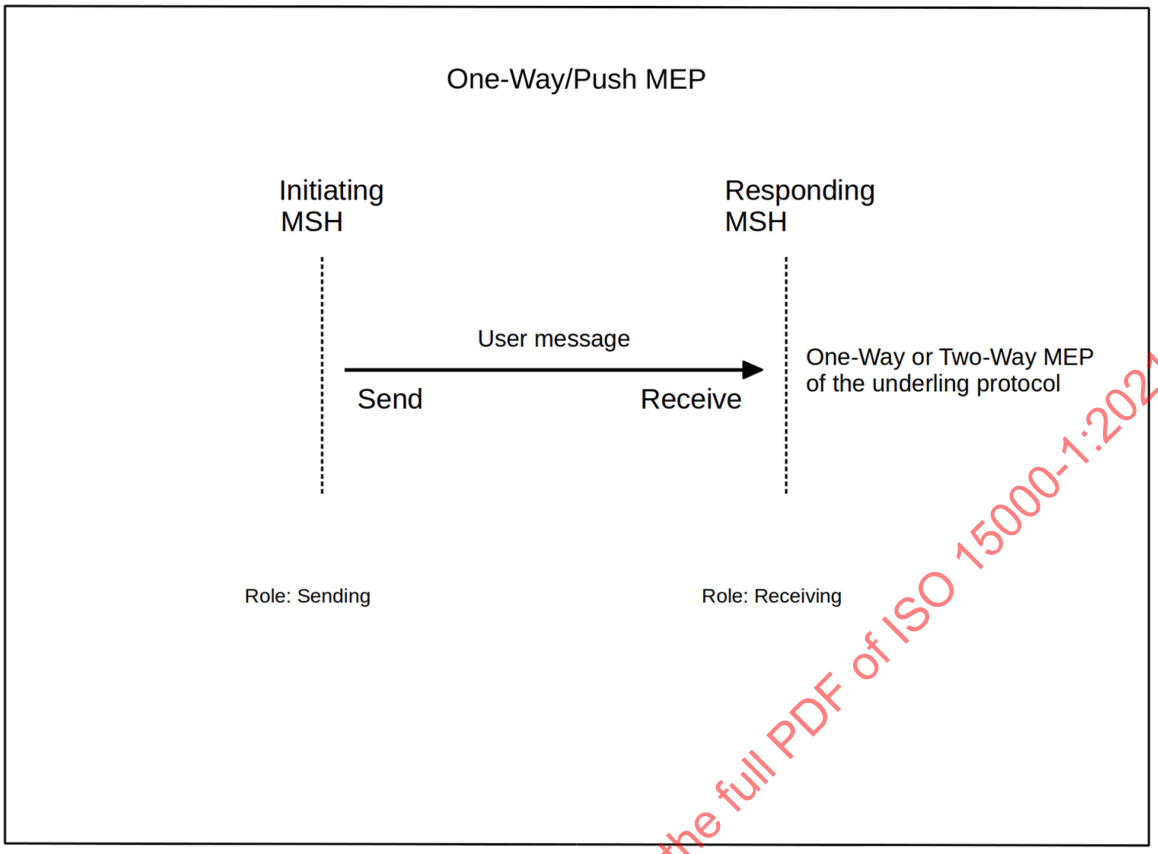

Figure 2: One-Way/Push MEP

5.2.6. The One-Way/Pull MEP

This transport-channel-bound MEP involves the transfer of a single ebMS user message unit (label: "one-way"). This MEP is initiated by the receiving MSH, over a two-way underlying transport protocol. The first leg of the protocol exchange carries a pull signal message. The second leg returns the pulled user message unit. To conform to this MEP the pulled user message unit shall not include an `eb:RefToMessageId` element. In case no message is available for pulling, an ebMS error signal of severity level "warning" and short description of "EmptyMessagePartitionChannel", as listed in 9.7.2, shall be returned over the response leg. Figure 3 illustrates this MEP.

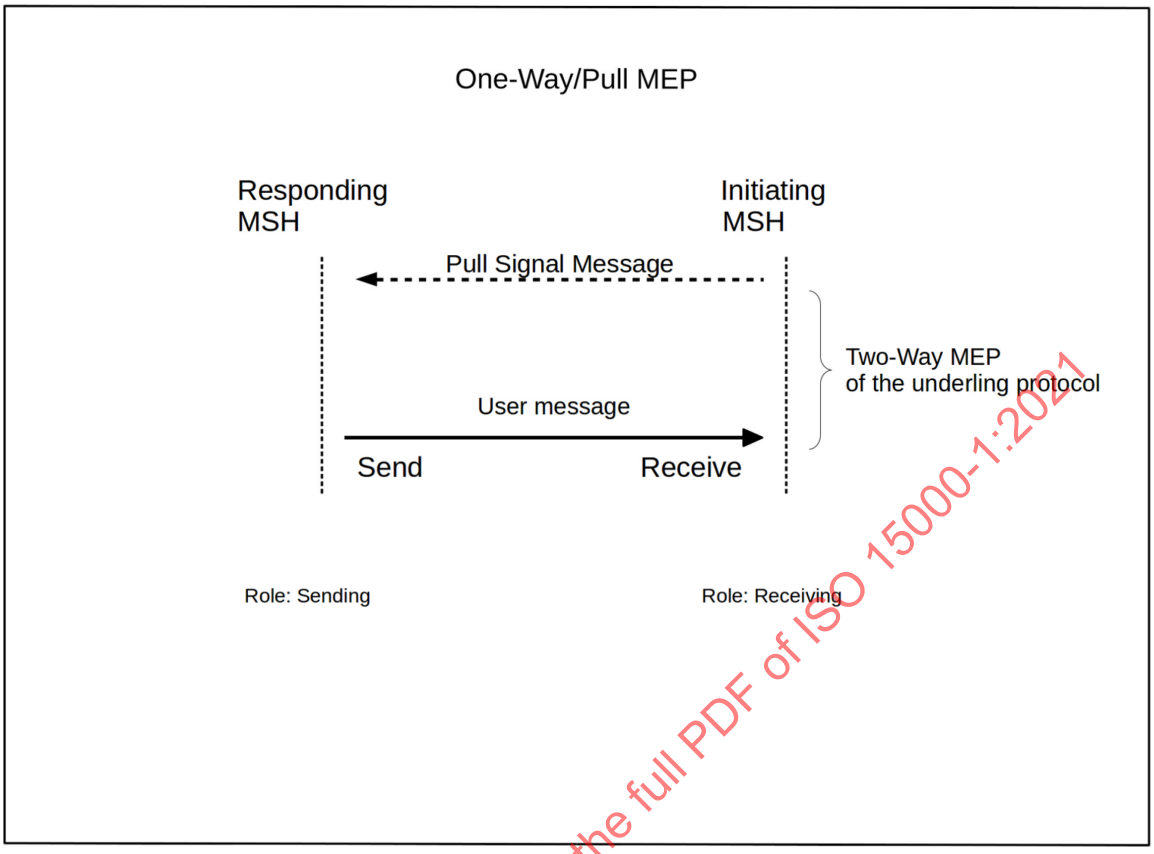

Figure 3: One-Way/Pull MEP

5.2.7. The Two-Way/Sync MEP

This transport-channel-bound MEP transfers two ebMS user message units over a single request-response exchange of the underlying two-way transport protocol. The initiator MSH sends the first user message called the "request". In the second leg of the MEP, a related user message unit called the "reply" is sent back. To conform to this MEP, the request shall not relate to any other user message unit (no `eb:RefToMessageId` element), and the reply shall refer to the request via the `eb:RefToMessageId` header element, as described in 8.2.3.2. Figure 4 illustrates this MEP.

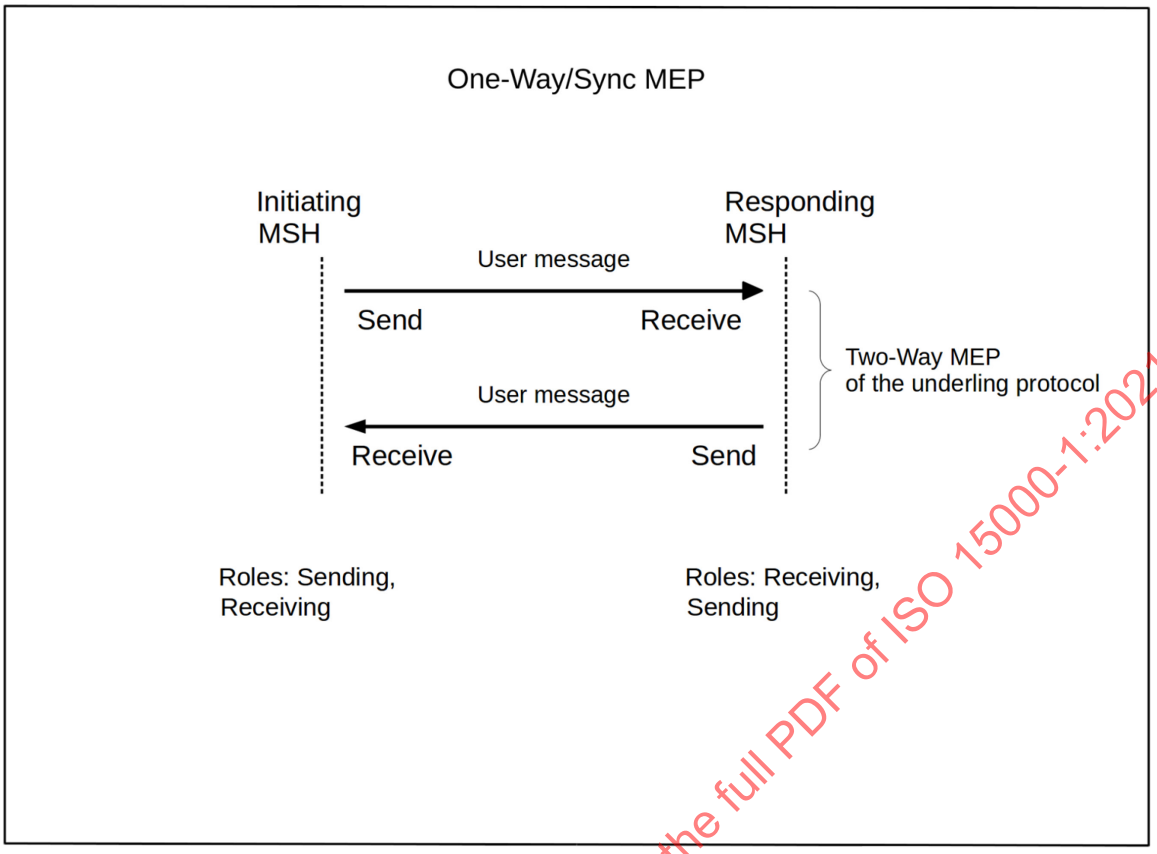

Figure 4: Two-Way/Sync MEP

5.2.8. Other transport-channel-bound MEPs

So far, message exchanges conforming to either one of the three previous transport-channel-bound MEPs were only using one instance of an underlying transport protocol MEP for their binding. The following channel-bound ebMS MEPs are expected to be among the most common ebMS MEPs that use more than one underlying transport protocol MEP instance between the Initiating MSH and the Responding MSH. In this sense, they may be qualified as asynchronous.

- The **Two-Way/Push-and-Push MEP** composes the choreographies of two One-Way/Push MEPs in opposite directions, the user message unit of the second referring to the user message unit of the first via `eb:RefToMessageId`. This MEP is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pushAndPush`.

- The **Two-Way/Push-and-Pull MEP** composes the choreography of a One-Way/Push MEP followed by the choreography of a One-Way/Pull MEP, both initiated from the same MSH (Initiator). The user message unit in the "pulled" message shall refer to the previously "pushed" user message unit. This MEP is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pushAndPull`.

- The **Two-Way/Pull-and-Push MEP** composes the choreography of a One-Way/Pull MEP followed by the choreography of a One-Way/Push MEP, with both MEPs initiated from the same MSH. The user message unit in the "pushed" message shall refer to the previously "pulled" user message unit. This MEP is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pullAndPush`.

In all these MEPs, the messages labels are respectively "request" and "reply". The two last MEPs support asynchronous exchanges where one party is constrained in terms of addressing or connectivity capability.

6. Message pulling and partitioning

6.1. Objectives

Business partners may experience differences in their ability to handle message flow, intermittent connectivity, lack of static IP addresses or firewall restrictions. In addition, when a message is transferred and successfully acknowledged, the responsibility for its management shifts sides. For these reasons, a receiver may want (a) to retain control over the transfer procedure of the underlying protocol by initiating transfers, and/or (b) to decide which messages it wants to receive first and when. Two features have been introduced in ebMS 3 that support this:

- Message pulling
- Message partition channels (MPCs)

Message pulling is defined in an abstract way by the One-Way/Pull ebMS MEP (see 5.2.6). This MEP allows an MSH to initiate the transfer of a message as a receiver. When used in combination with the One-Way/Push ebMS MEP, it allows an MSH full control over initiating asynchronous transfers with another MSH in both directions, engaging in a client-server type of interaction with the remote MSH, without any need to open a TCP/IP port to incoming requests. This MEP also supports exchanges with a partner that is intermittently connected: instead of periodically polling for partner presence, a sending MSH will simply wait for the partner MSH to pull its messages.

***Example**: A mobile, occasionally connected device without static IP address and with limited storage capability can only initiate requests and receive messages as synchronous responses to these requests. The One-Way/Pull MEP allows this device to enable and control the flow of received messages, and to adjust it to its own resources.*

Message partition channels (see 6.4) allow for partitioning the flow of messages from an MSH to another MSH into separate flows, so that each one of these flows can be controlled independently by either MSH, in terms of transfer priorities. A sending MSH shall be able to determine whether a submitted message should be pulled or pushed, and to which message partition channel (MPC) it shall be assigned. Similarly, the receiving MSH is aware of which MPC(s) should be pulled from, and which ones will be used for push. This knowledge is based on an agreement shared between parties prior to the exchanges, and modeled in this document as the P-Mode operation set (see clause 7).

6.2. Supporting message pulling

Using message pulling requires the ability of an MSH to support the One-Way/Pull MEP. The pull request signal that initiates this MEP is described in 8.2.4.2. Because there is always at least one MPC open between a sending MSH and a receiving MSH –the default MPC– the Pull mode can be supported regardless of the ability to support several MPCs.

When sending an `eb:PullRequest` signal, the name of the MPC to pull messages from shall be specified (in `eb:PullRequest/@mpc` attribute), unless the default value is to be assumed.

The processing model for a pulled message is as follows, for a typical and successful instance of One-Way/Pull MEP:

On responding MSH side:

1. Submit: submission of message data to the MSH by the producer party, intended for the consumer on the Initiator side. The message is associated with an MPC. If no MPC name is provided by the submitter, or if the MSH implementation has not been provided with a way to determine this association by itself, the default MPC is used. The MEP associated with this message (e.g. as specified by P-Mode.MEP; see 7.3) is a One-Way/Pull.

On initiating MSH side:

2. Sending of a pull request signal by the MSH. The pull request signal specifies the MPC from which to pull messages.

On responding MSH side:

3. Reception of the pull request signal. For every pull request signal received the Responder MSH (acting in sending role) selects a previously submitted message. Messages should be selected according to a FIFO policy with respect to the *Submit* operation. If there is no user message available in the specified MPC for sending, a warning signal with short description: "EmptyMessagePartitionChannel" (see 9.7.2) shall be sent back instead.

4. Send: the selected message is sent over the SOAP response to the pull request.

On initiating MSH side:

5. Receive: the pulled message is available for processing by the MSH. The header `@mpc` attribute indicates from which MPC it has been pulled, and is the same as the value of `@mpc` in the corresponding pull request signal.

6. Deliver: after processing of ebMS headers, delivery of the pulled message data to the consumer of the MSH.

Example 1: An example of `eb:Messaging` header for the pull request signal:

```
<S11:Envelope>
<S11:Header>
<eb:Messaging S11:mustUnderstand="1">
       <eb:SignalMessage>
              <eb:MessageInfo>
                     <eb:Timestamp>2006-10-01T10:01:00</eb:Timestamp>
                     <eb:MessageId>UUID-4@receiver.example.com</eb:MessageId>
              </eb:MessageInfo>
              <eb:PullRequest mpc="http://sender.example.com/mpc123"/>
       </eb:SignalMessage>
</eb:Messaging>
</S11:Header>
<S11:Body/>
</S11:Envelope>
```

Example 2: An outline of `eb:Messaging` header for the response to the pull request signal example from Example 1:

```
<S11:Envelope>
<S11:Header>
<eb:Messaging S11:mustUnderstand="1" >
   <eb:UserMessage mpc="http://sender.example.com/mpc123">
      <eb:MessageInfo>
         <eb:Timestamp>2006-10-01T10:02:00</eb:Timestamp>
         <eb:MessageId>UUID-5@sender.example.com</eb:MessageId>
```

```
         <eb:RefToMessageId>UUID-4@receiver.example.com</eb:RefToMessageId>
      </eb:MessageInfo>
      <eb:PartyInfo>
         ...
      </eb:PartyInfo>
      <eb:CollaborationInfo>
         ...
      </eb:CollaborationInfo>
      <eb:PayloadInfo>
         ...
      </eb:PayloadInfo>
   </eb:UserMessage>
</eb:Messaging>
</S11:Header>
<S11:Body>
...
</S11:Body>
</S11:Envelope>
```

6.3. Combining pulling with security and reliability

Reliability of a pulled message is usually associated with the reliability of the corresponding pull request signal. The reliability of the One-Way/Pull MEP instance is addressed in 11.3.

Security for the pull request signal is described in details in 10.12.

Example 1: An outline of a secure and reliable `eb:Messaging` header for the pull request signal follows. The reliability header used in the example assumes the use of WS-Reliability, and specifies At-Least-Once delivery, with an acknowledgment to be returned on the MEP response message:

```
<S11:Envelope>
<S11:Header>
<eb:Messaging S11:mustUnderstand="1" >
      <eb:SignalMessage>
            <eb:MessageInfo>
                  <eb:Timestamp>2006-10-01T10:01:00</eb:Timestamp>
                  <eb:MessageId>UUID-4@receiver.example.com</eb:MessageId>
            </eb:MessageInfo>
            <eb:PullRequest mpc="http://sender.example.com/mpc123"/>
      </eb:SignalMessage>
</eb:Messaging>
<wss:Security>
      ...
</wss:Security>
<wsr:Request S11:mustUnderstand="1">
      ...
      <ReplyPattern>
            <Value>Response</Value>
      </ReplyPattern>
      <AckRequested/>
      ...
</wsr:Request>
</S11:Header>
<S11:Body/>
</S11:Envelope>
```

Example 2: An outline of secure and reliable `eb:Messaging` header for the response to the pull request signal in Example 1:

```
<S11:Envelope>
<S11:Header>
<eb:Messaging S11:mustUnderstand="1" >
   <eb:UserMessage mpc="http://sender.example.com/mpc123">
```

```
      <eb:MessageInfo>
         <eb:Timestamp>2006-10-01T10:02:00</eb:Timestamp>
         <eb:MessageId>UUID-5@sender.example.com</eb:MessageId>
         <eb:RefToMessageId>UUID-4@receiver.example.com</eb:RefToMessageId>
      </eb:MessageInfo>
      <eb:PartyInfo>
      ...
      </eb:PartyInfo>
      <eb:CollaborationInfo>
      ...
      </eb:CollaborationInfo>
      <eb:PayloadInfo>
      ...
      </eb:PayloadInfo>
   </eb:UserMessage>
</eb:Messaging>
<wsr:Response S11:mustUnderstand="1">
       ...
</wsr:Response>
<wss:Security>
       ...
</wss:Security>
</S11:Header>
<S11:Body>
...
</S11:Body>
</S11:Envelope>
```

NOTE: In example 2, the reliability header, which assumes the use of WS-Reliability, is a `wsr:Response` element. It contains the reliability acknowledgment for the pull request signal. In this example there is no `wsr:Request` reliability header. A `wsr:Request` header could be present, in addition to `wsr:Response`, in case some specific reliability requirement is associated with the pulled message (see 11.3).

6.4. Message partition channels

6.4.1. Concept and purpose

Message partition channels (MPCs) allow for partitioning the flow of messages from a sending MSH to a receiving MSH into several flows that can be controlled separately and consumed differently. They also allow for merging flows from several sending MSHs, into a unique flow that will be treated as such by a receiving MSH. In particular, MPCs allow for:

1. setting transfer priorities: some messages may be transferred with higher priority than others regardless in which order they all have been submitted. For example, when using pulling mode, a receiving MSH may decide from which MPC to pull messages first, based on business needs and readiness to incur responsibility in managing these messages.

2. organizing the inflow of messages on receiving side, so that each flow can be consumed in a distinct way, yet without having to filter messages based on various header elements or payload content. The agreement between two parties on when messages are to be transferred and how they are to be consumed may then be reduced to which MPC will be used.

NOTE 1: The notion of MPC is abstract from any particular implementation device such as ports or queues: an implementation can implement MPCs using queues and a FIFO policy, though it is not required to.

NOTE 2: Although MPCs are most obviously beneficial to message pulling operations, MPCs can be used in association with pushed messages as well. The benefits of doing so, listed above, apply to the push case as well.

***Example**: A pair of business partners – a large buyer and a small supplier - have decided to create two MPCs for transferring messages sent by the buyer. Urgent messages that require immediate processing (e.g. high priority purchase orders, and updates to prior purchase orders) are assigned to one MPC; and less urgent messages (payments, catalog requests, confirmations, acknowledgments of receipts, etc.) are assigned to the other MPC. The buyer determines the level of urgency of a posting, which can be manifested inside the message. Per an agreement with the buyer, the supplier will pull and process first all messages from the "urgent" MPC; then, once that is exhausted, only the messages from the less urgent MPC. This way, the low-capacity receiving MSH (supplier) is able to prioritize the reception of its messages, focusing its resources on the most urgent messages and avoiding the overhead and risk in managing (persistence, recovery, security) less urgent but important messages that it cannot process in the short term.*

Any more complex filtering mechanism that requires checking a filter condition on header data, is out of scope of this document. Such filtering could be implemented in a sending MSH and/or in a receiving MSH as a complement to, or instead of, different MPCs. The notion of MPC is a simple and robust solution with low interoperability risk: it allows for partitioning messages based on prior agreement between producer and consumer on which type of message will use which MPC, without a need to communicate and process filter expressions for each message transfer.

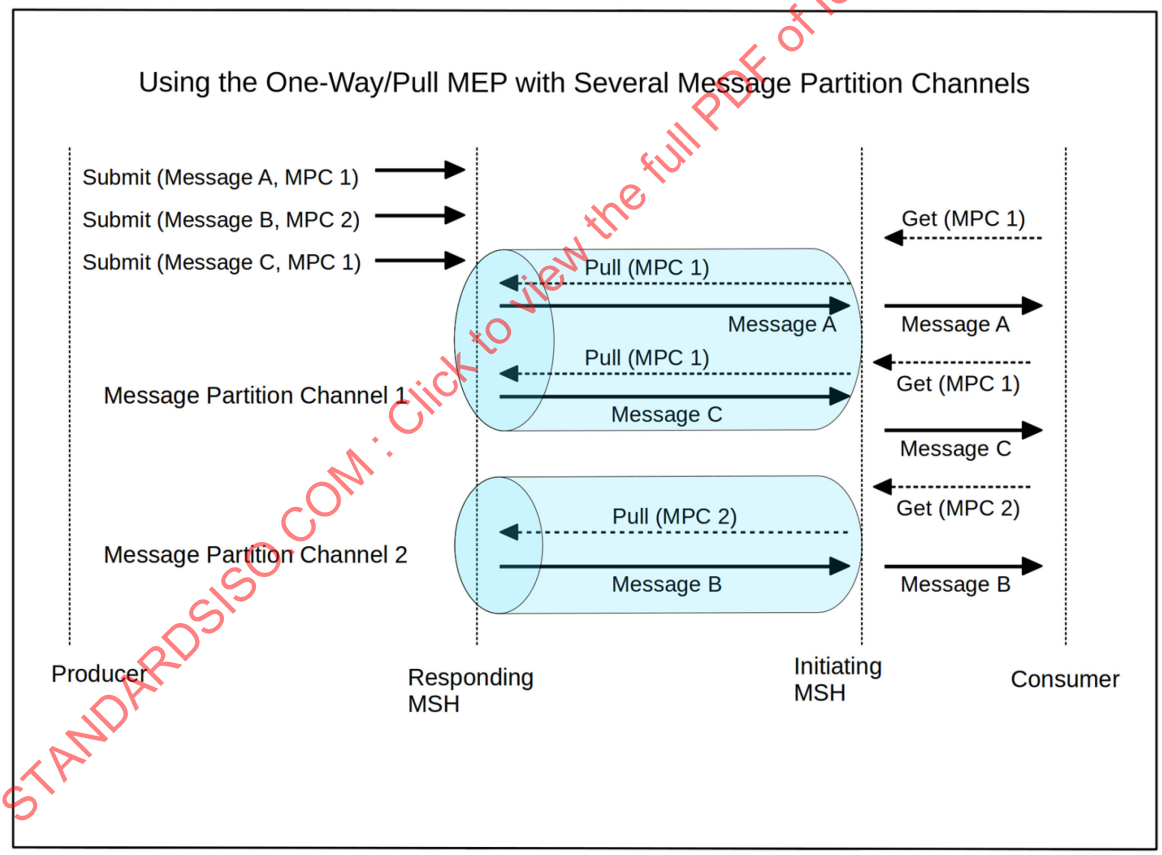

Figure 5: One-Way/Pull with Message Partition Channels

Figure 5 illustrates how MPCs and the One-Way/Pull MEP can be used by a consumer party to control the order of the messages it wants to receive and process. Messages on MPC 1 are "pulled" in priority by the consumer side.

There is no requirement for ordering messages in an MPC, unless specified otherwise by the reliability requirements to which these messages are subjected. The transfer of messages over an MPC is controlled by:

- The MEPs in which these messages participate. Messages over the same MPC can either be pulled or pushed, based on the different MEPs that govern the transfer of these messages.
- The regular addressing means used for sending messages (e.g. URL of receiving MSH when pushing messages). MPCs do not have any routing or addressing capability.

Before it is transferred from a sending MSH to a receiving MSH, regardless of whether it is pushed or pulled, a message is always assigned to an MPC. If no explicit assignment is requested (e.g. by the message producer at submission time or per configuration of the MSH), the default MPC name `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultMPC` is assigned.

6.4.2. Some use cases

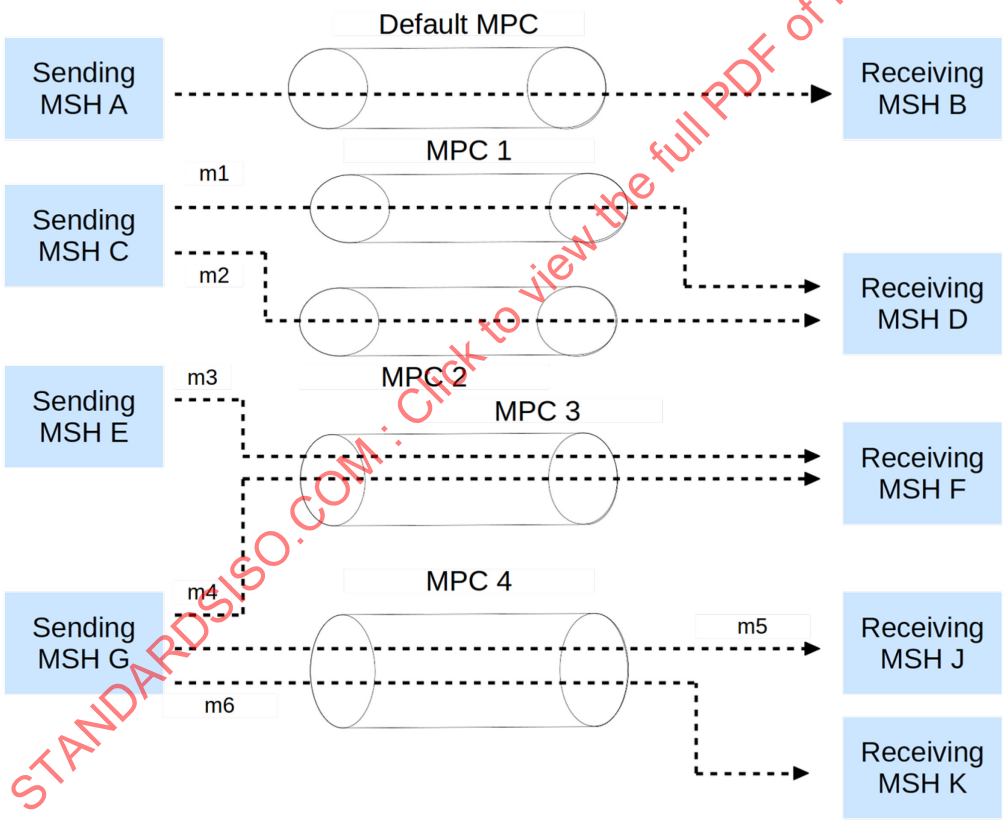

Figure 6: Message Partition Channel Use Cases

Figure 6 illustrates various cases in using MPCs. In Figure 6, each arrow represents the transfer of a user message, which could be either pushed or pulled.

Between MSHs A and B, no MPC has been explicitly defined or assigned. All messages transferred from A to B – whether pushed or pulled – will implicitly use the default MPC.

MSHs C and D have been configured to use MPCs 1 and 2 (in addition to the default MPC). Messages sent may be assigned to either one of these MPCs. In case these messages are pulled, MSH D may choose from which MPC to pull first.

MPC 3 is shared by two sending MSHs, E and G. The effect of using this MPC is to define on the receiving MSH F a merged inflow of messages from E and G, which may be presented to the consumer as a single flow. If messages m3 and m4 are pulled, MSH F has control over which MSH from which to pull first.

MPC 4 is used by MSH G to send either to MSH J or MSH K. When combined with message pulling, this use case allows for various scenarios. For example, the message flow might initially go exclusively from G to J. In case MSH J fails, another MSH (K) may immediately take over the message flow without any change on the sender side (assuming K is authorized) nor any knowledge by K of where the initial flow was intended for. Or, two receiving MSHs (J and K) that are remote from each other but used by equivalent applications may split the processing of messages submitted to the same sending MSH G. This may be, for example, two agencies equally qualified to process trouble tickets, indiscriminately pulling messages from the same MPC at the pace allowed by their processing capacity. MPC 4 may also be used by concurrent, pushed message flows. Using the same MPC does not introduce any dependency between the processing of m5 and m6 in J and K, but may be associated with a particular business meaning (i.e. is meaningful to consumers of J and K).

6.4.3. Definition and usage requirements

An MPC is a flow of messages from a set of sending MSHs to a set of receiving MSHs, in the sense given in flow networks theory. It is identified by a name–a string of characters–that is assigned to every message of the flow. For every message it sends or receives, an MSH shall be aware of which MPC this message is assigned to. MPC is a dynamic notion, the elements of which do not need to be fully defined prior to initiating this flow. For example, additional MSHs (either sending or receiving) may join the flow at any time, assuming they have knowledge of the MPC name, and assuming there is no other reason preventing them from transferring messages over this MPC (e.g. security).

The association between a user message and an MPC is apparent in the ebMS header of the message (see 8.2). Except for the default MPC, the MPC name shall appear in the header of a user message transferred over this MPC.

NOTE: As defined above, an MPC can involve more than a sending MSH and a receiving MSH. In particular, two unrelated pairs of sending/receiving MSHs (e.g. in the previous figure, C and D on the one hand, E and F on the other hand) could transfer messages using the same MPC name (e.g. MPC 3 in the figure could also be renamed MPC 2). Formally speaking, all these messages would be transferred over the same MPC. There can be some business significance in deciding whether two pairs of MSHs that have unconnected message flows use the same MPC to transfer these messages, even though as far as the MSHs are concerned, they will process these two separate sub-flows of messages independently from each other.

Only user messages may be assigned to MPCs, not signal messages.

A pull request signal message always indicates in its header (see 8.2.4.2) the MPC on which the message shall be pulled. If no MPC is explicitly identified, the default MPC shall be pulled from. The pulled message sent in response shall have been assigned to the indicated MPC.

The association of a message with an MPC shall be done either at submission time, e.g. requested by the message producer; or at any time between *Submit* and *Send*, e.g. based on configuration or processing mode (see Clause 7). This is left to the implementation.

Support for assigning messages to MPCs–e.g. by automatically mapping messages submitted by a producer to a particular MPC based on some rules, queries or filters–is out of scope of this document. Similarly, there is no requirement on what criteria (e.g. query expression, FIFO policy) can be used to

select messages when pulling messages from an MPC. This document only describes the properties of MPCs, and how their use affects the message protocol. It does not prescribe a particular way to implement MPCs or to use them.

A message associated with an MPC could fail to be transferred for various reasons (transport issue, security, intermediaries, etc.) and therefore could be removed from the MPC at any time. In other words, there is no additional delivery contract for messages over an MPC, other than that specified by the reliability agreement.

There is no specific quality of service associated with an MPC. Security and reliability remain associated with parties or with MSHs, in a way that is orthogonal to MPCs; although an implementation is free to associate QoS with MPCs as long as this conforms to an agreement between parties.

7. Processing modes

7.1. General

An MSH is operating–either for sending or receiving messages–with knowledge of some contextual information that controls the way messages are processed. This contextual information that governs the processing of a particular message is called processing mode (or P-Mode). Because different messages may be subject to different types of processing, an MSH generally supports several P-Modes.

A P-Mode represents some MSH input data that typically is not provided on a per-message basis, but that is common to a set of messages exchanged between or among parties. To this extent, the P-Mode may be interpreted as configuration data for a deployed MSH. On a sending MSH, together with the information provided by the application layer for each submitted message, the P-Mode fully determines the content of the message header. For example, the "security" part of the P-Mode will specify certificates and keys, as well as which messages will be subject to these. This in turn will determine the content of the `wsse:Security` header. The set of all P-Modes that are supported by an MSH during operation is called the P-Mode operation set of the MSH.

The association of a P-Mode with a message may be based on various criteria, usually dependent on header data (e.g. `eb:Service`, `eb:Action`, `eb:ConversationId`, or other message properties). Which security and/or which reliability protocol and parameters, as well as which MEP is being used when sending a message, is determined by the P-Mode associated with this message.

A data model for P-Modes is described in Annex D. Although this document does not require support for any particular representation of a P-Mode, a conformance profile for this document may require support for a particular representation. An MSH shall conform the processing of its messages to the values in the P-Mode associated with this message. The details of which P-Mode parameters shall be supported by an implementation is governed by the features associated with the conformance profile claimed by this implementation, i.e. by its profile feature set (see Annex G on Conformance). An MSH shall not process a message to normal completion if it has no matching P-Mode in its P-Mode operation set: i.e. the MSH shall not deliver such a message when in receiving role, or shall not send it when in sending role. When it cannot match a message to a P-Mode, an MSH shall generate a ProcessingModeMismatch (`EBMS:0010`) error.

NOTE: It is important to distinguish between Conformance Profiles (Annex G) and P-Modes. A conformance profile qualifies an MSH implementation and does not vary with the usage made of the MSH. A P-Mode qualifies the dynamic exchange and processing of messages, and is generally user defined. A P-Mode is set within the capabilities allowed by the conformance profile claimed by the MSH on which it is deployed.

7.2. Messaging service processing model

Although different P-Modes may apply from one message to the other, the overall processing model remains the same for all messages. The P-Modes set may be seen as configuring the execution parameters for the general model.

The ebXML messaging service may be conceptually broken down into the following three parts:

1. An abstract service interface,
2. Functions provided by the MSH and
3. The mapping to underlying transport service(s).

Figure 7 depicts a logical arrangement of the functional modules existing within one possible implementation of the ebXML Messaging Services architecture. These modules are arranged in a manner to indicate their inter-relationships and dependencies.

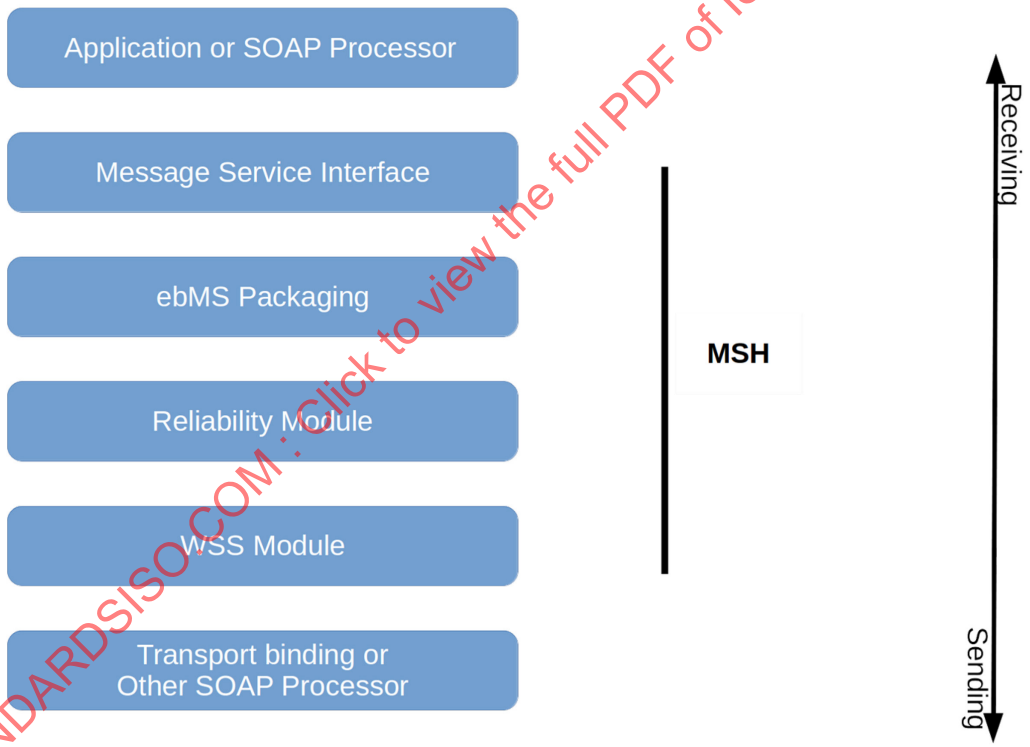

Figure 7: Component Relationships

Following is a description of each module illustrated in Figure 7. It should be noted that the stack diagram in the Figure is abstract, and this document does not mandate that implementations adopt the architecture suggested by it, although the processing order shown here should be used, especially in regard to security and reliability modules.

- **Application or SOAP processor** - This is where the business logic for a message exchange / business process exists.
- **Messaging service interface** - This is the interface through which messages are channelled between the MSH core and the ebXML Application.

- **ebMS packaging** - Handling, (de)enveloping and execution of payload services are performed by this module.
- **Reliable message processing** - This module fulfills the quality of service requirements for a message.
- **Web Services security processing** - Encryption/decryption of any SOAP message content and generation/verification of any digital signatures occurs in this module.
- **Transport protocol bindings** - These are the actual transport protocol bindings. This document defines bindings for HTTP and SMTP in Annex C, and supports the addition of other protocols.

7.3. Processing mode features

The P-Mode is partitioned into functional groups called P-Mode features. Each P-Mode feature covers one of the functional areas that is critical to achieving interoperability between two partners: security, reliability, transport, business collaboration, error reporting, message exchange patterns (MEPs) and message partition channels (MPCs).

The main P-Mode features are here identified by names of the form: P-Mode.<featurename>:

- **P-Mode.Protocol**: includes all transport related information that is necessary to achieve transport-level interoperability. This feature determines the type of transport involved (e.g. HTTP, SMTP, FTP) between two MSHs, and related configuration parameters. This feature usually treats all messages between two MSHs similarly. It also includes information about which SOAP version is to be used (SOAP 1.1 or SOAP 1.2).
- **P-Mode.Reliability**: includes all reliability contracts, or references to them, that will govern the reliability of messages exchanged. This feature determines the content of the reliability headers.
- **P-Mode.Security**: includes all security contracts, or references to them, including the security context and related resources (certificates, SAML assertions, etc.) that govern the message exchange. This feature determines the content of the `wsse:Security` header.
- **P-Mode.BusinessInfo**: includes all message-relevant data related to a collaboration between two parties. It also indicates which MPCs are to be used by these parties. This feature will complement or validate message data that is expected to be provided by the application on a per-message basis for these header elements:
  - `eb:UserMessage/eb:PartyInfo`
  - `eb:UserMessage/eb:CollaborationInfo`
  - `eb:UserMessage/eb:MessageProperties`
- **P-Mode.ErrorHandling**: defines how each ebMS Error type is to be reported by this MSH. E.g. if the reporting is done using ebMS signal messages, it defines the address of the destination MSH. It also may include the policy chosen for raising ebMS Errors from the errors generated by functional modules (reliability, security). This P-Mode feature shall define reporting mode parameters that will allow a receiving MSH to decide:
  - whether an error generated on reception of a message shall be returned as response over the same SOAP MEP. (e.g. **errorHandling.report.asResponse** = true/false).

- whether an error generated on reception of a message shall be returned to sender or to a third party over a new SOAP MEP. (e.g. **errorHandling.report.ReceiverErrorsTo** = <URL>).

- whether the consumer and/or producer (e.g. **errorHandling.Report.ProcessErrorNotifyConsumer**) of a message shall be notified of an error generated on reception of the message.

In this document, a P-Mode feature is abstractly considered to apply to both sending and receiving roles, although implementations may choose to represent only the subset relevant to the role in which they operate. A single P-Mode instance is also intended to govern all messages involved in an ebMS MEP. (The ebMS MEP and its transport channel binding are attributes of a P-Mode.) Because messages involved in an MEP (e.g. request and reply) may use different qualities of service, a single P-Mode may use different vectors of values for its parameters, depending on the message in the MEP. An outline of the data model for P-Modes is given in Annex D.

Agreeing on a P-Mode operation set is essential for two parties in order for their MSHs to interoperate. P-Modes are the MSH-level expression of a prior agreement between partners. A reference to such an agreement may be present in the message header (see `eb:AgreementRef` element in 8.2.3.8).

7.4. Default features for processing mode

In order to facilitate interoperability testing, or during the early phase of a deployment, it may be useful to drive message exchanges without relying on user-agreed P-Modes, without interfacing with any application, and (initially) without the added complexity of security and reliability features. To this end, a default semantics of each P-Mode feature is defined as follows:

- **Default P-Mode.MEP:** `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay`:

- **Default P-Mode.MEPbinding**: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push`

- **Default P-Mode.Protocol**: HTTP 1.1 transport is assumed, with default configuration (on standard port), using SOAP 1.2.

- **Default P-Mode.Reliability**: No reliable messaging assumed (no reliability header will be present).

- **Default P-Mode.Security**: No secure messaging assumed (no security header will be present.)

- **Default P-Mode.BusinessInfo**: In the absence of any application input at message level as well as for this P-Mode feature, the following default header element values will be used (shown here for a message sent by an initiator to a responder party). Any of these may be overridden by application input.

  - `eb:UserMessage/eb:PartyInfo`: The `eb:From` element contains a PartyId with value: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultFrom`.
    The `eb:To` element contains a PartyId with value: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultTo`.

- `eb:UserMessage/eb:CollaborationInfo`: Contains no `eb:AgreementRef`. The `eb:Service` element has the value: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service`.
  The `eb:Action` element has the value: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/test` (8.2.3 details the semantics of these values.)
  The `eb:ConversationId` element has the value: 1.
  The default MPC is in use.

- `eb:UserMessage/eb:MessageProperties`: This element is absent.

- `eb:UserMessage/eb:PayloadInfo`: This element is absent.

- **Default P-Mode.ErrorHandling**: No reporting via ebMS message is required. The MSH may handle error reporting in a way that does not involve the partner MSH, such as notification to local consumer or producer.

In the absence of a user-agreed P-Mode feature, an MSH should operate based on the above default semantics for this feature except in the following cases:

1. The MSH is designed to conform to this document along profiles (see Annex G) that are not compatible with the default P-Mode feature. For example, such an incompatibility would occur for the default P-Mode.MEP with a conformance profile that only requires the One-Way/Pull MEP.

2. The MSH has been pre-configured to operate with a non-default P-Mode feature. This would be the case when an MSH is distributed along with a predefined P-Mode feature, e.g. built-in security. This amounts to using a user-defined P-Mode feature.

A sending MSH and a receiving MSH may use a mix of default and non-default P-Mode features.

8. Message packaging

8.1. Message envelope and message parts

8.1.1. MIME structure and SOAP profile

In the ebMS SOAP header `eb:Messaging`, the prefix "eb" is an example prefix that corresponds to the ebMS 3.0 namespace, as defined in 4.3. The ebMS Message can be packaged as a plain SOAP 1.1 message (conform the W3C SOAP 1.1 specifiation) or as a SOAP 1.2 message (conform the W3C SOAP 1.2 specification), or within a MIME multipart to allow payloads or attachments to be included. Because either packaging option can be used, implementations shall support both multipart and non-multipart messages.

The ebMS Message may contain SOAP extension elements other than the `eb:Messaging` header block. For example, header blocks supporting message reliability and message security may be produced and consumed by an MSH in order to fulfill deployment requirements for those features.

An ebMS Message is packaged as a SOAP 1.1 or 1.2 message independent from communications protocols. When represented as a MIME multipart message envelope, this envelope shall be structured in compliance with the SOAP Messages with Attachments W3C Note, referred to as a Message Package, which shall conform to the MIME envelope structure defined in RFC 2045, and specifically to the RFC 2387 MIME Multipart/Related specification.

There are two logical sections within the Message Package:

- The first section is the ebMS Header (i.e. the `eb:Messaging` SOAP header block), itself contained in the SOAP Header.
- The second section is the ebMS payload, which itself comprises two sections: (a) the SOAP Body element within the SOAP `Envelope`, and in case of MIME packaging, (b) zero or more additional MIME parts containing additional application-level payloads. The SOAP `Body` and MIME parts are also referred to as ebMS payload containers. The SOAP `Body` is the only payload container that requires XML-structured content, though non-XML content may be included within an appropriately typed (binary or otherwise) element inside the `Body`. Any additional MIME parts shall have an IANA assigned MIME Media type.

The general structure and composition of an ebMS user message is described in Figure 8, and a signal message in Figure 9.

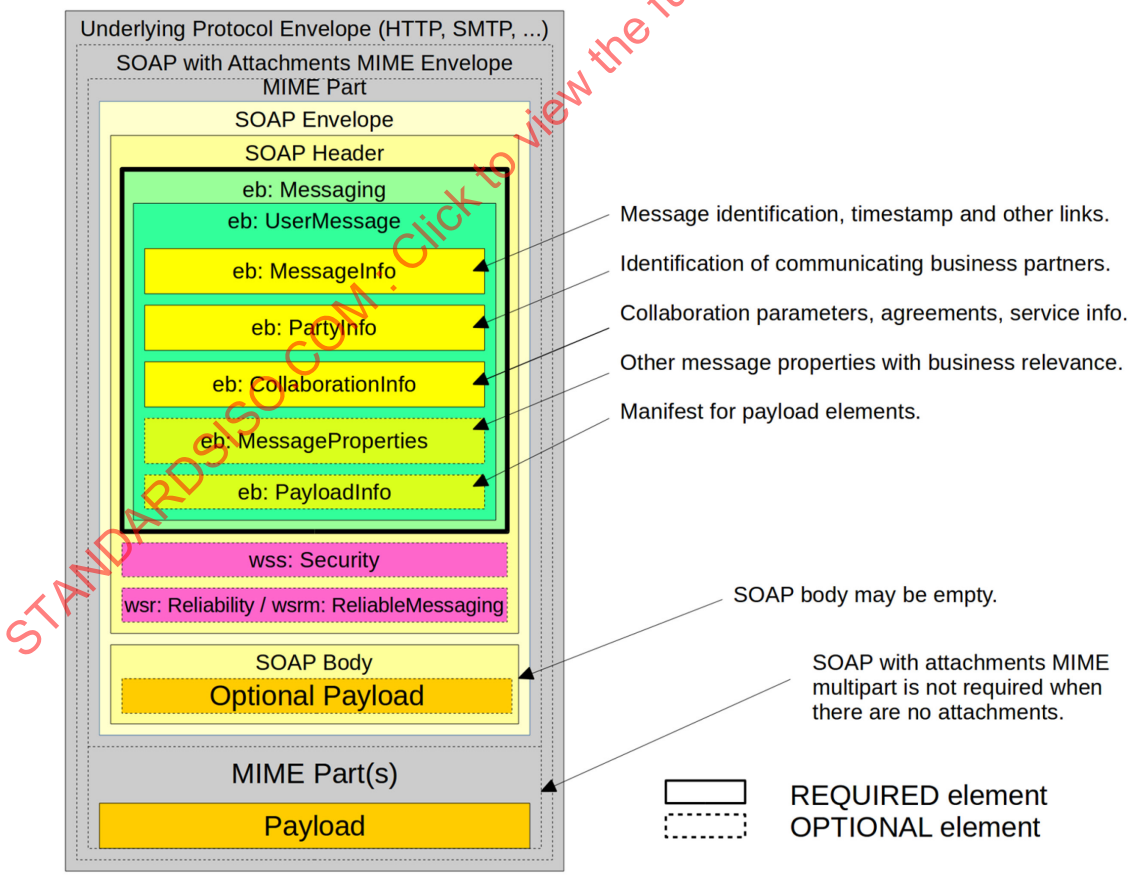

Figure 8: User Message Structure

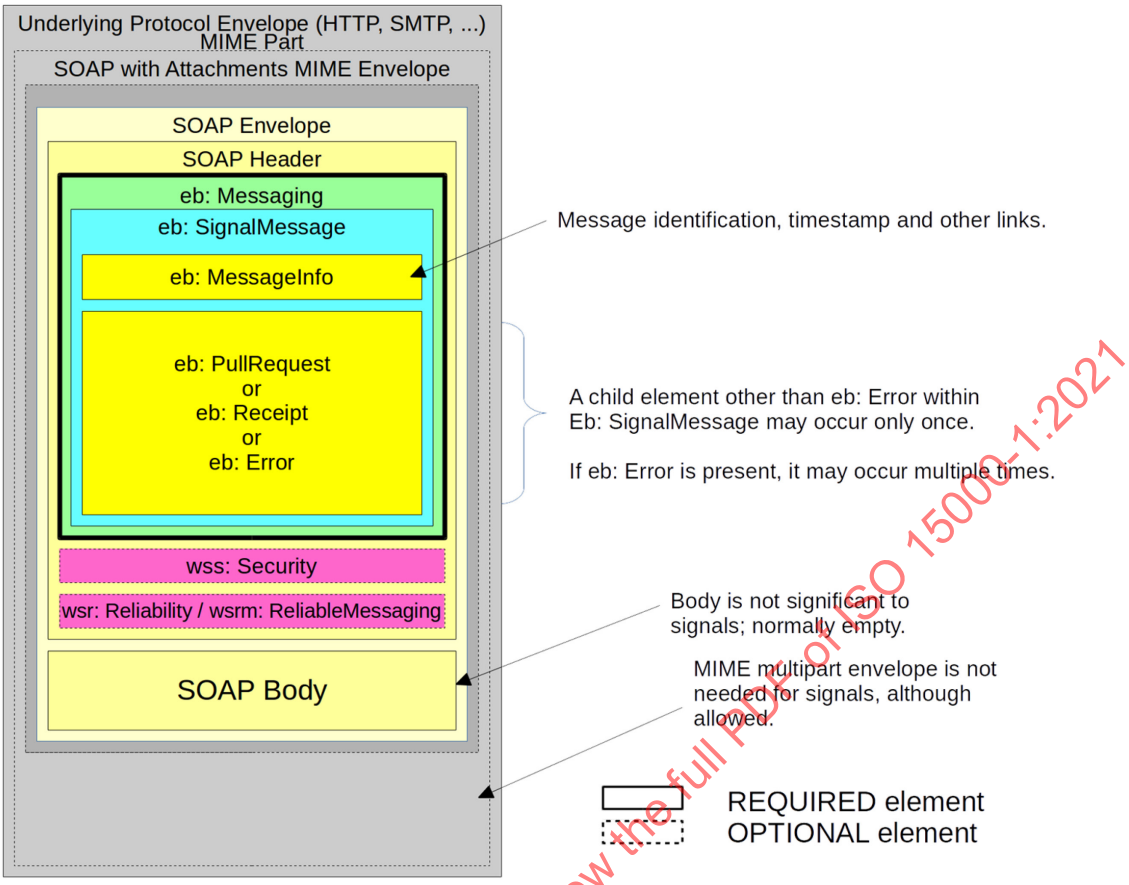

Figure 9: Signal Message Structure

The processing of the SOAP `eb:Messaging` header block is done according to the SOAP processing semantics: an MSH behaves as a SOAP processor or SOAP node that shall understand this header block. Other header blocks (except for those relevant to reliability and security of an ebMS Message) are not affected by the ebXML processing. Consequently, it is possible for a sending MSH implementation to generate an ebMS message from a well-formed input SOAP message simply by adding an `eb:Messaging` header; likewise, some receiving MSH implementation could deliver a well-formed SOAP message as output by removing (and processing) the `eb:Messaging` header.

All MIME headers of the Message Package shall conform with the SOAP Messages with Attachments W3C Note. In addition, the Content-Type MIME header of the Message Package shall contain a type parameter whose value matches the MIME media type of the MIME body part containing the SOAP Envelope document. In accordance with the SOAP 1.1 specification, the MIME media type of the SOAP 1.1 Message shall have the value "text/xml". The initial headers should contain a Content-ID MIME header structured in accordance with MIME, and in addition to the required parameters for the Multipart/Related media type, the start parameter, which is optional in the IETF RFC 2387 MIME Multipart/Related specification, shall be present. This permits more robust error detection. The following fragment is an example of the MIME headers for the multipart/related Message Package:

Example: MIME Header fragment for the multipart/related Message Package

```
Content-Type: multipart/related; type="text/xml";
boundary="boundaryValue";start="<messagepackage-123@example.com>"
--boundaryValue
Content-ID: messagepackage-123@example.com
```

Because implementations shall support non-multipart messages, an ebMS Message with no payload may be sent either as a plain SOAP message or as a SOAP with attachments MIME multipart message with only one body part (the SOAP Envelope).

8.1.2. MIME and XML considerations

8.1.2.1. Additional MIME parameters

Any MIME part described by this document may contain additional MIME headers in conformance with the RFC 2045 specification. Implementations may ignore any MIME header not defined in this document. Implementations shall ignore any MIME header they do not recognize. For example, an implementation could include Content-Length in a message. However, a recipient of a message with Content-Length could ignore it.

8.1.2.2. Reporting MIME errors

If a MIME error is detected in the Message Package then it shall be reported as specified in the SOAP with Attachments W3C Note.

8.1.2.3. XML prolog

The SOAP Message's XML Prolog, if present, may contain an XML declaration. This document has defined no additional comments or processing instructions appearing in the XML prolog. For example:

```
Content-Type: text/xml; charset="UTF-8"

<?xml version="1.0" encoding="UTF-8" ?>
```

8.1.2.4. XML declaration

The XML declaration may be present in a SOAP Message. If present, it shall contain the version specification required by the W3C 1.0 XML Recommendation and may contain an encoding declaration. The semantics described in 8.1.2.5 shall be implemented by a compliant ebXML message service.

8.1.2.5. Encoding declaration

If both the encoding declaration and the MIME root part charset parameter are present, the XML prolog for the SOAP Message shall contain the encoding declaration, and shall be equivalent to the charset attribute of the MIME Content-Type of the root part (see 8.1.4). If provided, the encoding declaration shall not contain a value conflicting with the encoding used when creating the SOAP message. UTF-8 should be used when encoding the SOAP Message. Use of UTF-8 shall conform to RFC 2279. If the character encoding cannot be determined by an XML processor using the rules specified in W3C XML 1.0 specification, clause 4.3.3, the XML declaration and its contained encoding declaration shall be provided in the ebXML SOAP Header Document.

NOTE: The encoding declaration is not required in an XML document according to the XML v1.0 specification.

8.1.3. ebXML SOAP envelope extension

8.1.3.1. General

In conformance with the W3C XML 1.0 specification, all extension element content shall be namespace qualified. A namespace declaration (xmlns pseudo-attribute) for the ebXML SOAP extension may be included in the SOAP Envelope or Header element, or directly in the ebXML SOAP extension element.

8.1.3.2. namespace pseudo attribute

Conform the W3C *Namespaces in XML* specification, the namespace declaration for the ebXML SOAP Envelope extension (xmlns pseudo attribute) shall have the value of:

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

8.1.3.3. xsi:schemaLocation attribute

The SOAP namespace:

```
http://schemas.xmlsoap.org/soap/envelope/
```

resolves to a W3C XML Schema specification. ebXML MSH implementations should include the XMLSchema-instance namespace qualified `schemaLocation` attribute in the SOAP `Envelope` element, to indicate to validating parsers a location of the schema document that should be used to validate the document. Failure to include the schemaLocation attribute could prevent XML schema validation of received messages.

For example:

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://schemas.xmlsoap.org/soap/envelope/">
       <S11:Header/>
       <S11:Body/>
</S11:Envelope>
```

In addition, the ebXML SOAP Header extension element content may be similarly qualified, so as to identify the location where validating parsers can find the schema document containing the ebXML namespace-qualified SOAP extension element definition. The ebXML SOAP extension element schema, found in Annex A, has been defined using the W3C Recommendation version of the W3C XML Schema Part 1 Structures. ebXML MSH implementations shall validate the structure of the ebXML SOAP Header extension element in conformance to the *XML Schema Part 1: Structures Second Edition* specification.

The XMLSchema-instance namespace qualified `schemaLocation` attribute should include a mapping of the ebXML SOAP Envelope extension namespace to its schema document in the same element that declares the ebXML SOAP Envelope extensions namespace.

The `schemaLocation` for the namespace described in 8.1.3.2 is:

https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-200704_2019.xsd

Separate `schemaLocation` attributes should be included. For example:

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                      http://schemas.xmlsoap.org/soap/envelope/">
  <S11:Header>
    <eb:Messaging                  xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/"
      xsi:schemaLocation="                  http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/      https://standards.iso.org/iso/15000/-1/ed-
1/en/ebms-header-3_0-200704_2019.xsd">
      <eb:UserMessage>
        <eb:MessageInfo >...</eb:MessageInfo>
             ...
        <eb:PayloadInfo >...</eb:PayloadInfo>
             ...
      </eb:UserMessage>
    </eb:Messaging>
  </S11:Header>
  <S11:Body>
       ...
  </S11:Body>
</S11:Envelope>
```

8.1.3.4. SOAP header element

The SOAP `Header` element is the first child element of the SOAP `Envelope` element. It shall have a namespace qualifier that matches the SOAP `Envelope` namespace declaration for the namespace `http://schemas.xmlsoap.org/soap/envelope/`.

8.1.3.5. SOAP body element

The SOAP `Body` element is the second child element of the SOAP `Envelope` element. It shall have a namespace qualifier that matches the SOAP `Envelope` namespace declaration for the namespace `http://schemas.xmlsoap.org/soap/envelope/`.

NOTE:
Unlike ebMS v2, ebXML Messaging 3.0 does not define or make use of any elements within the SOAP Body, which is wholly reserved for user-specified payload data.

8.1.3.6. ebXML SOAP extensions

An ebMS Message extends the SOAP Message with the extension element `eb:Messaging`, where "eb" is the namespace prefix for ebMS 3.0.

Other headers that support some aspects of ebMS messaging, such as the security header (`wsse:Security`) and reliability headers, may be present. These are not qualified under the ebMS namespace.

8.1.4. ebMS header

In case of MIME packaging, the root body part of the message package is the SOAP message, as defined in the W3C SOAP Messages with Attachments W3C Note. This root part always contains the ebMS header.

The MIME Content-Type header for the root part shall have the value "text/xml" to match the MIME media type of the MIME body part containing the SOAP 1.1 Envelope, or "application/soap+xml" in the case of a SOAP 1.2 Envelope. The Content-Type header may contain a "charset" parameter. For example:

```
Content-Type: text/xml; charset="UTF-8"
```

The MIME charset parameter identifies the character set used to create the SOAP message. The semantics of this attribute are described in the "charset parameter / encoding considerations" of text/xml as specified in RFC 3023. The list of valid values can be found at the Web site of the Internet Assigned Names Authority (IANA).

If both are present, the value of the MIME charset parameter shall be equivalent to the encoding declaration of the SOAP Message. If provided, the MIME charset parameter shall not contain a value conflicting with the encoding used when creating the SOAP Message.

For maximum interoperability UTF-8 as specified in RFC 2387 should be used when encoding this document. Due to the processing rules defined for media types derived from text/xml (see RFC 3023), this MIME attribute has no default.

The following fragment represents an example of a root part, for a MIME packaging of ebMS:

```
Content-ID: <messagepackage-123@example.com>
Content-Type: text/xml; charset="UTF-8"

<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/">
   <S11:Header>
      <eb:Messaging>
      …
      </eb:Messaging>
   </S11:Header>
   <S11:Body>
   …
   </S11:Body>
</S11:Envelope>
```

8.1.5. Payload containers

In addition to the SOAP `Body`, other payload containers may be present within a message package in conformance with the SOAP Messages with Attachments W3C note.

If there is no application payload within the message package, then the SOAP `Body` shall be empty, and there shall not be additional payload containers.

There should also be no additional MIME attachments that are not payload containers (i.e., that are not referenced by an `eb:PayloadInfo` element, as described in 8.2.3.13); but if any such attachments are present, they are outside the scope of MSH processing. An MSH shall not process application data that is not referenced by `eb:PayloadInfo`.

The contents of each payload container (including the SOAP `Body`) shall be identified in the `/eb:Messaging/eb:UserMessage/eb:PayloadInfo` element.

The ebXML messaging service specification makes no provision, nor limits in any way, the structure or content of application payloads. Payloads may be simple, plain-text objects or complex, nested, multipart objects. The specification of the structure and composition of payload objects is the prerogative of the organization defining the business process or information exchange using the ebXML messaging service.

Example: A SOAP message containing an ebMS header:

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                       http://schemas.xmlsoap.org/soap/envelope/">
   <S11:Header
xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-200704_2019.xsd">
      <eb:Messaging S11:mustUnderstand="1">
         <eb:UserMessage>
            ...
            <eb:PayloadInfo>
               ...
            </eb:PayloadInfo>
            ...
         </eb:UserMessage>
      </eb:Messaging>
   </S11:Header>
   <S11:Body>
      ...
   </S11:Body>
</S11:Envelope>
```

8.2. The eb:Messaging container element

8.2.1. General

The mandatory `eb:Messaging` element is a child of the SOAP `Header`. It is a container for either a user message or a signal message.

In the case of a user message, the ebXML header block shall contain an `eb:UserMessage` child element:

```
<eb:Messaging>
       <eb:UserMessage>
              <eb:MessageInfo>
              <!-- some headers here like Timestamp and MessageId -->
              </eb:MessageInfo>
              <!-- header elements of the ebMS user message -->
       </eb:UserMessage>
</eb:Messaging>
```

In the case of a signal message, the ebXML header block (`eb:Messaging`) shall contain at least one `eb:SignalMessage` child element:

```
<eb:Messaging>
       <eb:SignalMessage>
              <eb:MessageInfo>
                 <!-- some headers here like Timestamp and MessageId -->
              </eb:MessageInfo>
              <eb:signalname>
                  <!-- header elements of this ebMS signal message -->
              </eb:signalname>
       </eb:SignalMessage>
</eb:Messaging>
```

For example, *signalname* can be "PullRequest".

8.2.2. eb:Messaging element specification

The `eb:Messaging` element has the following attributes:

- `eb:Messaging/@S11:mustUnderstand`: indicates whether the contents of the element shall be understood by the MSH. This attribute shall be present, with namespace qualified to the SOAP namespace (`http://schemas.xmlsoap.org/soap/envelope/`). It shall have value of '1' (true) indicating the element shall be understood or rejected.

The `eb:Messaging` element has the following children elements:

- `eb:Messaging/eb:UserMessage`: The optional `eb:UserMessage` element contains all header information for a user message. If this element is not present, an element describing a signal message shall be present.

- `eb:Messaging/eb:SignalMessage/eb:[signalname]`: The optional element is named after a type of signal message. It contains all header information for the signal message. If this element is not present, an element describing a user message shall be present. Three types of signal messages are specified in this document: Pull signal (`eb:PullRequest`), error signal (`eb:Error`) and receipt signal (`eb:Receipt`).

Both `eb:UserMessage` element and `eb:SignalMessage` element may be present within the `eb:Messaging` element.

Example: ebMS Message Header:

```
<!-- (contained within S11:Header) -->

  <eb:Messaging S11:mustUnderstand="1" >

    <eb:UserMessage>

      <eb:MessageInfo>
        <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
        <eb:MessageId>UUID-2@example.com</eb:MessageId>
        <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
      </eb:MessageInfo>

      <eb:PartyInfo>
        <eb:From>
          <eb:PartyId>uri:example.com</eb:PartyId>
            <eb:Role>http://example.org/roles/Buyer</eb:Role>
        </eb:From>

        <eb:To>
          <eb:PartyId type="someType">QRS543</eb:PartyId>
          <eb:Role>http://example.org/roles/Seller</eb:Role>
        </eb:To>
      </eb:PartyInfo>

      <eb:CollaborationInfo>
        <eb:AgreementRef>http://registry.example.com/cpa/123456
        </eb:AgreementRef>
        <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
        <eb:Action>NewPurchaseOrder</eb:Action>
        <eb:ConversationId>4321</eb:ConversationId>
      </eb:CollaborationInfo >

      <eb:MessageProperties>
        <eb:Property name="ProcessInst">PurchaseOrder:123456
        </eb:Property>
        <eb:Property name="ContextID"> 987654321
        </eb:Property>
      </eb:MessageProperties >
```

```
      <eb:PayloadInfo>
        <eb:PartInfo href="cid:foo@example.com">
          <eb:Schema location="http://example.org/bar.xsd" version="2.0"/>
        </eb:PartInfo>
        <eb:PartInfo href="#idref">
        </eb:PartInfo>
      </eb:PayloadInfo>

    </eb:UserMessage>
  </eb:Messaging>
```

8.2.3. eb:Messaging/eb:UserMessage

8.2.3.1. General

This element has the following attributes:

- `eb:Messaging/eb:UserMessage/@mpc`: This optional attribute contains a URI that identifies the message partition channel to which the message is assigned. The absence of this element indicates the use of the default MPC. When the message is pulled, the value of this attribute shall indicate the MPC requested in the pull request message.

This element has the following children elements:

- `eb:Messaging/eb:UserMessage/eb:MessageInfo`: This element shall occur once, and contains data that identifies the message, and relates to other messages' identifiers.
- `eb:Messaging/eb:UserMessage/eb:PartyInfo`: This element shall occur once, and contains data about originating party and destination party.
- `eb:Messaging/eb:UserMessage/eb:CollaborationInfo`: This element shall occur once, and contains elements that facilitate collaboration between parties.
- `eb:Messaging/eb:UserMessage/eb:MessageProperties`: This element may occurs at most once, and contains message properties that are user-specific. As parts of the header such properties allow for more efficient monitoring, correlating, dispatching and validating functions (even if these are out of scope of ebMS specification) which would otherwise require payload access.
- `eb:Messaging/eb:UserMessage/eb:PayloadInfo`: This element may occur at most once, and identifies payload data associated with the message, whether included as part of the message as payload document(s) contained in a payload container, or remote resources accessible via a URL. The purpose of the `eb:PayloadInfo` is (a) to make it easier to directly extract a particular payload associated with this user message, (b) to allow an application to determine whether it can process the payload without having to parse it.

8.2.3.2. eb:Messaging/eb:UserMessage/eb:MessageInfo

This element has the following children elements:

- `eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:Timestamp`: The `eb:Timestamp` element shall be present with a value representing the date at which the message header was created, and is conforming to a dateTime, conform W3C XML Schema. It shall be expressed as UTC. Indicating UTC in the `eb:Timestamp` element by including the 'Z' identifier is optional.

- `eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:MessageId`: This element shall be present with a value representing – for each message - a globally unique identifier conforming to message identifier conform RFC 2822.

- `eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:RefToMessageId`: This element may occur at most once. When present, it shall contain the `eb:MessageId` value of an ebMS Message to which this message relates, in a way that conforms to the MEP in use (see C.4).

NOTE: In the `Message-Id` and `Content-Id` MIME headers, values are always surrounded by angle brackets. However references in `mid:` or `cid:` scheme URI's and the `eb:MessageId` and `eb:RefToMessageId` elements shall not include these delimiters.

8.2.3.3. eb:Messaging/eb:UserMessage/eb:PartyInfo

This element has the following children elements:

- `eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From`: The element shall occur once, and contain information describing the originating party.

- `eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To`: The element shall occur once, and contain information describing the destination party.

8.2.3.4. eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From

This element has the following children elements:

- `eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId`: The `eb:PartyId` element shall occur one or more times. If it occurs multiple times, each instance shall identify the same organization.

- `eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:Role`: The `eb:Role` element shall occur once, and identifiy the authorized role (fromAuthorizedRole or toAuthorizedRole) of the Party sending (when present as a child of the From element) or receiving (when present as a child of the To element) the message. The value of the Role element is a non-empty string, with a default value of `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultRole`.
  Other possible values are subject to partner agreement.

EXAMPLE: The following fragment demonstrates usage of the `eb:From` element.

```
<eb:From>
  <eb:PartyId         type="urn:oasis:names:tc:ebxml-cppa:partyid-type:duns">
123456789</eb:PartyId>
  <eb:PartyId type="SCAC">RDWY</eb:PartyId>
  <eb:Role>http://example.org/roles/Buyer</eb:Role>
</eb:From>
```

8.2.3.5. eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId

This element has a string value content that identifies a party, or that is one of the identifiers of this party.

It has a single attribute, `@type`. The type attribute indicates the domain of names to which the string in the content of the PartyId element belongs. The value of the type attribute should be a URI. Furthermore, these values should be taken from the EDIRA, EDIFACT or ANSI ASC X12 registries.

Technical specifications for the first two registries can be found in ISO 6523 and ISO 9735, respectively. Further discussion of PartyId types and methods of construction can be found in an annex of the ebXML Collaboration Protocol Profile and Agreement Technical Committee, *Collaboration-Protocol Profile and Agreement Specification Version 2.1*. The value of any given @type attribute shall be unique within a list of PartyId elements.

An example of the `eb:PartyId` element is:

```
<eb:PartyId type="urn:oasis:names:tc:ebxml-cppa:partyid-type:duns">
123456789</eb:PartyId>
```

If the `eb:PartyId/@type` attribute is not present, the content of the PartyId element shall be a URI, conform RFC 2396, otherwise the receiving MSH should report a "ValueInconsistent" error with severity "error". The content of the `eb:PartyId` element should be a URI.

8.2.3.6. eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To

This element has the same children elements as `eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From`, in 8.2.3.4.

EXAMPLE: The following fragment demonstrates usage of the `eb:To` element.

```
<eb:To>
  <eb:PartyId>mailto:joe@example.com</eb:PartyId>
  <eb:Role>http://example.org/roles/Seller</eb:Role>
</eb:To>
```

8.2.3.7. eb:Messaging/eb:UserMessage/eb:CollaborationInfo

This element has the following children elements:

- `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:AgreementRef`: This element may occur zero or once. The `eb:AgreementRef` element is a string that identifies the entity or artifact governing the exchange of messages between the parties.

- `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Service`: This element shall occur once. It is a string identifying the service that acts on the message and it is specified by the designer of the service.

- `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Action`: This element shall occur once. The element is a string identifying an operation or an activity within a service that may support several of these.

- `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:ConversationId`: This element shall occur once. The element is a string identifying the set of related messages that make up a conversation between parties.

8.2.3.8. eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:AgreementRef

`eb:AgreementRef` is a string value that identifies the agreement that governs the exchange. The P-Mode under which the MSH operates for this message should be aligned with this agreement.

The value of an `eb:AgreementRef` element shall be unique within a namespace mutually agreed by the two parties. This could be a concatenation of the `eb:From` and `eb:To` `eb:PartyId` values, a URI containing the Internet domain name of one of the parties, or a namespace offered and managed by

some other naming or registry service. The eb:AgreementRef should be a URI. The eb:AgreementRef may reference an instance of a CPA as defined in the OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee, *Collaboration-Protocol Profile and Agreement Specification*.

An example of the eb:AgreementRef element follows:

```
<eb:AgreementRef>http://registry.example.com/cpas/our_cpa.xml</eb:AgreementRef>
```

If a CPA is referred to and a receiving MSH detects an inconsistency, then it shall report it with an "ValueInconsistent" error of severity "error". If the AgreementRef is not recognized, then the receiving MSH shall report it as a "ValueNotRecognized" error of severity "error".

The eb:AgreementRef element may have two attributes:

- @type: This optional attribute indicates how the parties sending and receiving the message will interpret the value of the reference (e.g. the value could be "ebcppa2.1" for parties using a CPA-based agreement representation). There is no restriction on the value of the type attribute; this choice is left to profiles of this document. If the type attribute is not present, the content of the eb:AgreementRef element shall be a URI. If it is not a URI, then the MSH shall report a "ValueInconsistent" error of severity "error".

- @pmode: This optional attribute allows for explicit association of a message with a P-Mode. When used, its value contains the PMode.ID parameter.

8.2.3.9. eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Service

This element identifies the service that acts on the message. Its actual semantics is beyond the scope of this document. The designer of the service may be a standards organization, or an individual or enterprise.

Examples of the eb:Service element include:

EXAMPLE 1:

```
<eb:Service>urn:example.org:services:SupplierOrderProcessing</eb:Service>
```

EXAMPLE 2:

```
<eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
```

The eb:Service element may contain a single @type attribute, that indicates how the parties sending and receiving the message will interpret the value of the element. There is no restriction on the value of the type attribute. If the type attribute is not present, the content of the eb:Service element shall be a URI conform RFC 2396. If it is not a URI then the MSH shall report a "ValueInconsistent" error of severity "error".

When the value of the element is http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service, then the receiving MSH shall not deliver this message to the consumer. With the exception of this delivery behavior, and unless indicated otherwise by the eb:Action element, the processing of the message is not different from any other user message.

8.2.3.10. eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Action

This element is a string identifying an operation or an activity within a `eb:Service`. Its actual semantics is beyond the scope of this document. `eb:Action` shall be unique within the `eb:Service` in which it is defined. The value of the `eb:Action` element is specified by the designer of the service.

An example of the `eb:Action` element follows:

```
<eb:Action>NewOrder</eb:Action>
```

If the value of either the `eb:Service` or `eb:Action` element is unrecognized by the receiving MSH, then it shall report a "ValueNotRecognized" error of severity "error".

When the value of this element is `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/test`, then the `eb:Service` element shall have the value `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service`. Such a value for the `eb:Action` element only indicates that the user message is sent for testing purposes and does not require any specific handling by the MSH.

8.2.3.11. eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:ConversationId

This element is a string identifying the set of related messages that make up a conversation between Parties.

If a CPA is referred to by `eb:AgreementRef`, the number of conversations related to this CPA shall comply with CPA requirements. The value of `eb:ConversationId` shall uniquely identify a conversation within the context of this CPA.

An example of the `eb:ConversationId` element follows:

```
<eb:ConversationId>20001209-133003-28572</eb:ConversationId>
```

The party initiating a conversation determines the value of the `eb:ConversationId` element that shall be reflected in all messages pertaining to that conversation. The actual semantics of this value is beyond the scope of this document. Implementations should provide a facility for mapping between their identification scheme and a `eb:ConversationId` generated by another implementation.

8.2.3.12. eb:Messaging/eb:UserMessage/eb:MessageProperties

This element has zero or more `eb:Property` child elements.

An `eb:Property` element is of xs:anySimpleType (e.g. string, URI) and has two attributes:

- `@name`: This attribute shall be present. Its use shall be agreed upon between partners.
- `@type`: This attribute may be present. It allows for resolution of conflicts between properties with the same name, and may also help with Property grouping, e.g. various elements of an address.

Its actual semantics is beyond the scope of this document. The element is intended to be consumed outside the ebMS-specified functions. It may contain some information that qualifies or abstracts message data, or that allows for binding the message to some business process. A representation in the header of such properties allows for more efficient monitoring, correlating, dispatching and validating functions (even if these are out of scope of ebMS specification) that do not require payload access.

Example:

```
<eb:MessageProperties>
  <eb:Property name="ContextId">C1234</eb:Property>
  <eb:Property name="processinstanceID">3A4-1234</eb:Property>
  <eb:Property name="transactionID">45764321</eb:Property>
</eb:MessageProperties>
```

8.2.3.13. eb:Messaging/eb:UserMessage/eb:PayloadInfo

Each `eb:PayloadInfo` element identifies payload data associated with the message. The purpose of the PayloadInfo is:

- to make it easier to extract particular payload parts associated with this ebMS Message;
- and to allow an application to determine whether it can process these payload parts, without having to parse them.

The `eb:PayloadInfo` element has the following child element:

- `eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo`
  This element occurs zero or more times. The PartInfo element is used to reference a MIME attachment, an XML element within the SOAP `Body`, or another resource which may be obtained by resolving a URL, according to the value of the `href` attribute, described in 8.2.3.14.

8.2.3.14. eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo

This element has the following attribute:

- `eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/@href`
  This option attribute has a value that is the Content-ID URI of the payload object referenced, an xml:id fragment identifier, or the URL of an externally referenced resource; for example, "cid:foo@example.com" or "#idref". Reference to Content-ID shall conform to RFC 2392. The absence of the attribute href in the element `eb:PartInfo` indicates that the payload part being referenced is the SOAP `Body` element itself. For example, a declaration of the following form simply indicates that the entire SOAP Body is to be considered a payload part in this ebMS message:

```
<eb:PayloadInfo>
  <eb:PartInfo/>
</eb:PayloadInfo>
```

Any other namespace-qualified attribute may be present. A receiving MSH may choose to ignore any foreign namespace attributes other than those defined above.

The designer of the business process or information exchange using ebXML Messaging decides what payload data is referenced by the Manifest and the values to be used for xlink:role.

This element has the following child elements:

- `eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:Schema`
  This element occurs zero or more times. It refers to schema(s) that define the instance document identified in the parent `eb:PartInfo` element. If the item being referenced has schema(s) of some kind that describe it (e.g. an XML Schema, DTD and/or a database schema), then the `eb:Schema` element should be present as a child of the `eb:PartInfo` element. It

provides a means of identifying the schema and its version defining the payload object identified by the parent `eb:PartInfo` element. This metadata may be used to validate the payload part to which it refers, but the MSH. The `eb:Schema` element contains the following attributes:

- (a) `namespace` - the optional target namespace of the schema
- (b) `location` – the required URI of the schema
- (c) `version` – an optional version identifier of the schema.

- `eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:PartProperties`
  This element has zero or more `eb:Property` child elements. An `eb:Property` element is of `xs:anySimpleType` (e.g. string, URI) and has a required `@name` attribute, the value of which shall be agreed between partners. Its actual semantics is beyond the scope of this document. The element is intended to be consumed outside the ebMS specified functions. It may contain metadata that qualifies or abstracts the payload data. A representation in the header of such properties allows for more efficient monitoring, correlating, dispatching and validating functions (even if these are out of scope of ebMS specification) that do not require payload access.

EXAMPLE 1, part properties:

```
<eb:PartProperties>
  <eb:Property name="Description">Purchase Order for 11 widgets</eb:Property>
  <eb:Property name="MimeType">application/xml</eb:Property>
</eb:PartProperties>
```

EXAMPLE 2: Full payload info example:

```
<eb:PayloadInfo>
  <eb:PartInfo href="cid:foo@example.com">
    <eb:Schema location="http://example.org/bar.xsd" version="2.0"/>
    <eb:PartProperties>
      <eb:Property name="Description">Purchase Order for 11 widgets</eb:Property>
      <eb:Property name="MimeType">application/xml</eb:Property>
    </eb:PartProperties>
  </eb:PartInfo>
  <eb:PartInfo href="#goo_payload_id">
    <eb:Schema location="http://example.org/bar.xsd" version="2.0"/>
    <eb:PartProperties>
      <eb:Property name="Description">Purchase Order for 100 widgets</eb:Property>
      <eb:Property name="MimeType">application/xml</eb:Property>
    </eb:PartProperties>
  </eb:PartInfo>
</eb:PayloadInfo>
```


8.2.4. eb:Messaging/eb:SignalMessage

8.2.4.1. General

This element is an alternative to the `eb:UserMessage` element. It has two child elements:

- `eb:Messaging/eb:SignalMessage/eb:MessageInfo`
  This element shall be present. It is similar to `eb:MessageInfo` as defined for user messages.

- `eb:Messaging/eb:SignalMessage/eb:[`*`SignalName`*`]`
  This element shall be present. It defines the nature of the ebMS signal. There is only one *`SignalName`* child element when [`SignalName`]=`PullRequest` or [`SignalName`]=`Receipt`. There may be several children elements when `SignalName=Error`.

An ebMS signal does not require any SOAP `Body`: if the SOAP `Body` is not empty, it shall be ignored by the MSH, as far as interpretation of the signal is concerned.

8.2.4.2. eb:Messaging/eb:SignalMessage/eb:PullRequest

This element has the following attribute:

- `eb:Messaging/eb:SignalMessage/eb:PullRequest/@mpc`
  This attribute may be present to identify the message partition channel from which the message is to be pulled. The absence of this attribute indicates the default MPC.

8.2.4.3. eb:Messaging/eb:SignalMessage/eb:Error

The `eb:Error` element may occur zero or more times. For its complete specification, refer to Clause 9.

8.2.4.4. eb:Messaging/eb:SignalMessage/eb:Receipt

The `eb:Receipt` element may occur zero or one times; and, if present, should contain a single `ebbpsig:NonRepudiationInformation` child element, as defined in the ebBP Signal Schema specified in the OASIS. ebXML Business Process TC, *ebXML Business Signals Schema*, 2006. The value of `eb:MessageInfo/eb:RefToMessageId` shall refer to the message for which this signal is a receipt.

8.2.5. Message unit bundling

When the `eb:Messaging` element contains multiple children elements, i.e. multiple message units (either user message units or signal message units), this is called message unit bundling. The following general rules govern message unit bundling:

- The `eb:Messaging` element shall not contain more than one `eb:UserMessage` element.
- The `eb:Messaging` element may contain multiple `eb:SignalMessage` elements, in addition to an optional `eb:UserMessage` element, but shall not contain more than one signal message unit of the same type.

The following is a non-exhaustive list of valid bundling cases:

(a) `eb:Messaging` element with the following children:

- an `eb:UserMessage` element
- an `eb:SignalMessage` element with an `eb:PullRequest` child

(b) `eb:Messaging` element with the following children:

- an `eb:UserMessage` element
- an `eb:SignalMessage` element with one or more `eb:Error` children

(c) `eb:Messaging` element with the following children:

- an `eb:UserMessage` element
- an `eb:SignalMessage` element with an `eb:PullRequest` child
- an `eb:SignalMessage` element (distinct from the previous one) with one or more `eb:Error` children

(d) `eb:Messaging` element with the following children:

- an `eb:SignalMessage` element with an `eb:PullRequest` child
- an `eb:SignalMessage` element (distinct from the previous one) with an `eb:Receipt` child

With regard to MEP transport channel bindings, the following restrictions shall be observed:

- An ebMS message containing an `eb:SignalMessage/eb:PullRequest` element cannot bind to the back-channel of the underlying transport protocol, regardless of its bundling context (bundling cases (a) , (c) or (d)) i.e. even if it is also a user message. For example, such a message can neither appear as "reply" in the “sync” transport channel binding, nor as a "one-way" in the Pull transport channel binding.
- An ebMS message containing an `eb:SignalMessage/eb:PullRequest` element and a user message unit (case (a) or case (c)) cannot act as a "request" in the “sync” transport channel binding, as the semantics of this combination would require sending back two user message units over the back-channel, which is a bundling case not supported in this release.

8.3. Examples of ebMS messages (informative)

8.3.1. General

The following listings provide examples of various types of ebMS messages: user message, pull request signal, error signal, receipt signal and a "bundled" message.

NOTE: The examples are depicted using the SOAP 1.1 envelope; however, SOAP 1.2 could have been used instead, with the appropriate namespace adjustment. In that case, the contents of the `eb:Messaging` header would be the same, with the exception of the attribute `eb:Messaging/@S11:mustUnderstand`, which becomes `eb:Messaging/@S12:mustUnderstand`, having a boolean value of "true" instead of the integer value "1".

8.3.2. UserMessage example

The following is an example of an ebMS request user message packaged in a SOAP 1.1 envelope:

EXAMPLE 1: ebMS request user message:

```
<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" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-200704_2019.xsd">
      <eb:UserMessage>
```

```
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-1@requester.example.com</eb:MessageId>
         </eb:MessageInfo>
         <eb:PartyInfo>
            <eb:From>
               <eb:PartyId>uri:requester.example.com</eb:PartyId>
               <eb:Role>http://example.org/roles/Buyer</eb:Role>
            </eb:From>
            <eb:To>
               <eb:PartyId type="someType">QRS543</eb:PartyId>
               <eb:Role>http://example.org/roles/Seller</eb:Role>
            </eb:To>
         </eb:PartyInfo>
         <eb:CollaborationInfo>
            <eb:AgreementRef>http://registry.example.com/cpa/123456</eb:AgreementRef>
            <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
            <eb:Action>NewPurchaseOrder</eb:Action>
            <eb:ConversationId>4321</eb:ConversationId>
         </eb:CollaborationInfo>
         <eb:MessageProperties>
            <eb:Property name="ProcessInst">PurchaseOrder:123456</eb:Property>
            <eb:Property name="ContextID"> 987654321</eb:Property>
         </eb:MessageProperties>
         <eb:PayloadInfo>
            <eb:PartInfo href="cid:part@example.com">
               <eb:Schema location="http://registry.example.org/po.xsd" version="2.0"/>
               <eb:PartProperties>
                  <eb:Property name="Description">Purchase Order for 11 Widgets</eb:Property>
                  <eb:Property name="MimeType">application/xml</eb:Property>
               </eb:PartProperties>
            </eb:PartInfo>
         </eb:PayloadInfo>
      </eb:UserMessage>
   </eb:Messaging>

</S11:Header>
<S11:Body>
</S11:Body>
</S11:Envelope>
```

The following is an example of a possible response to the user message in Example 1.

EXAMPLE 2: ebMS response user message:

```
<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" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-200704_2019.xsd">
      <eb:UserMessage>
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-2@responder.example.com</eb:MessageId>
            <eb:RefToMessageId>UUID-1@requester.example.com</eb:RefToMessageId>
         </eb:MessageInfo>
         <eb:PartyInfo>
            <eb:From>
               <eb:PartyId type="someType">QRS543</eb:PartyId>
               <eb:Role>http://example.org/roles/Seller</eb:Role>
            </eb:From>
            <eb:To>
               <eb:PartyId>uri:requester.example.com</eb:PartyId>
               <eb:Role>http://example.org/roles/Buyer</eb:Role>
            </eb:To>
         </eb:PartyInfo>
         <eb:CollaborationInfo>
            <eb:AgreementRef>http://registry.example.com/cpa/123456</eb:AgreementRef>
            <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
            <eb:Action>PurchaseOrderResponse</eb:Action>
            <eb:ConversationId>4321</eb:ConversationId>
         </eb:CollaborationInfo>
```

```
         <eb:PayloadInfo>
            <eb:PartInfo href="cid:part@example.com">
               <eb:Schema location="http://registry.example.org/poc.xsd" version="2.0"/>
               <eb:PartProperties>
                  <eb:Property name="Description">Purchase Order Confirmation</eb:Property>
                  <eb:Property name="MimeType">application/xml</eb:Property>
               </eb:PartProperties>
            </eb:PartInfo>
         </eb:PayloadInfo>
      </eb:UserMessage>
   </eb:Messaging>

</S11:Header>
<S11:Body>
</S11:Body>
</S11:Envelope>
```

8.3.3. PullRequest message example

The following is an example of a pull request signal message:

EXAMPLE: pull request signal message:

```
<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" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
             https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-200704_2019.xsd">
      <eb:SignalMessage>
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-2@initiator.example.com</eb:MessageId>
         </eb:MessageInfo>
         <eb:PullRequest mpc="http://msh.example.com/mpc123" />
      </eb:SignalMessage>
   </eb:Messaging>

</S11:Header>

<S11:Body/>
</S11:Envelope>
```

8.3.4. Error message example

The following is an example of an error signal message:

EXAMPLE: error signal message:

```
<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" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
             https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-200704_2019.xsd">
     <eb:SignalMessage>
        <eb:MessageInfo>
           <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
           <eb:MessageId>UUID-2@receiver.example.com</eb:MessageId>
        </eb:MessageInfo>
        <eb:Error origin="ebMS" category="Content"
                  errorCode="EBMS:0001" severity="failure"
                  refToMessageInErrorr="UUID-1@sender.example.com">
           <eb:Description xml:lang="en">Value not recognized</eb:Description>
        </eb:Error>
```

```
        <eb:Error origin="Security" category="Processing" errorCode="0101"
                  severity="failure" refToMessageInError="UUID-23@esender.fxample.com">
           <eb:Description xml:lang="en">Failed Authentication</eb:Description>
        </eb:Error>
     </eb:SignalMessage>

  </eb:Messaging>

</S11:Header>

<S11:Body/>
</S11:Envelope>
```

8.3.5. Receipt message example

The following is an example a receipt signal message, as described in 8.2.4.4.

EXAMPLE: receipt signal message:

```
<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 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
             https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-200704_2019.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:ebbpsig="http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0">

    <eb:SignalMessage>
      <eb:MessageInfo>
        <eb:Timestamp>2006-07-01T13:42:37.429Z</eb:Timestamp>
        <eb:MessageId>uiwtoruiopwr2543890@b.example.com</eb:MessageId>
        <eb:RefToMessageId>uiopfdsmnf4898965563434@a.example.com</eb:RefToMessageId>
      </eb:MessageInfo>

      <eb:Receipt>
        <ebbpsig:NonRepudiationInformation>
          <ebbpsig:MessagePartNRInformation>
            <ebbpsig:MessagePartIdentifier></ebbpsig:MessagePartIdentifier>
            <ds:Reference URI="http://b.example.com/doc45/#b">
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>fX/iNylcUHNLV41CE0eC7aEGP28=</ds:DigestValue>
            </ds:Reference>
          </ebbpsig:MessagePartNRInformation>
          <ebbpsig:MessagePartNRInformation>
            <ebbpsig:MessagePartIdentifier></ebbpsig:MessagePartIdentifier>
            <ds:Reference URI="http:/a.example.com/doc23/#a">
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>fX/iNylcUHNLV41CE0eC7aEGP28=</ds:DigestValue>
            </ds:Reference>
          </ebbpsig:MessagePartNRInformation>
        </ebbpsig:NonRepudiationInformation>
      </eb:Receipt>

    </eb:SignalMessage>

  </eb:Messaging>
</S11:Header>

<S11:Body/>
</S11:Envelope>
```

8.3.6. "Bundled" message example

The following is an example of a user message unit bundled with both `eb:PullRequest` and `eb:Error` signal message units, as described in 8.2.5:

```
<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" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
            https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-200704_2019.xsd">

     <eb:SignalMessage>
        <eb:MessageInfo>
           <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
           <eb:MessageId>UUID-2@receiver.example.com</eb:MessageId>
        </eb:MessageInfo>
        <eb:Error origin="ebMS" category="Content"
                  errorCode="EBMS:0001" severity="failure"
                  refToMessageInError="UUID-1@sender.example.com">
           <eb:Description xml:lang="en">Value not recognized</eb:Description>
        </eb:Error>
        <eb:Error origin="Security" category="Processing" errorCode="0101"
                  severity="failure" refToMessageInError="UUID-23@esender.fxample.com">
           <eb:Description xml:lang="en">Failed Authentication</eb:Description>
        </eb:Error>
     </eb:SignalMessage>

      <eb:SignalMessage>
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-2@initiator.example.com</eb:MessageId>
         </eb:MessageInfo>
         <eb:PullRequest mpc="http://msh.example.com/mpc123"/>
      </eb:SignalMessage>

      <eb:UserMessage>
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-1@requester.example.com</eb:MessageId>
         </eb:MessageInfo>
         <eb:PartyInfo>
            <eb:From>
               <eb:PartyId>uri:requester.example.com</eb:PartyId>
               <eb:Role>http://example.org/roles/Buyer</eb:Role>
            </eb:From>
            <eb:To>
               <eb:PartyId type="someType">QRS543</eb:PartyId>
               <eb:Role>http://example.org/roles/Seller</eb:Role>
            </eb:To>
         </eb:PartyInfo>
         <eb:CollaborationInfo>
            <eb:AgreementRef>http://registry.example.com/cpa/123456</eb:AgreementRef>
            <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
            <eb:Action>NewPurchaseOrder</eb:Action>
            <eb:ConversationId>4321</eb:ConversationId>
         </eb:CollaborationInfo>
         <eb:MessageProperties>
            <eb:Property name="ProcessInst">PurchaseOrder:123456</eb:Property>
            <eb:Property name="ContextID"> 987654321</eb:Property>
         </eb:MessageProperties>
         <eb:PayloadInfo>
            <eb:PartInfo href="cid:foo@example.com">
               <eb:Schema location="http://registry.example.org/bar.xsd" version="2.0"/>
               <eb:PartProperties>
                  <eb:Property name="Description">Purchase Order for 11 widgets</eb:Property>
                  <eb:Property name="MimeType">application/xml</eb:Property>
               </eb:PartProperties>
            </eb:PartInfo>
         </eb:PayloadInfo>
      </eb:UserMessage>

  </eb:Messaging>

</S11:Header>

<S11:Body/>
</S11:Envelope>
```

9. Error handling

9.1. General

Error handling shall take into account the composed nature of an MSH, which includes relatively independent (SOAP) modules such as those handling reliability and security. Error reporting is also subject to the same connectivity constraints as the exchange of regular messages. This calls for a more comprehensive error model. With regard to different ways to report errors, this model shall allow for a clear distinction between what is relevant to an agreement, and what is relevant to immutable interoperability requirements.

Error generation and error reporting are treated here as orthogonal concepts. While the generation of errors is a matter of conformance, the reporting of errors may be subject to an agreement. Consequently, the way errors are to be reported is specified in the P-Mode (P-Mode.ErrorHandling feature) that results from such an agreement.

9.2. Packaging of ebMS errors

9.2.1. eb:Error element

An ebMS error is represented by an `eb:Error` XML infoset, regardless of the way it is reported. Each error raised by an MSH has the following properties:

- `origin` (optional attribute)
- `category` (optional attribute)
- `errorCode` (required attribute)
- `severity` (required attribute)
- `refToMessageInError` (optional attribute)
- `shortDescription` (optional attribute)
- `Description` (optional element)
- `ErrorDetail` (optional element)

Example:

```
<eb:Error origin="ebMS" category="Unpackaging"
          shortDescription="InvalidHeader"
          errorCode="EBMS:0009" severity="fatal">
    <eb:Description xml:lang="en"> … </eb:Description>
</eb:Error>
```

9.2.2. eb:Error/@origin

This OPTIONAL attribute identifies the functional module within which the error occurred. This module could be the the ebMS Module, the Reliability Module, or the Security Module. Possible values for this attribute include "ebMS", "reliability", and "security".

9.2.3. eb:Error/@category

This OPTIONAL attribute identifies the type of error related to a particular origin. For example: `Content`, `Packaging`, `UnPackaging`, `Communication`, `InternalProcess`.

9.2.4. eb:Error/@errorCode

This attribute shall be present. Its value is a unique identifier for the type of error.

9.2.5. eb:Error/@severity

This attribute shall be present. Its value indicates the severity of the error. Valid values are: `warning`, `failure`.

The `warning` value indicates that a potentially disabling condition has been detected, but no message processing and/or exchange has failed so far. In particular, if the message was supposed to be delivered to a consumer, it would be delivered even though a warning was issued. Other related messages in the conversation or MEP can be generated and exchanged in spite of this problem.

The `failure` value indicates that the processing of a message did not proceed as expected, and cannot be considered successful. If, in spite of this, the message payload is in a state of being delivered, the default behavior is not to deliver it, unless an agreement states. This error does not presume the ability of the MSH to process other messages, although the conversation or the MEP instance this message was involved in is at risk of being invalid.

9.2.6. eb:Error/@refToMessageInError

This OPTIONAL attribute indicates the `eb:MessageId` of the message in error, for which this error is raised.

9.2.7. eb:Error/@shortDescription

This OPTIONAL attribute provides a short description of the error that can be reported in a log, in order to facilitate readability.

9.2.8. eb:Error/Description

This OPTIONAL element provides a narrative description of the error in the language defined by the `xml:lang` attribute. The content of this element is left to implementation-specific decisions.

9.2.9. eb:Error/ErrorDetail

This OPTIONAL element provides additional details about the context in which the error occurred. For example, it may be an exception trace.

9.3. ebMS Error message

When reported as messages, ebMS Errors are packaged as ebMS signal messages. Several `eb:Error` elements may be present under `eb:SignalMessage`. If this is the case, and if `eb:RefToMessageId` is present as a child of `eb:SignalMessage/eb:MessageInfo`, then every `eb:Error` element shall be related to the ebMS message (message-in-error) identified by `eb:RefToMessageId`.

If the element `eb:SignalMessage/eb:MessageInfo` does not contain `eb:RefToMessageId`, then the `eb:Error` element(s) shall not be related to a particular ebMS message.

For an example of an ebXML error message, see 8.3.4.

9.4. Extensibility of the eb:Error element

9.4.1. Adding new ebMS errors

The errorCode attribute (`eb:Messaging/eb:SignalMessage/eb:Error/@errorCode`) shall be an identifier that is unique within the scope of an MSH. ebMS Errors in addition to those specified here may be added by creating new errorCode values. The value of the errorCode attribute shall begin with the five characters "`EBMS:`".

9.5. Generating ebMS errors

This document identifies key ebMS Errors, as well as the conditions under which they shall be generated. Some of these error-raising conditions include the escalation as ebMS errors of either faults or errors generated by reliability and security modules. These modules could be those contained in the MSH raising the error, or those contained in a remote MSH communicating with the MSH raising the error. Except for some cases defined in this document, error escalation policies are left to an agreement between users, represented in the processing mode of an MSH (**P-Mode.ErrorHandling**).

9.6. Error reporting

There are three primary means of error reporting:

- Reporting with fault sending: An MSH may generate a SOAP `Fault` for reporting ebMS processing errors of severity "failure", which prevent further message processing. This `Fault` shall comply with SOAP `Fault` processing, i.e. be sent back as an HTTP response in case the message in error was over an HTTP request. In case of ebMS processing errors (see 9.7.2), the `Fault` message shall also include the `eb:SignalMessage/eb:Error` element in the `eb:Messaging` header.

- Reporting with notification: An out-of-band transfer of error information from MSH to some entity (message producer, consumer, or any other entity, be it local or remote). In case of notification to the message producer or consumer, such reporting action is abstracted by the *Notify* operation in the messaging model.

- Error message: an ebMS signal message sent from one MSH to another, which contains at least one `eb:Error` element. Such a reporting action is modeled by *Send* and *Receive* abstract operations over such a message. The reporting message shall always be combined with a SOAP `Fault` unless the severity is "warning".

Example 1: Different options in reporting errors raised on a sending MSH*: Some error detected on a submitted message and before it is even packaged, would normally be locally notified to the message producer, and not even reported to the destination MSH. However, in case this message was part of a larger exchange that is holding its state waiting for completion on the receiving side, the preferred policy could state that the message-in-error be also reported (using an error message) to the receiving MSH. If the receiving MSH is getting its messages as responses to pull request signals, such ebMS errors can be transmitted as responses to these signals. If user messages are pushed sender to receiver, it could be decided that errors generated on the sender side will be pushed like any regular message.*

Example 2: Different options in reporting errors raised on a receiving MSH*: If a receiving MSH detects an error in a received message, the reporting policy may vary depending on the context and the ability of parties to process such errors. For example, the error-raising receiving MSH may just notify its own consumer party, or send back an error message to the sending MSH, or both. The usual common requirement in all these cases, is that the error be reported somehow, and complies with the* `eb:Error` *element structure.*

Annex E shows possible options for combining error reporting with ebMS MEPs, when binding to a two-way protocol such as HTTP. It also shows how these combinations can be controlled with P-Mode parameters.

9.7. Standard ebMS errors

9.7.1. General

This subclause defines the standard error codes expected to be generated and processed by a conformant MSH. They are segmented according to the stage of processing they are likely to occur: during reliable message processing, security processing, and general ebMS processing.

9.7.2. ebMS processing errors

Table 2 describes the errors that may occur within the ebMS Module itself (ebMS errors that are not escalated errors), i.e. with `@origin="ebms"`. These errors shall be supported by an MSH, meaning generated appropriately, or understood by an MSH when reported to it.

Table 2: ebMS Processing Errors

Error Code	*Short Description*	*Recommended Severity*	*Category Value*	*Description or Semantics*
EBMS:0001	ValueNotRecognized	failure	Content	Although the message document is well formed and schema valid, some element/attribute contains a value that could not be recognized and therefore could not be used by the MSH.
EBMS:0002	FeatureNotSupported	warning	Content	Although the message document is well formed and schema valid, some element/attribute value cannot be processed as expected because the related feature is not supported by the MSH.
EBMS:0003	ValueInconsistent	failure	Content	Although the message document is well formed and schema valid, some element/attribute value is inconsistent either with the content of other element/attribute, or with the processing mode of the MSH, or with the normative requirements of the ebMS specification.
EBMS:0004	Other	failure	Content	
EBMS:0005	ConnectionFailure	failure	Communication	The MSH is experiencing temporary or permanent failure in trying to open a transport connection with a remote MSH.

Error Code	*Short Description*	*Recommended Severity*	*Category Value*	*Description or Semantics*
EBMS:0006	EmptyMessagePartitionChannel	warning	Communication	There is no message available for pulling from this MPC at this moment.
EBMS:0007	MimeInconsistency	failure	Unpackaging	The use of MIME is not consistent with the required usage in this document.
EBMS:0008	FeatureNotSupported	failure	Unpackaging	Although the message document is well formed and schema valid, the presence or absence of some element/attribute is not consistent with the capability of the MSH, with respect to supported features.
EBMS:0009	InvalidHeader	failure	Unpackaging	The ebMS header is either not well formed as an XML document, or does not conform to the ebMS packaging rules.
EBMS:0010	ProcessingModeMismatch	failure	Processing	The ebMS header or another header (e.g. reliability, security) expected by the MSH is not compatible with the expected content, based on the associated P-Mode.
EBMS:0011	ExternalPayloadError	failure	Content	The MSH is unable to resolve an external payload reference (i.e. a Part that is not contained within the ebMS Message, as identified by a PartInfo/href URI).

9.7.3. Security processing errors

Table 3 describes the Errors that originate within the security Module, i.e. with `@origin="security"`. These errors shall be escalated by an MSH, meaning generated appropriately, or understood by an MSH when reported to it.

Table 3: Security Processing Errors

Error Code	*Short Description*	*Recommended Severity*	*Category Value*	*Description or Semantics*
EBMS:0101	FailedAuthentication	Failure	Processing	The signature in the Security header intended for the "ebms" SOAP actor, could not be validated by the Security module.
EBMS:0102	FailedDecryption	Failure	Processing	The encrypted data reference the Security header intended for the "ebms" SOAP actor could not be decrypted by the Security Module.
EBMS:0103	PolicyNoncompliance	Failure	Processing	The processor determined that the message's security methods, parameters, scope or other security policy-level requirements or agreements were not satisfied.

9.7.4. Reliable messaging errors

Table 4 describes the Errors that originate within the reliable Messaging module, i.e. with `@origin="reliability"`. These errors shall be escalated by an MSH, meaning generated appropriately, or understood by an MSH when reported to it.

Table 4: Reliable Messaging Errors

Error Code	*Short Description*	*Recommended Severity*	*Category Value*	*Description or Semantics*
EBMS:0201	DysfunctionalReliability	failure	Processing	Some reliability function as implemented by the Reliability module, is not operational, or the reliability state associated with this message sequence is not valid.
EBMS:0202	DeliveryFailure	failure	Communication	Although the message was sent under Guaranteed delivery requirement, the Reliability module could not get assurance that the message was properly delivered, in spite of resending efforts.

10. Security module

10.1. General

The ebXML messaging service, by its very nature, presents certain security risks. A Messaging Service may be at risk by means of:

- Unauthorized access
- Data integrity and/or confidentiality attacks (e.g. through man-in-the-middle attacks)
- Denial-of-Service and spoofing

Each security risk is described in detail in the ebXML Technical Architecture Risk Assessment Technical Report.

Each of these security risks may be addressed in whole, or in part, by the application of one, or a combination of the countermeasures described in this clause. This document describes a set of profiles, or combinations of selected countermeasures, selected to address key risks based upon commonly available technologies. Each of the specified profiles includes a description of the risks that are not addressed.

Application of countermeasures should be balanced against an assessment of the inherent risks and the value of the asset(s) that might be placed at risk.

10.2. Security element

Web Services Security version 1.0 (WSS 1.0, specified in OASIS.*Web Services Security: SOAP Message Security 1.0*) or version 1.1 (WSS 1.1, specified in OASIS.*Web Services Security: SOAP Message Security 1.1*) may be utilized to secure an ebMS message. Web Services Security provides three mechanisms to secure messages: ability to send security tokens as part of a message, message integrity and message confidentiality.

Zero or one Security elements per target, belonging to the Web Services Security-defined namespace, may be present as a child of the SOAP `Header`. The `wsse:Security` element shall be namespace qualified in accordance with Web Services Security. The structure and content of the `wsse:Security` element shall conform to the OASIS Web Services Security 1.0 or 1.1 specifications and the W3C Web Services Security SOAP Messages with Attachments Profile.

To promote interoperability the security element shall conform to the WS-I Basic Security Profile Version 1.0, and WS-I Attachments Profile Version 1.0.

An MSH implementation can use WSS 1.0 and/or or WSS 1.1. Note that the security of attachment defined in WSS 1.1 is not only applicable to SOAP 1.1; security of attachment is orthogonal to the SOAP version, even though all examples in the WSS 1.1 specification depict only the SOAP 1.1 variant when securing attachments. In other words, an MSH can secure a SOAP 1.2 with Attachments message in the same way a SOAP 1.1 with Attachment can be secured in WSS 1.1. Refer to Annex C for complete details of the ebMS SOAP binding.

This document outlines the use of Web Services Security x.509 Certificate Token Profile, specified in OASIS.*Web Services Security UsernameToken Profile 1.0*, 2004 OASIS.*Web Services Security UsernameToken Profile 1.1*, 2006. and the Web Services Security Username Token Profile, specified in OASIS.*Web Services Security UsernameToken Profile 1.0*, 2004 or OASIS.*Web Services Security UsernameToken Profile 1.1*, 2006. An MSH implementation may choose to support other Web Services Security Profiles.

10.3. Signing messages

Signing of ebMS messages is defined in Web Services Security (WSS 1.0 and WSS 1.1) and shall conform to the OASIS *Web Services Security: SOAP Message Security 1.0*, 2004 and OASIS.*Web Services Security X.509 Certificate Token Profile 1.1*, 2006 specifications, if using WSS 1.0, or to the OASIS.*Web Services Security: SOAP Message Security 1.1*, 2005 and OASIS.*Web Services Security X.509 Certificate Token Profile 1.1*, 2006 specifications, if using WSS 1.1.

MSH implementations shall support detached signatures as defined by the W3C *XML-Signature Syntax and Processing* specification.

An MSH implementation may support enveloped signatures as defined by the W3C XML Signature Specification. Enveloped signatures add an additional level of security in detecting the addition of XML elements to the SOAP `Header`. The use of enveloped signatures may limit the ability of intermediaries to process messages.

To ensure the integrity of the user-specified payload data and ebMS message headers the entire `eb:Messaging` Container Element and the SOAP `Body` should be included in the signature.

10.4. Signing SOAP with attachments messages

Application payloads that are are built in conformance with the [SOAPATTACH] specification may be signed. To sign a SOAP with Attachment message the Security element shall be built in accordance with WSS 1.1.

Compliant MSH implementations shall support the Attachment-Content-Only transform. Compliant MSH implementations should support the Attachment-Complete transform.

To ensure the integrity of the user-specified payload data and ebMS headers the entire `eb:Messaging` Container Element, and all MIME Body parts of included payloads should be included in the signature.

10.5. Encrypting messages

Encryption of ebMS Messages is defined in Web Services Security (WSS 1.0 and WSS 1.1) and shall conform to the OASIS *Web Services Security: SOAP Message Security 1.0*, 2004 and OASIS.*Web Services Security X.509 Certificate Token Profile 1.1*, 2006 specifications, if using WSS 1.0, or to the OASIS.*Web Services Security: SOAP Message Security 1.1.* To encrypt ebMS Messages, the Web Services Security X.509 Certificate Token Profile shall be used.

An MSH Implementation may encrypt the `eb:Messaging` Container Element. It may also encrypt select child elements of `the eb:Messaging` header, leaving other elements unencrypted. For example, the `eb:PartyInfo` section may be used to aid in message routing before decryption of other elements has occurred. Therefore, when third-party routing of a message is expected, the `eb:PartyInfo` section should not be encrypted. To ensure the confidentiality of the user-specified payload data, it is the SOAP Body should be encrypted.

10.6. Encrypting SOAP with attachments messages

Application payloads that are built in conformance with the SOAP with attachments W3C Note specification may be encrypted. To encrypt a SOAP with Attachment message the `wsse:Security` element shall be built in accordance to WSS 1.1. To ensure the confidentiality of the user-specified payload data the MIME Body parts of included payloads should be encrypted.

10.7. Signing and encrypting messages

When both signature and encryption are required of the MSH, the message shall be signed prior to being encrypted.

10.8. Security token authentication

In constrained environments where management of XML digital signatures is not possible, an authentication alternative that is based on Web Services Security Username Token Profile should be supported, and may include support for `wsse:PasswordText`-type passwords. The use of the Username Token Profile shall conform to the OASIS.*Web Services Security UsernameToken Profile 1.0* specification, if using WSS 1.0, or to the OASIS.*Web Services Security UsernameToken Profile 1.1*, if using WWW 1.1.

The value of the `wsse:UserName` element is an implementation issue. The "user" may represent the MSH itself, or may represent a party using the MSH. In the latter case, there is no requirement that this user name be identical to some `eb:From/eb:PartyId` value.

An MSH may support other types of security tokens, as allowed by the WS-Security family of standards.

10.9. Security policy errors

A responding MSH may respond with an error if a received ebMS message does not meet the security policy of the responding MSH. For example, a security policy might indicate that messages with unsigned parts of the SOAP `Body` or `eb:Messaging` container element are unauthorized for further processing. If a responding MSH receives a message with unsigned data within the SOAP `Body` an error may be returned to the initiating MSH.

10.10. Secured message examples

10.10.1. Digitally signed and encrypted ebXML message

```
Mime-Version: 1.0
Content-Type: text/xml
Content-Transfer-Encoding: binary
SOAPAction: ""
Content-Length: 7205

<?xml version="1.0" encoding="UTF-8"?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
            https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-
200704_2019.xsd">
    <S11:Header xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/">
        <eb:Messaging id="ebMessage" S11:mustUnderstand="1">
            <eb:UserMessage>
                <eb:MessageInfo>
                    <eb:Timestamp>2006-10-31T17:36:20.656Z</eb:Timestamp>
                    <eb:MessageId>UUID-2@msh-server.example.com</eb:MessageId>
                    <eb:RefToMessageId>UUID-1@msh-
server.example.com</eb:RefToMessageId>
                </eb:MessageInfo>
                <eb:PartyInfo>
                    <eb:From>
                        <eb:PartyId>uri:msh-server.example.com</eb:PartyId>
                        <eb:Role>http://example.org/roles/Buyer</eb:Role>
                    </eb:From>
                    <eb:To>
                        <eb:PartyId type="someType">QRS543</eb:PartyId>
                        <eb:Role>http://example.org/roles/Seller</eb:Role>
                    </eb:To>
                </eb:PartyInfo>
                <eb:CollaborationInfo>
                    <eb:AgreementRef>http://msh-
server.example.com/cpa/123456</eb:AgreementRef>
                    <eb:Service type="someType">QuoteToCollect</eb:Service>
                    <eb:Action>NewPurchaseOrder</eb:Action>
                    <eb:ConversationId>2a81ffbd-0d3d-4cbd-8601-
d916e0ed2fe2</eb:ConversationId>
                </eb:CollaborationInfo>
                <eb:MessageProperties>
                    <eb:Property name="ProcessInst">PurchaseOrder:123456</eb:Property>
                    <eb:Property name="ContextID">987654321</eb:Property>
                </eb:MessageProperties>
                <eb:PayloadInfo>
                    <eb:PartInfo href="#enc"/>
                </eb:PayloadInfo>
            </eb:UserMessage>
        </eb:Messaging>
        <wsse:Security S11:mustUnderstand="1"
            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">
            <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3"
                wsu:Id="signingCert">...</wsse:BinarySecurityToken>
            <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3"
                wsu:Id="encryptionCert">...</wsse:BinarySecurityToken>
            <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
                <enc:EncryptionMethod  Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
1_5"
```

```
                    xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"/>
                <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#encryptionCert"/>
                    </wsse:SecurityTokenReference>
                </KeyInfo>
                <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">

<CipherValue>F3HmZ2Ldyn0umLCx/8Q9B9e8OoslJx9i9hOWQjh6JJwYqDLbdg0QVFiVT1LVjazlThS9m9rkR
tpkhCUIY1xjFKtDsuIIAW8cLZv7IHkVoDtQ7ihJc8hYIlEESX9qZN65JgyAa3BYgW9ipjGHtNgZ9RzUdzKdeY7
4DFm27R6m8b0=</CipherValue>
                </CipherData>
                <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
                    <DataReference URI="#enc"/>
                </ReferenceList>
            </enc:EncryptedKey>
            <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <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="#ebMessage">
                        <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>Ae0PLUKJUnUyAMXkLQD/WwKiFiI=</ds:DigestValue>
                    </ds:Reference>
                    <ds:Reference URI="#body">
                        <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>kNY6X7LnRTwxXXBzSw07tcA0KSU=</ds:DigestValue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>

T24okA0MUh5iBNMG6tk8QAKZ+lFMmY1rcPnkOr9j3fHRGM2qqUnoBydOTnClcEMzPZbnlhdN

YZYmab1lqa4N5ynLjwlM4kpQuMip9hapijwL67aBnUeHiFmUau0x9DBOdKZTVa1QQ92106ge
                    j2YPDt3VKIlLLT2c8O4TfayGvuY= </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference
                        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
                        <wsse:Reference URI="#signingCert"/>
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </S11:Header>
    <S11:Body wsu:Id="body"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
        <EncryptedData Id="enc" Type="http://www.w3.org/2001/04/xmlenc#Content"
            xmlns="http://www.w3.org/2001/04/xmlenc#">
            <EncryptionMethod    Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/>
            <CipherData>

<CipherValue>tjOgUPMmQwd6hXiHuvl42swqv4dTYiBfmg8u1SuFVRC3yfNlokshvoxs1/qQoqN1prDiSOxsx
sFvg1la7dehjMWb0owuvU2de1eKr5KPcSApnG+kTvNrtg==</CipherValue>
            </CipherData>
        </EncryptedData>
    </S11:Body>
</S11:Envelope>
```

10.10.2. Digitally signed and encrypted ebXML SOAP with attachments message

```
Mime-Version: 1.0
Content-Type: multipart/related; type="text/xml";
    boundary="----=_Part_2_6825397.1130520599536"
SOAPAction: ""
Content-Length: 7860

------=_Part_2_6825397.1130520599536
Content-Type: text/xml
Content-Transfer-Encoding: binary

<?xml version="1.0" encoding="UTF-8"?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
             https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-
200704_2019.xsd">
    <S11:Header xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/">
        <eb:Messaging id="ebMessage" S11:mustUnderstand="1">
            <eb:UserMessage>
                <eb:MessageInfo>
                    <eb:Timestamp>2006-10-28T17:29:59.119Z</eb:Timestamp>
                    <eb:MessageId>UUID-2@msh-server.example.com</eb:MessageId>
                    <eb:RefToMessageId>UUID-1@msh-
server.example.com</eb:RefToMessageId>
                </eb:MessageInfo>
                <eb:PartyInfo>
                    <eb:From>
                        <eb:PartyId>uri:msh-server.example.com</eb:PartyId>
                        <eb:Role>http://example.org/roles/Buyer</eb:Role>
                    </eb:From>
                    <eb:To>
                        <eb:PartyId type="someType">QRS543</eb:PartyId>
                        <eb:Role>http://example.org/roles/Seller</eb:Role>
                    </eb:To>
                </eb:PartyInfo>
                <eb:CollaborationInfo>
                    <eb:AgreementRef>http://msh-
server.example.com/cpa/123456</eb:AgreementRef>
                    <eb:Service type="someType">QuoteToCollect</eb:Service>
                    <eb:Action>NewPurchaseOrder</eb:Action>
                    <eb:ConversationId>782a5c5a-9dad-4cd9-9bbe-
94c0d737f22b</eb:ConversationId>
                </eb:CollaborationInfo>
                <eb:MessageProperties>
                    <eb:Property name="ProcessInst">PurchaseOrder:123456</eb:Property>
                    <eb:Property name="ContextID">987654321</eb:Property>
                </eb:MessageProperties>
                <eb:PayloadInfo>
                    <eb:PartInfo href="cid:PO_Image@example.com" />
                </eb:PayloadInfo>
            </eb:UserMessage>
        </eb:Messaging>
        <wsse:Security S11:mustUnderstand="1"
            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">
            <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3"
                wsu:Id="signingCert">...</wsse:BinarySecurityToken>
            <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3"
                wsu:Id="encryptionCert">...</wsse:BinarySecurityToken>
            <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
                <enc:EncryptionMethod  Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
1_5"
```

```
                    xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"/>
                <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#encryptionCert"/>
                    </wsse:SecurityTokenReference>
                </KeyInfo>
                <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">

<CipherValue>jJRbQBjzYpfdCkPk5F7jUoFjw6Ls6DQ8D9sdI62fwjW9Um/g9QfivLeVzvSndgnthfEBC1Z6l
oKiuEF5/Ztw/tFrRgkboR7EBG5XaJUnt0rt8iCChy4PfxCEhH1KjFgTJhUbXxNW3FxSLkouCn2qIBDrJqwZXAI
istt29JrANCc=</CipherValue>
                </CipherData>
                <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
                    <DataReference URI="#encrypted-attachment"/>
                </ReferenceList>
            </enc:EncryptedKey>
            <EncryptedData Id="encrypted-attachment" MimeType="image/jpeg"
                Type="http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-
1.1#Attachment-Content-Only"
                xmlns="http://www.w3.org/2001/04/xmlenc#">
                <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
                <CipherData>
                    <CipherReference URI="cid:PO_Image@example.com">
                        <Transforms>
                            <Transform
                                Algorithm="http://docs.oasis-open.org/wss/oasis-wss-
SwAProfile-1.1#Attachment-Ciphertext-Transform"
                                xmlns="http://www.w3.org/2000/09/xmldsig#"/>
                        </Transforms>
                    </CipherReference>
                </CipherData>
            </EncryptedData>
            <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <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="#ebMessage">
                        <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>xUISuIg5eVxy3FL/4yCrZoEZrTM=</ds:DigestValue>
                    </ds:Reference>
                    <ds:Reference URI="cid:PO_Image@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/xmldsig#sha1"/>
                        <ds:DigestValue>R4hCV4K4I5QZdSsrP4KrLu46hFo=</ds:DigestValue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>

BGnJV/b7EUbAEsn7GmNhZ8yYN6ZoO6uz29E5r9GHxDW+MUH4wksgA654w+sB0rWl8xNranag

3dhKoHbaRERzYHDGq1VfIRqgEwOrHwhz4h7uoLX4yxOU6G9T/gily67Q3pENGpmVowzoppHm
                    /yd/A2T0+v4vso20aJiSieEIzSQ= </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference
                        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
                        <wsse:Reference URI="#signingCert"/>
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </S11:Header>
```

```
    <S11:Body/>
</S11:Envelope>

------=_Part_2_6825397.1130520599536
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
Content-Id: <PO_Image@example.com>
Content-Description: WSS XML Encryption message; type="image/jpeg"

VEhmwb4FHFhqQH8m5PKqVu8H0/bq2yUF

------=_Part_2_6825397.1130520599536--
```

10.10.3. Digitally signed receipt signal message

The following is an example of a signed Receipt for the user message shown in 10.10.1. Note the correlations to that message in the `eb:RefToMessageId` and `ds:Reference` elements.

```
Mime-Version: 1.0
Content-Type: text/xml
Content-Transfer-Encoding: binary
SOAPAction: ""
Content-Length: 7205

<?xml version="1.0" encoding="UTF-8"?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
        https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-
200704_2019.xsd">
  <S11:Header xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/">
    <eb:Messaging id="ThisebMessage" S11:mustUnderstand="1">

      <eb:SignalMessage>
        <eb:MessageInfo>
          <eb:Timestamp>2006-10-31T18:02:37.429Z</eb:Timestamp>
          <eb:MessageId>UUID-3@msh-server.example.com</eb:MessageId>
     <eb:RefToMessageId>UUID-2@msh-server.example.com</eb:RefToMessageId>
        </eb:MessageInfo>

        <eb:Receipt>
          <ebbpsig:NonRepudiationInformation
   xmlns:ebbpsig="http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ebbpsig:MessagePartNRInformation>
              <ebbpsig:MessagePartIdentifier>ebMessage</ebbpsig:MessagePartIdentifier>
              <ds:Reference URI="#ebMessage">
                <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>Ae0PLUKJUnUyAMXkLQD/WwKiFiI=</ds:DigestValue>
              </ds:Reference>
            </ebbpsig:MessagePartNRInformation>
            <ebbpsig:MessagePartNRInformation>
              <ebbpsig:MessagePartIdentifier>body</ebbpsig:MessagePartIdentifier>
              <ds:Reference URI="#body">
                <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>kNY6X7LnRTwxXXBzSw07tcA0KSU=</ds:DigestValue>
              </ds:Reference>
            </ebbpsig:MessagePartNRInformation>
          </ebbpsig:NonRepudiationInformation>
        </eb:Receipt>
      </eb:SignalMessage>

    </eb:Messaging>

    <wsse:Security S11:mustUnderstand="1"
```

```
      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">
      <wsse:BinarySecurityToken
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
message-security-1.0#Base64Binary"
        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
profile-1.0#X509v3"
        wsu:Id="signingCert">...</wsse:BinarySecurityToken>

      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <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="#ThisebMessage">
            <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>Ae0PLUKJUnUyAMXkLQD/WwKiFiI=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>T24okA0MUh5iBNMG6tk8QAKZ+lFMmY1rcPnkOr9j3fHRGM2qqUnoB
                           ydOTnC1cEMzPZbnlhdNYZYmab1lqa4N5ynLJylM4kp0uMip9hapij
                           wL67aBnUeHiFmUau0x9DBOdKZTVa1QQ9210ogej2YPDt3VKIlLLT2
                           c8O4TfayGvuY= </ds:SignatureValue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
            <wsse:Reference URI="#signingCert"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </S11:Header>
  <S11:Body/>
</S11:Envelope>
```

10.11. Message authorization

Message authorization is defined here as authorizing the processing of a message in conformance with the parameters of the P-Mode associated with this message. This includes authorizing the access to some ebMS resources such as:

- "delivery" resources as identified by `eb:Service` and `eb:Action`
- Message partition channel (MPC) that a pull signal message is accessing for pulling messages.

This is different from simply authorizing a received message for further processing by the MSH, which can be achieved by processing the wsse:`Security` header described in Clause 10, regardless of ebMS-specific resources claimed by the message. A message could successfully be authenticated by the security module (see 7.2), yet not be authorized to pull from a particular MPC, or to effect delivery of data to a particular Service. For implementations in which there is limited interaction between processing modules of the MSH - e.g. in case of an architecture based on composing SOAP nodes, the Security header may be consumed by the WSS module before reaching the ebMS message processor. (Even if the header is forwarded, it may be impractical to require an ebMS processor implementation to parse it.)

This document provides a resource-level authorization mechanism. Since any resource a message may claim access to is identified by the P-Mode associated with the message, this is equivalent to authorizing the association of the message with the P-Mode.

For this purpose, a second `wsse:Security` header, which contains only an authentication token, may be present. This document describes in particular one token option, not exclusively of others: the `wsse:UsernameToken` profile. This secondary `wsse:Security` header may itself be secured (e.g. encrypted) by the main security header.

In the P-Mode model (see Annex D) such tokens are represented as the **PMode.Initiator.Authorization** parameter set (for authorizing the initiator of an MEP) and the **PMode.Responder.Authorization** parameter set.

This header is not intended to be processed or consumed by the same WSS module as the "main" Security header, but is targeted further along to the "ebms" actor - typically a role played by the ebMS header processor, which has knowledge of the association between these tokens and the P-Modes that govern the message processing.

The following example shows a pull request message for which this type of authorization is required. Both security headers (shown here as a SOAP1.1 message) are present, with one of them - the secondary header - targeted to the "ebms" actor. This pull signal can effect message delivery from MPC "`http://msh.example.com/mpc123`" only if its credentials match the authorization parameters of at least one P-Mode associated with pulling messages on this MPC.

```
<?xml version="1.0" encoding="UTF-8"?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/">

  <S11:Header
     xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">

    <eb:Messaging S11:mustUnderstand="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
         https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-
200704_2019.xsd">
      <eb:SignalMessage>
        <eb:MessageInfo>
          <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
          <eb:MessageId>UUID-2@initiator.example.com</eb:MessageId>
        </eb:MessageInfo>
        <eb:PullRequest mpc="http://msh.example.com/mpc123" />
      </eb:SignalMessage>
    </eb:Messaging>

    <wsse:Security S11:mustUnderstand="1">
      <!-- main security header -->
    </wsse:Security>

    <wsse:Security S11:mustUnderstand="1" actor="ebms"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
      <!-- authorization security header (here non encrypted) -->
      <wsse:UsernameToken wsu:Id="ebms-1234">
        <wsse:Username>acme</wsse:Username>
        <wsse:Password Type="...">xyz123</wsse:Password>
        <wsu:Created> ... </wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>

  </S11:Header>
  <S11:Body />
</S11:Envelope>
```

Permission to use a P-Mode for processing a received message is granted or denied at the time the P-Mode authorization parameters are compared with the credentials in the message.

10.12. Securing the PullRequest signal

10.12.1. Authentication

A sending MSH shall be able to authenticate a receiving MSH that sends a pull request. When authentication is required for a particular receiving MSH, the sending MSH should use security at the SOAP protocol level (WSS). In case a receiving MSH is not able to use SOAP level security, other authentication mechanisms may be used, e.g. the HTTP Basic or Digest Access Authentication schemes conform RFC 2617.

10.12.2. Authorization

The processing of a pull request signal received by a sending MSH may be authorized based on any of the following, or combination of the following, mechanisms:

(a) Digital signature validation by the Security (WSS) module (see 10.3 and 10.4),

A WSS authentication token addressed to the "default" actor/role (see 10.8).

A WSS authentication token addressed to the "ebms" actor/role (see 10.11).

(b) A transfer-protocol-level identity-authentication mechanism, such as those described in 10.12.1.

10.12.3. Preventing replay attacks

Malignant duplication and reuse of a pull request signals could lead to transfer of user messages to an unauthorized destination in spite of valid claims in the signal message. In order to prevent this attack, (1) At-Most-Once reliability should be used so that duplicate elimination would eliminate pull request duplicates, (2) and the integrity of reliability headers should be enforced by proper compliance with WSS.

10.13. Countermeasure technologies

10.13.1. Persistent digital signature

The only available technology that can be applied to the purpose of digitally signing an ebMS Message (the ebXML SOAP `Header` and `Body` and its associated payload objects) is provided by technology that conforms to the Web Services Security and Web Services Security SOAP Messages with Attachments Profile. An XML Signature conforming to these specifications can selectively sign portions of an XML document(s), permitting the documents to be augmented (new element content added) while preserving the validity of the signature(s).

If signatures are being used to digitally sign an ebMS Message then Web Services Security and Web Services Security SOAP Messages with Attachments Profile shall be used to bind the ebXML SOAP `Header` and `Body` to the ebXML payload container(s) or data elsewhere on the web that relate to the message.

An ebMS Message requiring a digital signature shall be signed following the process defined in this subclause and shall be in full compliance with Web Services Security and Web Services Security SOAP Messages with Attachments Profile.

10.13.2. Persistent signed receipt

An ebMS Message that has been digitally signed may be acknowledged with a message containing an `eb:Receipt` Signal (described in 8.2.4.4), that itself is digitally signed in the manner described in 10.13.1. The receipt signal shall contain the information necessary to provide nonrepudiation of receipt of the original message; that is, an XML digital signature `ds:Reference` element list consistent with that contained in the Web Services Security Signature element of the original message.

10.13.3. Non-persistent authentication

Non-persistent authentication is provided by the communications channel used to transport the ebMS message. This authentication may be either in one direction or bi-directional. The specific method will be determined by the communications protocol used. For instance, the use of a secure network protocol, such as TLS (IETF RFC 2246) or IPSec (IETF RFC 2402) provides the sender of an ebMS message with a way to authenticate the destination for the TCP/IP environment.

10.13.4. Non-persistent integrity

A secure network protocol such as TLS or IPSec may be configured to provide for digests and comparisons of the packets transmitted via the network connection.

10.13.5. Persistent confidentiality

Persistent confidentiality is provided by technology that conforms to Web Services Security and Web Services Security SOAP Messages with Attachments Profile. Encryption conforming to these specifications can provide persistent, selective confidentiality of elements within an ebMS Message including the SOAP Header.

10.13.6. Non-persistent confidentiality

A secure network protocol, such as TLS or IPSEC, provides transient confidentiality of a message as it is transferred between two ebXML adjacent MSH nodes.

10.13.7. Persistent authorization

Persistent authorization may be provided using Web Services Security: SAML Token Profile.

10.13.8. Non-persistent authorization

A secure network protocol such as TLS or IPSEC may be configured to provide for bilateral authentication of certificates prior to establishing a session. This provides for the ability for an ebXML MSH to authenticate the source of a connection and to recognize the source as an authorized source of ebMS Messages.

10.14. Security considerations

Implementers should take note, there is a vulnerability present even when Web Services Security is used to protect to protect the integrity and origin of ebMS Messages. The significance of the vulnerability necessarily depends on the deployed environment and the transport used to exchange ebMS Messages.

The vulnerability is present because ebXML messaging is an integration of both XML and MIME technologies. Whenever two or more technologies are conjoined there are always additional (sometimes unique) security issues to be addressed. In this case, MIME is used as the framework for the message package, containing the SOAP `Envelope` and any payload containers. Various elements of the SOAP `Envelope` make reference to the payloads, identified via MIME mechanisms. In addition, various labels are duplicated in both the SOAP `Envelope` and the MIME framework, for example, the type of the content in the payload. The issue is how and when all of this information is used.

Specifically, the MIME Content-ID: header is used to specify a unique, identifying label for each payload. The label is used in the SOAP Envelope to identify the payload whenever it is needed. The MIME Content-Type: header is used to identify the type of content carried in the payload; some content types may contain additional parameters serving to further qualify the actual type. This information is available in the SOAP Envelope.

The MIME headers are not protected; not even when a Web Services Security based digital signature and/or Web Services Security based encryption is applied. Thus, an ebMS Message may be at risk depending on how the information in the MIME headers is processed as compared to the information in the SOAP Envelope.

The Content-ID: MIME header is critical. An adversary could easily mount a denial-of-service attack by mixing and matching payloads with the Content-ID: headers. As with most denial-of-service attacks, no specific protection is offered for this vulnerability. However, it should be detected since the digest calculated for the actual payload will not match the digest included in the SOAP Envelope when the digital signature is validated.

The presence of the content type in both the MIME headers and SOAP Envelope is a problem. Ordinary security practices discourage duplicating information in two places. When information is duplicated, ordinary security practices require the information in both places to be compared to ensure they are equal. It would be considered a security violation if both sets of information fail to match.

An adversary could change the MIME headers while a message is en route from its origin to its destination and this would not be detected when the security services are validated. This threat is less significant in a peer-to-peer transport environment as compared to a multi-hop transport environment. All implementations are at risk if the ebMS Message is ever recorded in a long-term storage area since a compromise of that area puts the message at risk for modification.

The actual risk depends on how an implementation uses each of the duplicate sets of information. If any processing beyond the MIME parsing for body part identification and separation is dependent on the information in the MIME headers, then the implementation is at risk of being directed to take unintended or undesirable actions. How this might be exploited is best compared to the common programming mistake of permitting buffer overflows: it depends on the creativity and persistence of the adversary.

Thus, an implementation could reduce the risk by ensuring that the unprotected information in the MIME headers is never used except by the MIME parser for the minimum purpose of identifying and separating the body parts. This version of the specification makes no recommendation regarding whether or not an implementation should compare the duplicate sets of information nor what action to take based on the results of the comparison.

11. Reliable messaging module

11.1. The reliable messaging model

11.1.1. General

The reliable delivery of messages has two aspects:

1. a contractual aspect regarding delivery conditions and error notification, where the contracting parties are the MSHs and the entities using the MSH - the message producer and consumer.

2. a protocol aspect, that describes the reliability mechanism "on the wire".

This clause emphasizes the contractual aspect. The details of the protocol aspect depend on the specifics of the reliability module and its binding, described in Annex B.

11.1.2. Message processing

A basic design principle in ebMS 3.0 is to modularize major messaging QoS features, meaning no interference – except of black-box style – with other aspects of message processing, so that (a) the MSH can rely on existing standards in the area of concern, but also (b) so that implementations of such standards can be reused with no or little modification.

The reliability function is processed separately from the ebMS header. This processing will be abstractly defined as performed by a module possibly acting as a separate SOAP node, called a **reliable messaging processor (RMP).** The reliability of ebMS Messages is supported by SOAP header extensions – called here "reliability header(s)" – that are distinct from ebMS headers.

The following serialization is shall be applied between reliability headers and ebMS-qualified headers:

On sending side:

1. processing of ebMS headers (the ebMS-qualified headers are added to the message).

2. processing of reliability headers (the headers are added to the message).

On receiving side:

1. processing of reliability headers (the headers are removed from the message).

2. processing of ebMS headers (the ebMS-qualified headers are removed from the message).

NOTE: Other steps in the processing of ebXML headers, such as security headers, are not mentioned here. The above message processing flows do not exclude the insertion of such additional steps, which are depicted in Figure 7 and described in 7.2.

11.1.3. The reliable messaging processor in the MSH

As illustrated in Figure 10 and Figure 11, the reliability model requires two instances of RMP playing different roles when executing a reliable MEP: the initiator RMP (associated with the initiator MSH) and the responder RMP (associated with the responder MSH). It should be noted that these roles do not change over the execution of a simple ebMS MEP instance, as opposed to the roles of sending and receiving, which may vary for each user message exchanged. This means, for example, that the Initiator

will assume the necessary functions to send a request message reliably, and also receive its response, if any (successively taking on a sending and then receiving role, as defined in the messaging model in 5.1.1).

Five abstract operations, RM-Submit, RM-Deliver, RM-SubmitResponse, RM-DeliverResponse, RM-Notify, represent the abstract interface of the RMP. They transfer either message data or notification data between an RMP and another component of the MSH. This other component is normally the module that is processing the ebMS header and its packaging, as described in the processing model (7.2). On the sender side, this module is abstracted as the RM-Producer. On the receiver side, it is abstracted as the RM-Consumer. In this subclause, the expression "sent reliably" means that the sending is subject to a reliability contract (see 11.2.2).

The abstract RM operations are defined as follows:

- **RM-Submit**
  An abstract operation that transfers a SOAP message from an RM-Producer to an Initiator RMP, so that this message can be sent reliably.

- **RM-Deliver**
  An abstract operation that transfers a SOAP message from a Responder RMP to its RM-Consumer, so that the payload from this message can later be delivered by the MSH.

- **RM-SubmitResponse**
  An abstract operation that transfers a SOAP message from an RM-Producer to a Responder RMP as a response to a message received reliably. This response is sent back reliably.

- **RM-DeliverResponse**
  An abstract operation that transfers a received SOAP response message from an Initiator RMP to its RM-Consumer.

- **RM-Notify**
  An abstract operation that makes available to the RM-Producer or to the RM-Consumer a failure status of a message sent reliably (e.g. a notification telling that the message was not delivered).

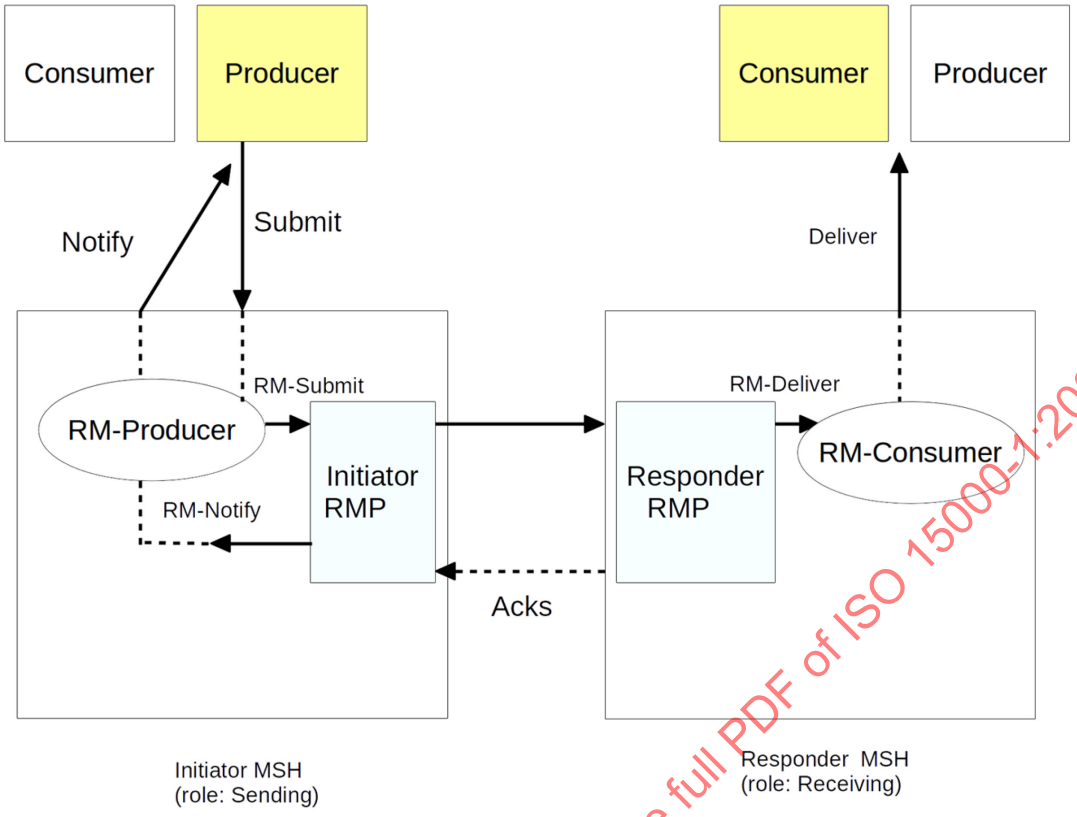

Figure 10: Sending an ebMS Message Reliably

Figure 10 shows the operations involved when sending a request reliably. As indicated in 11.3, this sequence of operations applies either to the user message in the One-Way/Push MEP, the pull request Signal of a One-Way/Pull MEP, or the first leg of a Two-Way/Sync MEP.

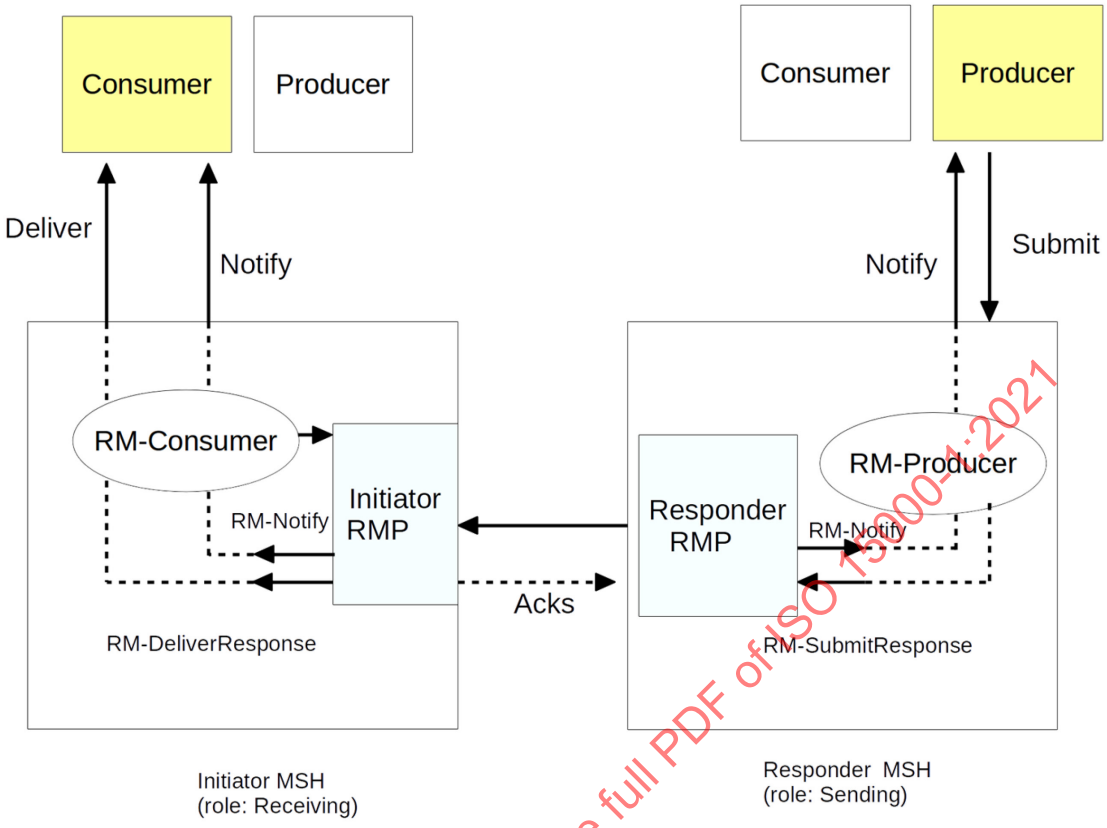

Figure 11: Sending an ebMS MEP Response Message Reliably

Figure 11 shows the abstract operations and components involved when sending a response reliably. As indicated in 11.3, this sequence of operations applies ither to a pulled user message in the One-Way/Pull MEP or the response user message in a Two-Way/Sync MEP. Note that depending on the reliability processing mode (P-Mode.Reliability), awareness of delivery failure may occur on either side.

11.2. Reliable delivery of ebMS messages

11.2.1. General

Because the reliability function is supported by a module (RMP) within the MSH, the contractual aspect has to be considered at two levels: (a) between the RMP and the MSH internals, and (b) between the MSH and its consumer/producer entities (e.g. an application).

11.2.2. Reliability contracts for the RMP

Depending on the reliability required for a request message, an RMP shall support some or all of the following contracts:

- **At-Least-Once RM-Delivery**
  When sending a message with this reliability requirement (RM-Submit invocation), one of the two following outcomes shall occur: either (1) the Responder RMP successfully delivers (RM-Deliver operation) the message to the RM-Consumer or (2) either the Initiator RMP or the Responder RMP notifies (RM-Notify operation) respectively the RM producer or the RM consumer of a delivery failure.

- **At-Most-Once RM-Delivery**
  Under this reliability requirement, a message submitted by an RM producer (RM-Submit operation) to an Initiator RMP shall not be delivered more than once by the Responder RMP to its RM-Consumer. The notion of message duplicate is based on a notion of message ID that shall be supported by the reliability specification being used.

- **In-Order RM-Delivery**
  Under this reliability requirement, a sequence of messages submitted to an Initiator RMP (sequence of RM-Submit invocations) shall be delivered in the same order by the Responder RMP to its RM-Consumer.

These contracts may also apply to response messages, as illustrated in Figure 11. In such a case they are expressed in the above contracts with RM-SubmitResponse and RM-DeliverResponse operations (instead of RM-Submit and RM-Deliver, respectively), and the Responder and Initiator RMPs switch roles.

These contracts may be combined; e.g. Exactly-Once results from the combination of At-Least-Once and At-Most-Once.

In order to support these reliability contracts, both Initiator and Responder RMPs shall use a reliability protocol independent from the transport protocol and that provides end-to-end acknowledgment and message resending capabilities. The details and parameters associated with these protocol functions are described in Annex B.

11.2.3. Reliability contracts for the MSH

Because reliability quality of service (QoS) shall have significance for the user of the MSH (producer, consumer), and not just for the internal components of the MSH (called RM-Producer and RM-Consumer) that interact with the RMP component, it is necessary to extend the above contracts and express them in terms of abstract MSH operations:

- **At-Least-Once ebMS Delivery**
  When sending a message with this reliability requirement (*Submit* invocation), one of the two following outcomes shall occur: either (1) the Responder MSH successfully delivers (*Deliver* operation) the message to the consumer or (2) a delivery failure notification is communicated (*Notify* operation) to either the producer or the consumer.

- **At-Most-Once ebMS Delivery**:
  Under this reliability requirement, a message transmitted as the result of a *Submit* invocation on the initiator MSH shall not be delivered more than once by the responder MSH to its consumer. An ebMS message is a duplicate of another if it has same `eb:MessageId` value.

- **In-Order ebMS Delivery**
  Under this reliability requirement, a sequence of messages submitted to the initiator MSH by its producer shall be delivered by the responder MSH in the same order to its consumer.

In order to fulfill the above QoS requirements, an MSH shall do the following, in addition to interfacing with the reliability functions provided by the RMP:

- Ensure a proper mapping between MSH abstract operations and RMP abstract operations. This mapping, which depends on the ebMS MEP being used, is described in 11.3.

- Ensure the handling of additional failure cases that may happen outside the RMP processing and outside the transport layer. For example, in the case of At-Least-Once delivery, the sending MSH shall ensure that if a message that has been submitted (*Submit*) fails before RM-Submit is

invoked, then a delivery failure Error is generated, as would be the case if the message processing failed just after RM-Submit was invoked. Similarly, if a message fails to be delivered on receiver side (*Deliver*) even after RM-Deliver has been successfully invoked, then a delivery failure Error shall be generated and reported either to the producer or the consumer, depending on the **P-Mode.ErrorHandling**.

- Have sufficient control on which RM sequence is used when submitting a message (RM-Submit), so that an RM sequence may be mapped to an ebMS conversation (`eb:ConversationId`).

Similar contracts apply to response messages (e.g. second leg of an ebMS Two-Way/Sync MEP), by switching Initiator MSH and Responder MSH in the above definitions.

11.2.4. Reliability for signal messages

Messages that have `eb:CollaborationInfo/eb:Service` set to `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service` are not intended to be delivered (*Deliver*) to an MSH consumer, although they may be submitted by an MSH producer. They are intended for internal MSH consumption. They may also be subject to reliability contracts. In this case, the at-least-once contract is fulfilled with a successful RM-delivery. In case of at-least-once delivery, a failure to deliver shall cause the generation of a delivery failure Error. If this message was submitted or initiated by an MSH producer (*Submit*) instead of the MSH itself, the producer may be notified (Notify) of the failure depending on the reporting mode, as for regular user messages.

11.2.5. Handling of delivery failures

Delivery is an abstract operation that may be implemented in various ways. It is the responsibility of an implementation or product to clearly state at what point in its processing it considers that a message is delivered. Such a statement amounts to defining a concrete "binding" to the *Deliver* operation, that a user can rely on for interpreting the reliability contracts defined and required in this document, relative to this implementation.

There are two options when supporting the At-Least-Once delivery contract:

1. Delivery failures are always notified to the producer (the sending side).

2. Delivery failures are always notified, though either to the producer or to the consumer (the receiving side), depending on the nature of the failure.

It is part of an agreement between parties to decide which notification option (1 or 2) shall be enforced. An MSH implementation may also be limited in its ability to support option 1. Conformance profiles for this document may require either option to be supported.

Delivery Failures (DFs) may be caused by network failure, or by processing failure on either side. In the remaining part of this subclause, the following is assumed:

- An MSH is always aware of processing failures that occur locally or that have been communicated to it, and it is always able to report these to its local party (producer or consumer) in some way. E.g. a message processing failure in a Receiving RMP can always be notified to the consumer.

- A DF that needs to be communicated from MSH to MSH should not itself rely on the transfer of an Error message (or a Fault), as such message may precisely also fail to be transferred. It is safer that it relies on the "non-transfer" of a message, such as a missing Acknowledgment.

NOTE: By relying on the non-reception of an Acknowledgment for notifying DF, "false" DFs can occur (in case of Acknowledgment loss), but the case where a message fails to be delivered unknowingly from the producer (false delivery) cannot occur. False DF - which can never be completely eliminated - can always be detected outside the reliable messaging (RM) layer, in a tractable and less urgent way - e.g. the sending party may synchronize on a daily basis by communicating its list of assumed delivery failures, for confirmation by receiver.

Restrictions in the ability to support notification option 1 usually depend on the semantics of Acknowledgment that is supported by the RMP. Three cases are to be considered:

Case 1: The acknowledgment is "on receipt" (as in WS-ReliableMessaging) and has no delivery semantics. In that case:

- DF notifications to the producer rely on lack of acknowledgments for network failures (non-reception of a user message)
- DF notifications to the producer rely on Error messages (or Faults) for any other failure occurring after reception, on consumer side.

For reasons mentioned above, this acknowledgment semantics does not generally support option 1. However, in the case of the HTTP binding with no intermediaries present, non-delivery due to processing failure can still be indicated in a reliable way to the sending side (and will trump the acknowledgment), as either a SOAP Fault is received on the HTTP response or the HTTP response fails.

The requirements for this transport-specific solution to option 1 which is reliable only for non-delivered pushed messages (as opposed to pulled) are detailed in Annex B.

Case 2: The acknowledgment is "on MSH-delivery" (supported in WS-Reliability). In that case, notification option 1 can be supported as well as option 2. In order for option 1 to be supported, an RMP shall implement RM-Deliver operation so that it is only considered successful (worthy of sending an acknowledgment) if the Deliver operation from MSH to consumer also succeeds. An implementation should support this acknowledgment semantics.

Case 3: The acknowledgment is "on RM-delivery" (supported in WS-Reliability). In case the condition in Case 2 is not supported by an RMP implementation, RM-Delivery is only concerning the RMP module and does not coincide with MSH delivery. Acknowledgments are "on RM-delivery" only.

Support for option 1 may be accomplished by relying on the transport-specific solution mentioned in Case 1. This solution is easier to implement here, as it only concerns the module processing the ebMS header (not the RMP implementation), as described in Annex B.

11.3. Reliability of ebMS MEPs

11.3.1. General

This subclause describes the reliability model for MEPs. For a concrete enumeration of all reliability options for MEPs in the context of an HTTP binding, see Annex E, which also shows how these combinations can be controlled with P-Mode parameters.

11.3.2. Reliability of the One-Way/Push MEP

The sequence of abstract operation invocations for a successful reliable instance of this MEP is as follows:

On Initiator MSH side:

- Step (1): **Submit**: submission of message data to the MSH by the producer party.
- Step (2): **RM-Submit**: after processing of ebXML headers, submission to the RMP.

On Responder MSH side:

- Step (3): **RM-Deliver**: after processing of reliability headers, delivery to other MSH functions.
- Step (4): **Deliver**: after processing of ebXML headers, delivery of message data to the consumer of the MSH.

NOTE: In case of delivery failure, either step (4) (*Deliver*) fails and Notify is invoked on Responder side, or both (3) and (4) fail and RM-Notify (then Notify) is invoked on either one of each side. A step "fails" either when it is not invoked in this sequence, or when it is invoked but does not complete successfully.

Figure 12 illustrates the message flow for this reliable MEP.

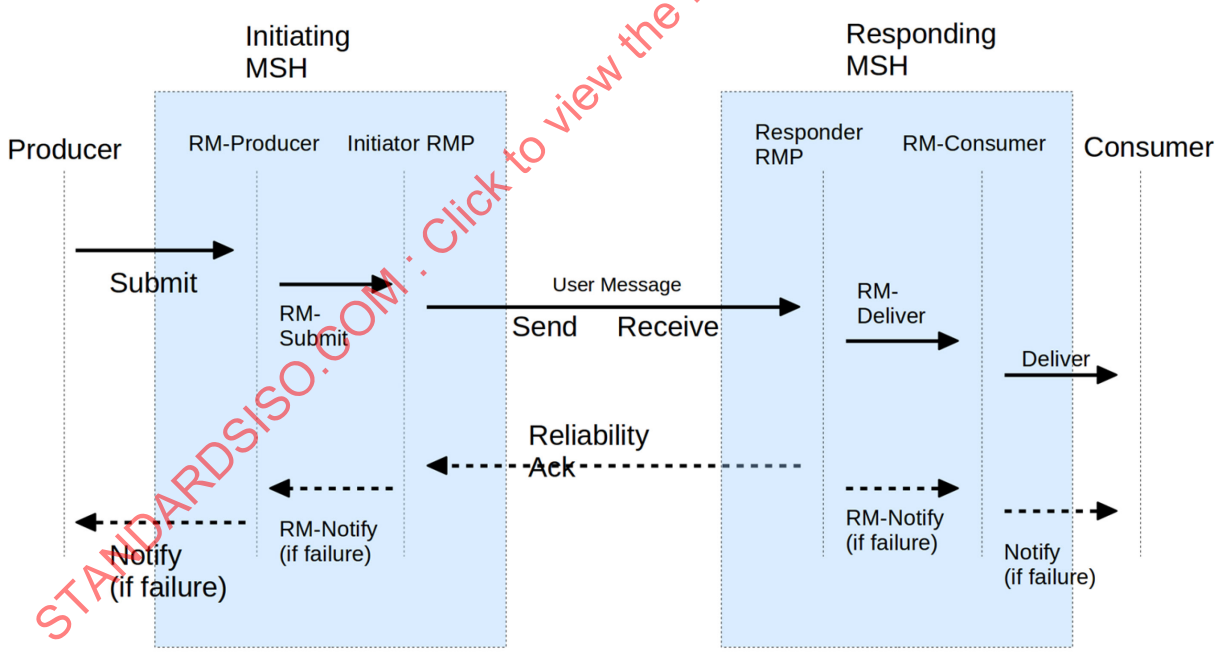

Figure 12: Reliable One-Way/Push MEP

The way in which the reliability acknowledgment binds to the underlying protocol - e.g. as a separate HTTP request, or on the back-channel of a previous message - is controlled by the P-Mode parameter **Reliability.AtLeastOnce.ReplyPattern**.

11.3.3. Reliability of the One-Way/Pull MEP

The processing model is as follows, for a typical and successful reliable instance of this MEP:

On Responder MSH side:

- Step (1): **Submit**: submission of message data to the MSH by the producer party, intended for the consumer on the Initiator side.

On Initiator MSH side:

- Step (2): Generation of a `PullRequest` signal by the MSH. **RM-Submit** is invoked on the Initiator RMP for this signal.

On Responder MSH side:

- Step (3): Reception of the `PullRequest` signal by MSH functions. **RM-Deliver** is invoked on the Responder RMP for this signal.
- Step (4): Submission of the pulled message to the RMP. This results in an **RM-SubmitResponse** invocation.

On Initiator MSH side:

- Step (5): **RM-DeliverResponse**: after processing of reliability headers of the pulled message, delivery to the RM-Consumer.
- Step (6): **Deliver**: after processing of ebMS headers, delivery of the pulled message data to the consumer of the MSH.

Figure 13 illustrates the message flow for this reliable MEP.

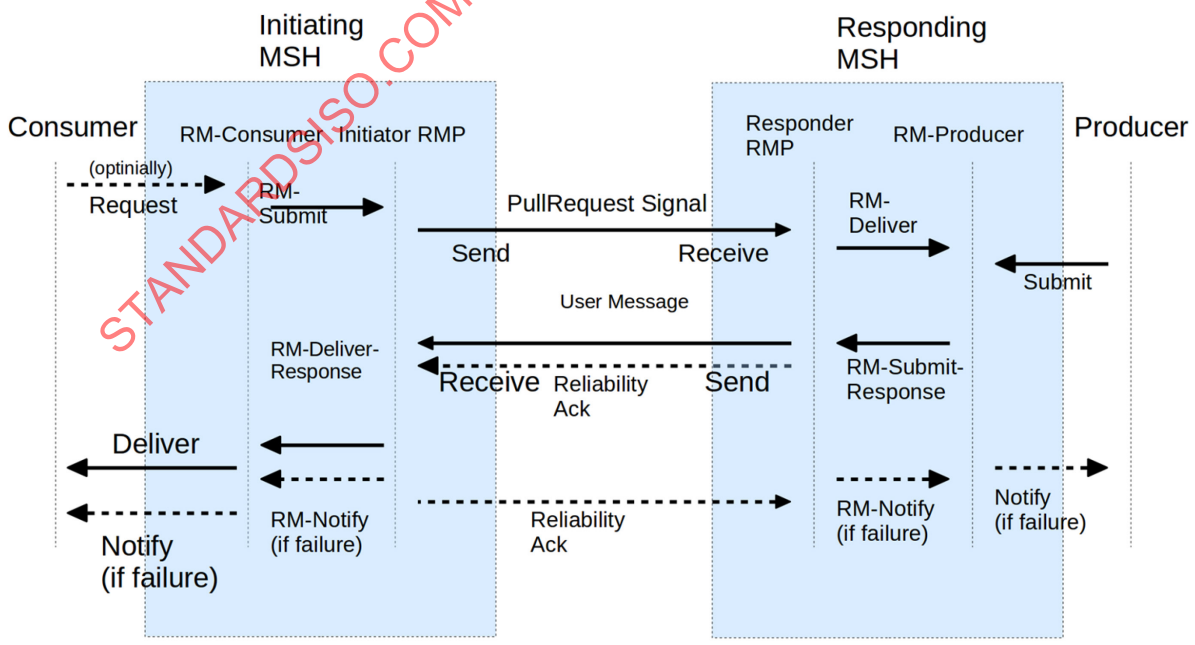

Figure 13: Reliable One-Way/Pull MEP

The way in which the reliability Acknowledgments are bound to the underlying protocol is controlled by the P-Mode parameter **Reliability.AtLeastOnce.ReplyPattern**.

In this MEP, as well as in the Simple Request-reply MEP defined in 11.3.4, the same reliability contracts that apply to the MEP request (here the `PullRequest` signal) may apply to the MEP response handled by RM-SubmitResponse and RM-DeliverResponse operations.

In such cases, when an MEP response is under reliability contract, the following requirements apply:

- When the MEP response is under At-Least-Once reliability contract, then the MEP request shall also be under At-Least-Once reliability contract. In addition, if the MEP request is also under At-Most-Once reliability contract, and if it has been delivered and responded to by the Responder RMP, then if a duplicate of the MEP request is received later, a duplicate of the same response that has been returned for the initial request shall be returned for the duplicate request. Note: depending on where a response delivery failure needs be notified (either on Initiator or Responding side, based on P-Mode.Reliability content), an acknowledgment may or may not need be returned for the response message by the Initiator RMP.

- When the MEP response is under At-Most-Once delivery, then the MEP request shall also be under At-Most-Once delivery.

11.3.4. Reliability of the Two-Way/Sync MEP

The processing model is as follows, for a typical and successful instance of this MEP:

On initiator MSH side:

- Step (1): **Submit**: submission of the request message data to the MSH by the producer party.
- Step (2): **RM-Submit**: submission of the request message to the initiator RMP.

On responder MSH side:

- Step (3): **RM-Deliver**: after processing of reliability headers, delivery of the request message to RM-Consumer.
- Step (4): **Deliver:** delivery of the request message data to the consumer of the MSH.
- Step (5): **Submit**: submission of a response message data to the MSH by the consumer of the request message, intended for the producer on the initiator side.
- Step (6): **RM-SubmitResponse:** submission by the RM-Producer of the response message to the responder RMP.

On initiator MSH side:

- Step (7): **RM-DeliverResponse**: delivery of the response message to the RM-Consumer.
- Step (8): **Deliver**: delivery of the response message data to the consumer of the initiator MSH.

Figure 14 illustrates the message flow for this reliable MEP.

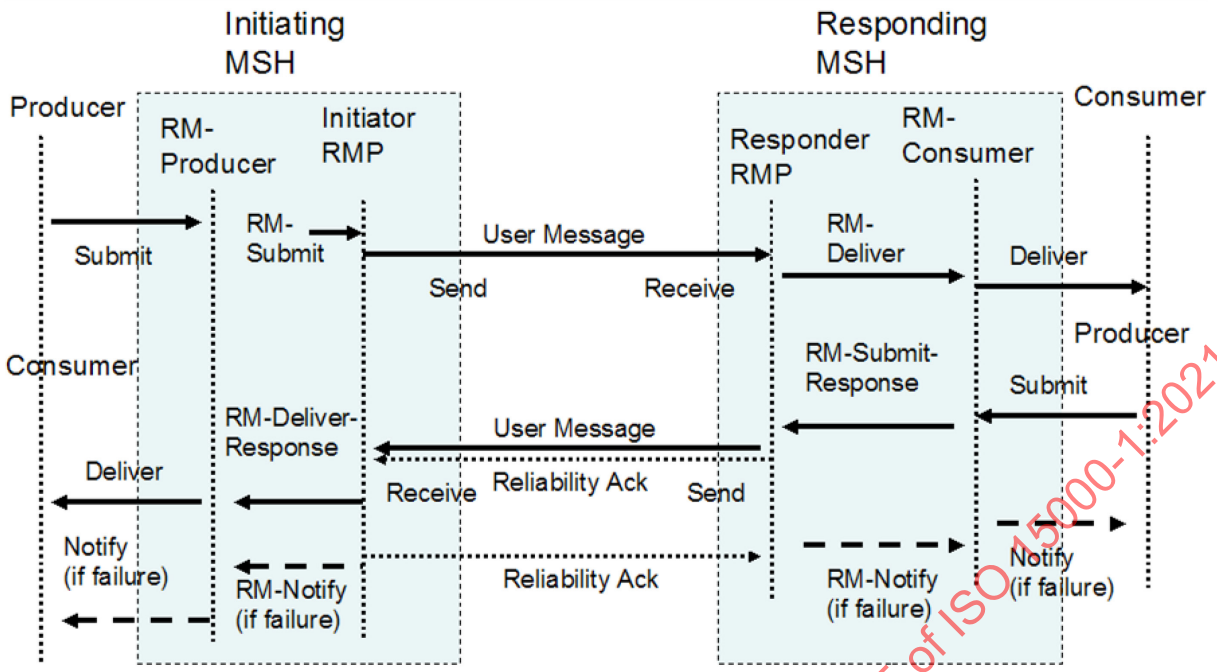

Figure 14: Reliable Two-Way/Sync MEP

The way in which the reliability acknowledgments are bound to the underlying protocol is controlled by the P-Mode parameter **Reliability.AtLeastOnce.ReplyPattern**.

When the MEP response is under reliability contract, the same dependencies with the reliability of the MEP request that are described for the One-Way/Pull MEP, also apply here.

11.3.5. Reliability of other transport-channel-bound MEPs

Each one of the MEPs defined in 5.2.8: Two-Way/Push-and-Push, Two-Way/Push-and-Pull, and Two-Way/Pull-and-Push, has been characterized as having a message choreography equivalent to a sequence of two of the previous MEPs (e.g. Two-Way/Push-and-Pull has a choreography equivalent to One-Way/Push + One-Way/Pull). The reliability of these more complex MEPs may be handled by composing reliable versions of these simpler exchanges, which are described in 11.3.2, 11.3.3 and 11.3.4. It can be noted that the reliable Two-Way/Push-and-Push MEP will not make use of the RM-SubmitResponse operation.

Annex A.
(informative)

The ebXML SOAP extension element schema

Following is the XML schema that describes the `eb:Messaging` header, as described in 8.2. This copy is provided for convenience only. The normative version of the schema is available as a separate file, at https://standards.iso.org/iso/15000/-1/ed-1/en/ebms-header-3_0-200704_2019.xsd.

```
<?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="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>
```

Annex B.
(informative)

Reliable messaging bindings

B.1. General

The reliability contracts defined in Clause 11 may be implemented by profiling different reliability specifications. Either one of two OASIS reliability specifications may be used by an MSH implementation: WS-Reliability 1.1, or WS-ReliableMessaging 1.1. Use of WS-Reliability shall conform to the OASIS. *WS-Reliability 1.1* specification. Use of WS-ReliableMessaging shall conform to OASIS. Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1.

Although either one of these OASIS reliability specifications is sufficient, each one has strong arguments in favor of its use. In the same way as two MSH implementations shall support the same transfer protocol or cryptographic algorithms in order to interoperate, two MSHs shall also implement the same reliability specification in order to have interoperable reliability features. The reliability specification being used in an implementation is a parameter of the conformance profiles for ebMS (see Annex G).

B.2. WS-Reliability binding

B.2.1. Operations and contracts binding

The reliable messaging processor (RMP) in ebMS is instantiated by the RMP as defined in WS-Reliability 1.1. To avoid confusion, we will call the RMP as defined in WS-Reliability 1.1 the WSR-RMP.

The RMP abstract operations RM-Submit, RM-Deliver, RM-SubmitResponse, RM-DeliverResponse and RM-Notify, map respectively to Submit, Deliver, Respond, Notify and Notify in WS-Reliability 1.1. Note that a single operation in WS-Reliability (Notify) is used to carry both notification of failure, and response message. In order to avoid confusion with WS-Reliability operations, the MSH operations Submit, Deliver, Notify, are respectively renamed in this subclause: MSH-Submit, MSH-Deliver, MSH-Notify.

The reliability contracts At-Least-Once Delivery, At-Most-Once Delivery and In-Order Delivery respectively map to the RM agreement items: GuaranteedDelivery, NoDuplicateDelivery, OrderedDelivery in WS-Reliability.

- Message processing faults such as FeatureNotSupported, PermanentProcessingFailure, or GroupAborted faults, when received by an RMP shall be communicated to the MSH. The MSH shall escalate such faults as DysfunctionalReliability ebMS errors (`EBMS:0201`).

- Message format faults, if they result in non-delivery, shall be escalated as DeliveryFailure ebMS errors (`EBMS:0202`).

B.2.2. Complement to the reliability of the One-Way/Push MEP

When At-Least-Once delivery is required, an Initiator MSH should be made aware of a delivery failure from the responder MSH to its consumer. Such a failure is notified to the producer party via MSH-Notify. In order to achieve this awareness, the RM-Deliver operation should be implemented so that it will fail if the MSH-Deliver invocation fails. In such a case the responder WSR-RMP generates a

MessageProcessingFailure fault, and will not acknowledge the reliable message that has not been successfully delivered by the Responder MSH to its consumer.

The RM-Agreement associated with the message, as defined in WS-Reliability, is restricted as follows:

- In case ReplyPattern has value "Poll" in a message sent reliably, the `PollRequest` sent later by the sending RMP for this message shall be synchronous (the `ReplyTo` element shall not be present).

B.2.3. Complement to the reliability of the One-Way/Pull MEP

When At-Least-Once delivery is required, a responder MSH should be made aware of a delivery failure from the initiator MSH to its consumer. Such a failure is notified to the producer party (Responder side) via MSH-Notify. In order to achieve this awareness, the RM-DeliverResponse operation should be implemented so that it will fail if the MSH-Deliver invocation fails (Initiator side). In such a case the Initiator WSR-RMP generates a **MessageProcessingFailure** fault, and will not acknowledge the reliable message that has not been successfully delivered by the Initiator MSH to its consumer.

The RM-Agreement associated with the pulled message shall comply with the restrictions expressed in Table 5.

Table 5: Requirements for Reliability of the One-Way/Pull MEP

Name	*Allowed Values*	*Additional Requirements*
GuaranteedDelivery	"enabled", "disabled"	When enabled, the pull request signal message associated with this pulled message shall be also sent with this parameter enabled. When the pull request signal is sent with GuaranteedDelivery enabled, two additional requirements shall be satisfied: 1. The ReplyPattern value associated with the pull request signal is "Response". 2. The NoDuplicateDelivery agreement item is also enabled for the pull request signal. The Responder RMP sends back a copy of the original pulled message if the latter is not expired, when a duplicate of the pull request signal is received, e.g. due to resending (see 11.3.3). This is achieved by supporting the first option for responding to duplicates of messages sent with Response ReplyPattern (OASIS *WS-Reliability 1.1*, 2004, clause 3.2.2, second part of protocol requirements).
NoDuplicateDelivery	"enabled", "disabled"	When enabled, the pull request signal message associated with this pulled message shall also be sent with this parameter enabled.
OrderedDelivery	"enabled", "disabled"	No restriction.
ReplyPattern	"Callback"	

WS-Reliability 1.1 is silent about the reliability of messages submitted as responses to other messages, over the same SOAP MEP instance. Such messages would be submitted using the abstract operation RM-Respond, which requires a WSR-RMP to correlate the response message with the related request. This document requires that the reliability of these responses, in the case of pulled messages, be also supported by the Responder MSH. This means that the implementation of WSR-RMP used in an MSH should also support RM agreements that cover such responses.

B.2.4. Complement to the reliability of the Two-Way/Sync MEP

As already mentioned for the One-Way/Push MEP and the One-Way/Pull MEP when At-Least-Once delivery is required, that the Initiator MSH should be made aware of a request delivery failure from the Responder MSH to its consumer, and also that the Responder MSH be made aware of a response delivery failure from the Initiator MSH to its consumer.

The RM-Agreement associated with the request message shall comply with the same restrictions as for the One-Way/Push MEP, and also with those entailed by the RM-Agreement options used for the response message. The RM-Agreement associated with the Response message shall comply with the restrictions expressed in Table 6.

Table 6: Requirements for Reliability of the Two Way Sync MEP

Name	*Allowed Values*	*Additional Requirements*
GuaranteedDelivery	"enabled", "disabled"	When enabled, the Request message associated with this Response message shall be also sent with this parameter enabled. When the Request is sent with GuaranteedDelivery enabled, two additional requirements shall be satisfied: 1. The ReplyPattern value associated with the pull request signal is "Response". 2. The NoDuplicateDelivery agreement item is also enabled for the Request. The Responder WSR-RMP sends back a copy of the original Response message if the latter is not expired, when a duplicate of the Request is received, e.g. due to resending (see 11.3.3). This is achieved by supporting the first option for responding to duplicates of messages sent with Response ReplyPattern (OASIS *WS-Reliability 1.1*, 2004, clause 3.2.2, second part of protocol requirements).
NoDuplicateDelivery	"enabled", "disabled"	When enabled, the Request message associated with this Response message shall also be sent with this parameter enabled.
OrderedDelivery	"enabled", "disabled"	No restriction.
ReplyPattern	"Callback"	

NOTE: The request message and response message can have a different RM-Agreement.

B.3. WS-ReliableMessaging binding

B.3.1. Operations and contracts binding

The Reliable Messaging Processor (RMP) in ebMS is mapping to the following notions in WS-RM [WS-ReliableMessaging]: the sending RMP maps to RMS (reliable messaging source), the receiving RMP maps to RMD (reliable messaging destination).

The RMP abstract operations RM-Submit, RM-Deliver, map respectively to Send, Deliver in WSRM. So do RM-SubmitResponse, RM-DeliverResponse, as there is no distinction in applying reliability features to a SOAP request and to a SOAP response in WS-RM. RM-Notify shall be implemented so that failures detected by RMS are escalated to the MSH as follows:

- CreateSequenceRefused, SequenceTerminated, SequenceClosed, MessageNumberRollover or UnknownSequence faults, when received by an RMS and when the RMS cannot establish a substitute sequence that would support reliable transmission of messages in the same conditions as the failed sequence would have, shall be communicated to the MSH on the Source side. The MSH shall escalate such faults as DysfunctionalReliability ebMS errors (`EBMS:0201`).

- WSRM-Required fault, when received by an RMS, shall be communicated to the MSH on Source side. The MSH shall escalate such faults as ProcessingModeMismatch (`EBMS:0010`). The RM Error code should be reported in the ErrorDetail element of `EBMS:0010`.

- InvalidAcknowledgment and UnknownSequence, when received by the RMD, shall be communicated to the MSH on Destination side. The MSH shall escalate such faults as DysfunctionalReliability ebMS errors (`EBMS:020N`).

The reliability contracts At-Least-Once Delivery, At-Most-Once Delivery and In-Order Delivery map to equivalent delivery assurance definitions in the WS-RM specification. Although WS-RM does not mandate support for these delivery assurances (DAs), and only specifies the protocol aspect, a conformance profile supporting reliable messaging requires the use of a WS-RM implementation (RMD) that supports at least some of these DAs as extensions.

All messages transmitted over a same sequence should use the same MPC. This becomes a requirement for the In-Order reliability contract.

NOTE: The WS-RM protocol always assumes acknowledgment of messages. Although acknowledgments are unnecessary for the At-Most-Once reliability contract, the use of sequence numbers allows for an efficient duplicate detection. The WS-RM protocol for At-Most-Once should then be used.

Parameters of the WS-RM protocol such as acknowledgment interval, timeouts, resending frequency, etc. may be specified in the Processing Mode, as extensions to the PMode.Reliability group (see Annex D).

Sequence acknowledgments and sequence operations (such as CreateSequence, CreateSequenceResponse) shall use MEPs of the underlying protocol in a way that is compatible with the conformance profile of the MSH which defines the ebMS MEPs that shall be supported, along with the underlying protocol binding. For example, if the ebMS conformance profile for an MSH only requires ebMS messages to be reliably pulled by this MSH over HTTP, then their sequence shall either be created by a CreateSequence message carried over an HTTP response, the HTTP request being initiated by this MSH, or be offered (using `wsrx:Offer`) by the CreateSequence used for opening a sequence for sending Pull signals reliably.

Either one of the two following options shall be used, in order to enable MSH interoperability based on WS-ReliableMessaging, regarding the reliability contracts for messages exchanged between two MSHs:

1. The reliability contract and parameters apply equally to all messages sent between two MSHs. All messages exchanged in the same direction between two MSHs are subject to the same reliability quality of service. In such a case, the P-Modes.Reliability parameters associated with each one of these messages shall not conflict with this common quality of service.

2. The reliability contract and parameters may vary from one message to the other. In that case, the scope of application of a reliability contract shall be the sequence, meaning all messages within the same sequence are subject to the same reliability contract.

When support for case (2) above is required, the source of a sequence (RMS) shall be able to indicate which delivery assurance is associated with this sequence, so that the RMD implements the expected DA. Indeed, although both MSHs share knowledge of the reliability contracts associated with each message (P-Mode.reliability), the RMD has no access to the ebMS header, and can only rely on the sequence number. In order to avoid the constraint of using predefined sequence numbers, the association DA-sequence shall be dynamically supported by an RMS. Consequently, an implementation of WS-ReliableMessaging that supports case (2) shall also support the extension of the `wsrx:CreateSequence` element with a child element structured as a policy assertion as defined in OASIS *Web Services Reliable Messaging Policy (WS-RM Policy) Version 1.1*, i.e. either one of the following:

```
(a) <wsrmp:AtLeastOnceDelivery wsrmp:InOrder='true|false'/>
(b) <wsrmp:AtMostOnceDelivery wsrmp:InOrder='true|false'/>
(c) <wsrmp:ExactlyOnceDelivery wsrmp:InOrder='true|false'/>
```

The above extensions shall also be supported in `wsrx:Accept/{any}` and understood, in case of a conformance profile that requires support for reliable One-Way/Pull or reliable Two-Way/Sync. The above extensions should be supported in `wsrx:Offer/{any}` and understood.

The above DA assertion (a) shall match a P-Mode.Reliability with parameters AtMostOnce.Contract = "false", AtLeastOnce.Contract = "true"; and its attribute `@wsrmp:InOrder` shall match the InOrder.Contract value.

The above DA assertion (b) shall match a P-Mode.Reliability with parameters AtMostOnce.Contract = "true", AtLeastOnce.Contract = "false"; and its attribute `@wsrmp:InOrder` shall match the InOrder.Contract value.

The above DA assertion (c) shall match a P-Mode.Reliability with parameters AtMostOnce.Contract = "true", AtLeastOnce.Contract = "true"; and its attribute `@wsrmp:InOrder` shall match the InOrder.Contract value.

Additional reliability parameters – if any, e.g. resending frequency, etc. - associated with each one of the reliability contracts (At-Least-Once, At-Most-Once, In-Order) are to be defined in P-Mode.Reliability extensions and known from both parties prior to the exchange with no need to be transmitted via the RM protocol. When receiving a CreateSequence message with the above extension specifying a reliability contract, the RMD shall be able to resolve it to a single set of additional parameters governing this mode of reliability. For example, the P-Modes of all messages sent under At-Least-Once should have same values for the set of PMode.Reliability parameters related to this contract (AcksTo, AcksOnDelivery, ReplyPattern and any other custom parameters such as those controlling message resending, if any), as well as for the NotifyProducerDeliveryFailures parameter about failure reporting.

Because acknowledgments in WS-ReliableMessaging are on receipt, the Reliability.AckOnDelivery parameter in the P-Mode of messages sent reliably shall be "false".

B.3.2. Complement to the reliability of the One-Way/Push MEP

When At-Least-Once delivery is required for the ebMS user message carried by this MEP, the RMP on Initiator side is acting as an RMS, and the RMP on Responder side is acting as an RMD.

The sequence should be initiated by the RMS sending a `wsrx:CreateSequence` message, as opposed to responding to an `wsrx:Offer`.

In case the P-Mode.Reliability.AtLeastOnce.ReplyPattern has value "Response", then the `CreateSequence/AcksTo` element shall contain an WS-Addressing anonymous IRI.

In case the P-Mode.Reliability.AtLeastOnce.ReplyPattern has value "Callback", then the `CreateSequence/AcksTo` element shall contain an URI specified in an additional P-Mode.Reliability parameter.

The P-Mode.Reliability.AtLeastOnce.ReplyPattern shall not have value "Poll",

When an underlying two-way protocol is used, any pair of sequence lifecycle message (`CreateSequence/CreateSequenceResponse`, `CloseSequence/CloseSequenceResponse`, `TerminateSequence/TerminateSequenceResponse`) should be exchanged over a single request-response MEP of the protocol.

The Initiator MSH should be made aware of a delivery failure from the Responder MSH to its consumer (`NotifyProducerDeliveryFailures` = "true"). Such a failure is notified to the producer party via Notify.

- A failure to deliver that is detected by the RMS, e.g. failure to get an acknowledgment for a sent message, shall be communicated to the Initiator MSH. The MSH shall escalate such a fault as `DeliveryFailure` ebMS errors (`EBMS:0202`).

- A failure to deliver that is detected by the RMD (responder side), e.g. failure to deliver (operation Deliver) after the message has been received and acknowledged by the RMD, shall be communicated to the responder MSH. The MSH shall escalate such a fault as `DeliveryFailure` ebMS errors (`EBMS:0202`). This ebMS error should be reported to the Initiator MSH.

B.3.3. Complement to the reliability of the One-Way/Pull MEP

When At-Least-Once delivery is required for the ebMS user message carried by this MEP, the RMP on Responder side is acting as an RMS, and the RMP on Initiator side (which sent the pull request) is acting as an RMD.

When initiating an instance of the One-Way/Pull MEP, and if it is expected – based on P-Modes deployed - that pulled message may be sent reliably, then the pull request signal itself shall be sent under At-Least-Once delivery (see Clause 11). Acknowledgments for Pull signals should be sent over the second leg of the One-Way/Pull MEP (PMode.Reliability.AtLeastOnce.ReplyPattern ="Response"), bundled with the pulled ebMS user message. However the frequency of acknowledgments may not need be on a per message basis.

In case pulled messages shall be sent reliably, the following requirements apply:

- When a sequence is initiated (CreateSequence) to be associated with pull request signals intended for the same MPC, then the `wsrx:Offer` shall be present in the `CreateSequence` element. The offered sequence should be used for sending back pulled messages reliably.

- When no more messages have to be pulled reliably from an MPC, the sending MSH should close and terminate the associated sequences. When the sending MSH decides to terminate a reliable sequence of pulled messages, a CloseSequence message or a TerminateSequence should be sent over a pulled message, e.g. piggybacked over the EmptyMessagePartitionChannel warning (`EBMS:0006`).

The responder MSH should be made aware of a delivery failure from the initiator MSH to its consumer. Such a failure is notified to the producer party (responder side) via *Notify*.

- A failure to deliver that is detected by the RMS, e.g. failure to get an acknowledgment on the Responder side for a sent message, shall be communicated to the responder MSH. The MSH shall escalate such a fault as DeliveryFailure ebMS errors (`EBMS:0202`).

- A failure to deliver that is detected by the RMD (initiator side), e.g. failure to deliver (operation *Deliver*) after the message has been received and acknowledged by the RMD shall be communicated to the initiator MSH. The MSH shall escalate such a fault as DeliveryFailure ebMS errors (`EBMS:0202`). This ebMS error should be reported to the responder MSH.

B.3.4. Complement to the reliability of the Two-Way/Sync MEP

In the reliable Two-Way/Sync MEP, either:

- The request message alone is sent reliably, in which case the requirements and recommendations for the One-Way/Push also apply here.

- Or both the request and the reply are sent reliably. The response alone shall not be sent reliably.

In case both request and reply are sent reliably, both sequences should be established and discarded in a coordinated way. The same rules apply as for the reliability of the One-way Pull MEP. The in-bound sequence termination should be terminated on the initiative of the MEP Initiator, after the out-bound sequence is terminated.

Annex C.
(informative)

SOAP format and bindings

C.1. General

This annex specifies the SOAP format (SOAP versions, packaging of attachments and/or binary data) used in ebMS-3, as well as how this SOAP format is transported over HTTP and SMTP. Transport over HTTP shall conform to RFC 2616. Transport over SMTP shall conform to RFC 2821.

ebMS-3 does not require the usage of SOAP 1.1 and/or SwA (SOAP-1.1 With Attachments). We consider the attachments specification of SwA as being orthogonal to the SOAP version. In other words, attachments could well be used for SOAP 1.2 in the same way they are used for SOAP 1.1. Similarly, we also consider MTOM being orthogonal to the SOAP version (however, MTOM will not be addressed in this core specification).

A conformant implementation of ebMS-3 may well choose to use SOAP-1.2 instead of SOAP-1.1. Since SwA is orthogonal to the SOAP version, there are two possibilities:

(1) An implementation of ebMS-3 may choose SOAP-1.1 with Attachments

(2) An implementation of ebMS-3 may choose SOAP-1.2 with Attachments

Although a SOAP 1.2 version of SwA has not been formally submitted to W3C, it appears that most SOAP products have anticipated that usage, and after investigation, it appears that they have done so in a consistent, interoperable way. This document is acknowledging these *de facto* upgrades of SwA, which are summarized in this annex.

SwA uses the multipart/related MIME encapsulation. This encapsulation is independent of the version of SOAP being used (in fact it can encapsulate any XML document, not just SOAP), and also independent of the transport protocol (the encapsulation could be transported via HTTP, SMTP, etc.).

C.2. Using SwA with SOAP-1.1

The following example shows an ebMS-3 message using SOAP 1.1 with attachments. The ebMS-3 message in this example contains two payloads:

- The first payload is the picture of a car. This picture is in binary form as an attachment with a Content-ID equal to "`car-photo@cars.example.com`".
- The second payload is an XML fragment within the SOAP `Body`. This XML fragment has `id` attribute equal to "`carData`"

The XML fragment in the SOAP `Body` contains a reference to another binary data, namely the picture of the car owner).