1.
(DRAFT)

May 31, 2004

Editor: Jacques Durand

Table of Contents

1.Messaging Model

1.1 Terminology and Concepts

1.1.1 Components of the Model

The ebMS messaging model assumes the following components:

- **ebMS MSH (Message Service Handler)**: An entity able to generate or process ebMS messages, and to act in at least one of two ebMS roles defined below: Sending and Receiving. In terms of SOAP processing, an MSH is either a SOAP processor [SOAP 1.1] <*decide about SOAP 1.2 option* > or a chain of SOAP processors. In either case, an MSH must be able to understand headers intended to actors <ebMS URI>, <reliability>, <security>. The transmission of an ebMS message requires a Sending MSH and a Receiving MSH.

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

- **User-Consumer (or Consumer)**: An entity that interacts with a Receiving MSH (i.e. an MSH in Receiving role) to consume data from a received ebMS user message. Some examples are: an application, a queuing system, another SOAP processor. Message data can be consumed in various forms: a programming object combining payload and header elements, a SOAP message, etc.

Figure MM1. Entities of the Messaging Model and their Interaction

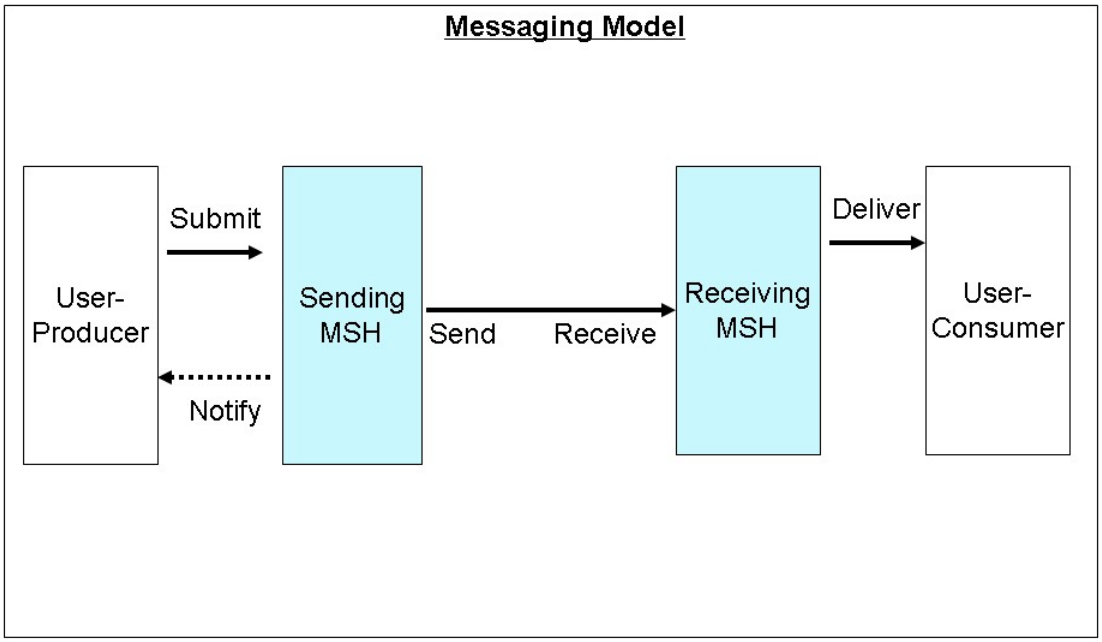


Figure MM1 shows the entities and operations involved in a message exchange.

1.1.2 Message Types

An ebMS MSH component must be able to exchange the following types of messages :

- **ebMS Message**: A message that contains SOAP header(s) qualified with the ebMS namespace, and that conforms to this specification.
- **ebMS Signal Message**: An ebMS message that is initiated by an MSH (instead of initiated by a user-producer using this MSH) and intended to another MSH (i.e. not intended to a user-consumer associated with this other MSH). An example is the ebMS PullRequest signal.
- **ebMS User Message**: An ebMS message that is initiated by a user-producer entity, and intended to a user-consumer. (i.e. an ebMS message that is not an ebMS signal message.)

1.1.3 Messaging Roles

The Messaging Model assumes the following roles for an MSH:

- **Sending**: An MSH acts in Sending role when performing functions associated

with generating a message and sending this message to another MSH. When the message is a user message this involves getting message data from a Producer entity. The abstract operations Submit, Send, Notify and Pull (see the Pull MEP) are supported by this role.

- **Receiving**: An MSH acts in Receiving role when performing functions associated with the receiving and processing of ebMS user messages, and passing message data to a Consumer entity. The abstract operations Receive and Deliver are supported by this role. (Note that in a Receiving role, an MSH could also have to send ebMS signal messages.)

1.1.4 Abstract Messaging Operations

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

- **Submit:** When invoked, the operation transfers message data from the user-producer to the Sending MSH. The operation is successful when an ebMS user message has either been sent, or queued (i.e. ready for pulling, when the Sending MSH is in Pull mode).

- **Deliver:** When invoked on a Receiving MSH, the operation makes an ebMS user message data available to the user-consumer.

- **Notify:** When invoked on a Sending MSH, the operation notifies a user-producer about the status of a previously submitted ebMS user message, or about general MSH status.

- **Pull:** This operation is usually invoked by reception of the PullRequest ebMS signal. When invoked on a Sending MSH, the operation initiates the sending of an ebMS user message that was queued, as a SOAP response to the PullRequest signal.

- **Send**: When invoked on a Sending MSH, the operation initiates the transfer of a user message from the Sending MSH toward the Receiving MSH.

- **Receive**: When invoked on a Receiving MSH, the operation completes the transfer of a user message from the Sending MSH to the Receiving MSH.

1.1.5 Supported Topologies

An ebMS MSH may be used in the following topologies of messaging. In every one of these topologies, the MSH can either be implemented as a SOAP node, or as a chain of SOAP nodes that understand different headers: (a) a node that understands ebMS headers, (b) a node that understands reliability headers, and (c) a node that understands security headers.

- MSH as a messaging endpoint. In this context, a sending MSH is a SOAP

message originator. A receiving MSH is an ultimate SOAP receiver. In such a case, the Deliver operation consists of delivering the message to an application-level consumer, such as a business application, or a queuing system. The Submit operation consists of passing to the MSH the data and payload that are necessary so that it can generate an ebMS message.

- MSH as a SOAP Intermediary. In this context, a sending MSH is acting as SOAP intermediary and just adding headers to a SOAP message it received (via Submit operation). For an MSH to be used in this context, it is recommended that no security header be present in the input SOAP message. At least, no signature other than detached XML signature shall be present, so that the addition of headers does not break signatures. A receiving MSH acting as SOAP intermediary will process and remove some headers (including ebMS headers, reliability and some security headers) and forward the SOAP message – which is no longer an ebMS message - to the next SOAP node.

- MSH as intermediary in a MSH multi-hop chain. In this context, a sending MSH acting as an MSH intermediary is forwarding the ebMS messages it received (via Receive operation) toward an other MSH (via operation Send). See the multi-hop module for the behavior of an MSH intermediary.

1.2 Message Exchange Patterns

1.2.1 Definition

A Message Exchange Pattern (MEP) is an abstract description of a typical sequence of message exchanges that may occur between two or more MSH instances. An MEP instance is a message exchange that conforms to the pattern described in the MEP. An ebMS MEP is only defined in terms of ebMS messages, i.e. as SOAP messages carrying ebMS-qualified headers. We only consider here ebMS MEPs that involve at least one ebMS user message. ebMS MEPs will be distinguished based on two characteristics:

- For each ebMS user message involved, the direction of this message between two partners.
- For each ebMS user message involved, the mode of transfer: Push, Pull.

When more than one ebMS user message is exchanged in the same MEP, there must be some explicit referencing between them. In other words, for every user message in the same MEP instance, either one of these statements is true:

- the message is the first user message to occur in the MEP instance.
- the message is referring to the ebMS ID of another user message (and only one), in the same MEP instance.

The MSH sending of the first message of an MEP instance is called the *initiator* MSH. The other(s) MSH is (are) called *responding* MSH(s).

1.2.2Assumed SOAP Message Exchange Patterns

Each ebMS MEP is also defined in terms of what SOAP MEP(s) it is using. Two SOAP MEPs must be supported for implementing the ebMS MEPs described in this specification:

SOAP One-way MEP:

From an MSH perspective, support for this MEP assumes the following:

- The Sending RMP (as a SOAP node) is able to initiate the sending of a SOAP envelope over the underlying protocol (i.e., not as a result of a previous protocol action such as an HTTP GET or POST).
- No response containing a SOAP envelope is sent back – although a non-SOAP response (e.g., an HTTP error code) may be returned.

SOAP Request-response MEP:

From an MSH perspective, support for this MEP assumes the following:

- The Sending RMP is able to initiate the sending of a SOAP envelope over the underlying protocol (i.e., not as a result of a previous protocol action such as an HTTP GET or POST).
- The Receiving RMP can send back a message with a SOAP envelope (called a response) after somehow associating the response with the request. (For example, this association can be realized by the use of a request-response underlying protocol such as HTTP.)

The full definition of this MEP can be found in [SOAP] part 1, Adjunct. The concept of SOAP MEP has only been introduced with SOAP 1.2, although this concept can apply as well to SOAP 1.1. As far as an MSH is concerned, the above properties of the SOAP One-way and Request-response MEPs are assumed, regardless of the SOAP version (1.1 and 1.2).

1.3 Simple ebMS Message Exchange Patterns

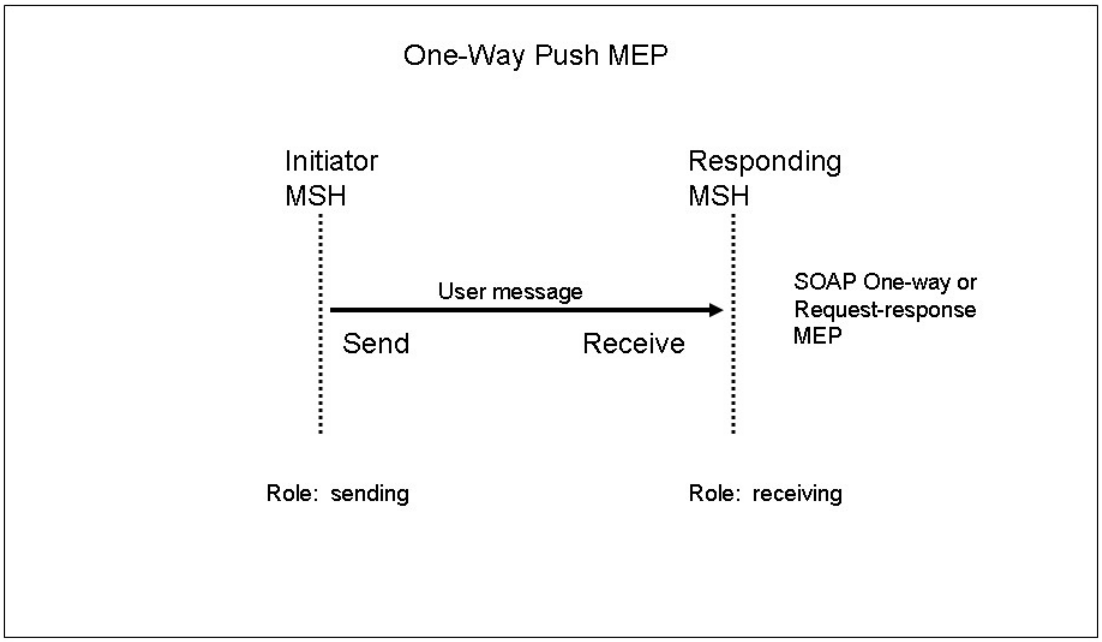
A simple ebMS MEP maps to a single SOAP MEP instance. This specification identifies three simple MEPs: One-Way Push, One-Way Pull, and Request-Response. The ebMS user messages that participate in a simple MEP must specify the name of this MEP in the header element eb:mep.

1.3.1 The One-Way Push Message Exchange Pattern

This MEP involves a single ebMS user message. In a One-Way Push MEP the message is sent by a Sending MSH either over a SOAP One-way MEP instance or as a SOAP Request in a SOAP Request-response MEP instance.
In case the Receiving MSH returns a SOAP response, the latter must NOT contain an ebXML user message. Figure MM2 illustrates the exchange pattern and operations

involved for this MEP. The value of eb:mep in the user message is “push”.

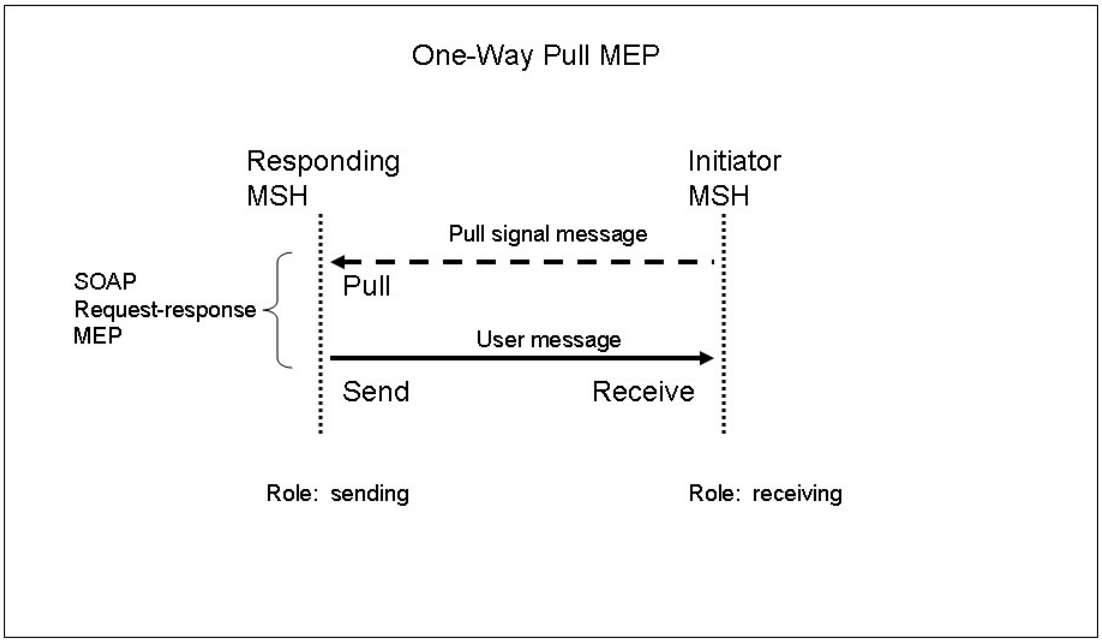
Figure MM2. One-Way Push MEP.



1.3.2 The One-Way Pull Message Exchange Pattern

This MEP involves a single ebMS user message. In this MEP the message is sent as a SOAP Reponse over a SOAP Request-response MEP instance. The first leg of the SOAP MEP (Request) sends the PullRequest signal. The second leg of the MEP returns the pulled user message. In case no message is available for pulling, a SOAP Response is returned with a StatusResponse ebMS signal that indicates an “empty queue”. Figure MM3 illustrates the exchange pattern and operations involved for this MEP. The value of eb:mep in the user message is “pull”.

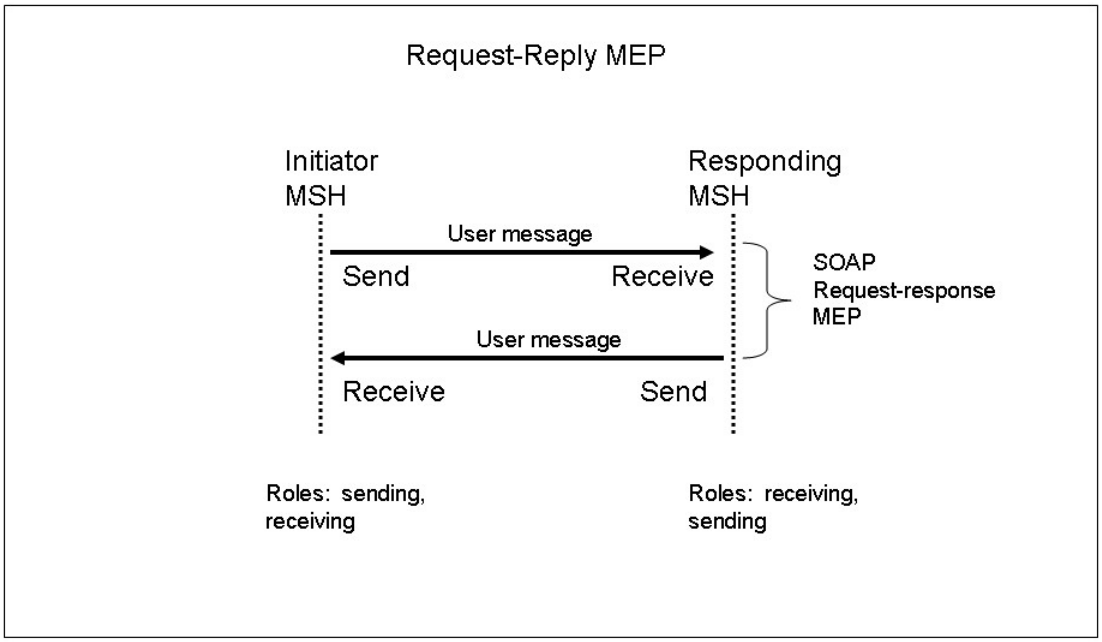
Figure MM3. One-Way Pull MEP



1.3.3 The Request-Reply Message Exchange Pattern

This MEP involves two ebMS user messages over a single SOAP Request-response MEP. In the first leg of the MEP, a message called the “ebMS request” is sent over the SOAP Request message. In the second leg of the MEP, a message called the “ebMS reply” is sent as the SOAP Response. The ebMS reply refers to the message ID of the ebMS request via the wsa:RelatesTo header (see Section...). Figure MM4 illustrates the exchange pattern and operations involved for this MEP. The value of eb:mep in the user messages involved is “request-reply”.

Figure MM4. Request-Reply MEP.



1.4 Aggregate ebMS Message Exchange Patterns

An aggregate ebMS MEP is composed of two or more simple ebMS MEPs. (An instance of such an MEP maps to two or more SOAP MEP instances.) Among an infinity of aggregate MEPs. users can define their own. For illustration purpose, this specification defines three aggregate MEPs among the most common. Although an MSH is not required to process or to validate the definition of aggregate MEPs, this specification provides means to define and identify them in the message header so that their definition does not depend on payload data, and so that their processing (e.g. monitoring) is more efficient.

For each one of the simple MEP instances that participate in the same instance of aggregate MEP, either one of the following is true :

- The simple MEP instance is the first one to occur in the aggregate instance.
- The first user message of the simple MEP is referring to a message ID of a previous simple MEP instance in the same aggregate instance.

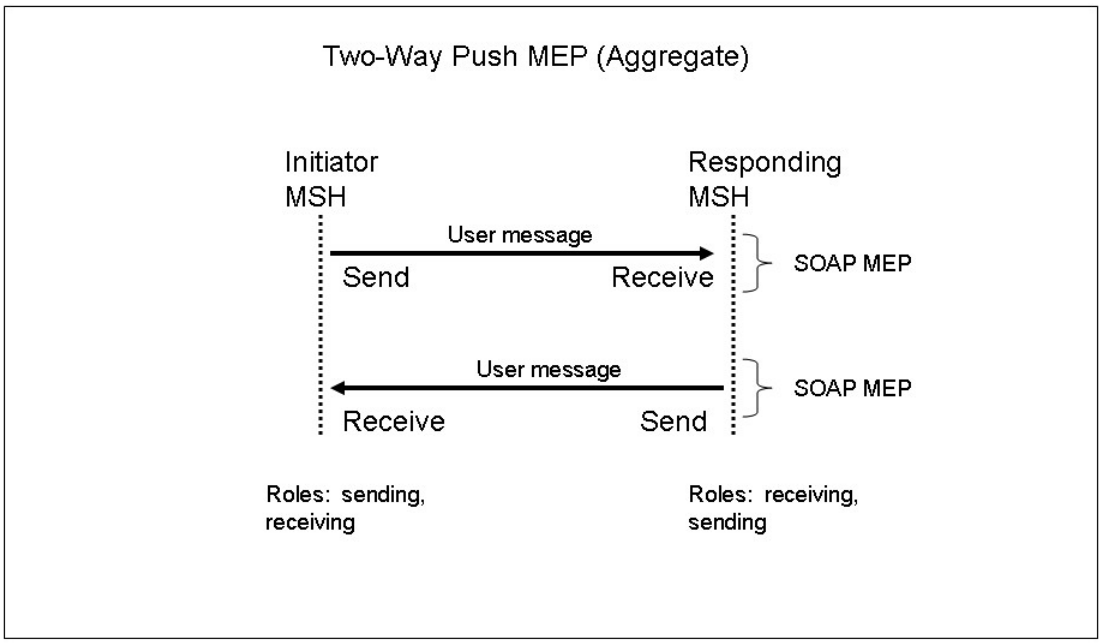
An aggregate ebMS MEP is identified by a URI that is generally user-defined. It is recommended that this URI be specified in a user property of name “aggmep” in the ebMS header. Instances of an MEP (simple or aggregate) may be identified by another user property. User messages that participate in an aggregate MEP must specify the simple MEP they also participate in (eb:mep).

This specification identifies three aggregate MEPs: Two-Way Push, Push-and-Pull, and Pull-and-Push, each of them using two simple MEPs.

1.4.1 The Two-Way Push Message Exchange Pattern

This MEP involves two ebMS user messages. It is composed as a sequence of two One-Way Push MEPs. The user message in the second One-Way Push MEP refers to the message ID of the user message in the first One-Way Push MEP. This MEP can be used to correlate asynchronously a response message with a request message. Figure MM5 illustrates the exchange pattern and operations involved for this MEP.

Figure MM5. Two-Way Push MEP.

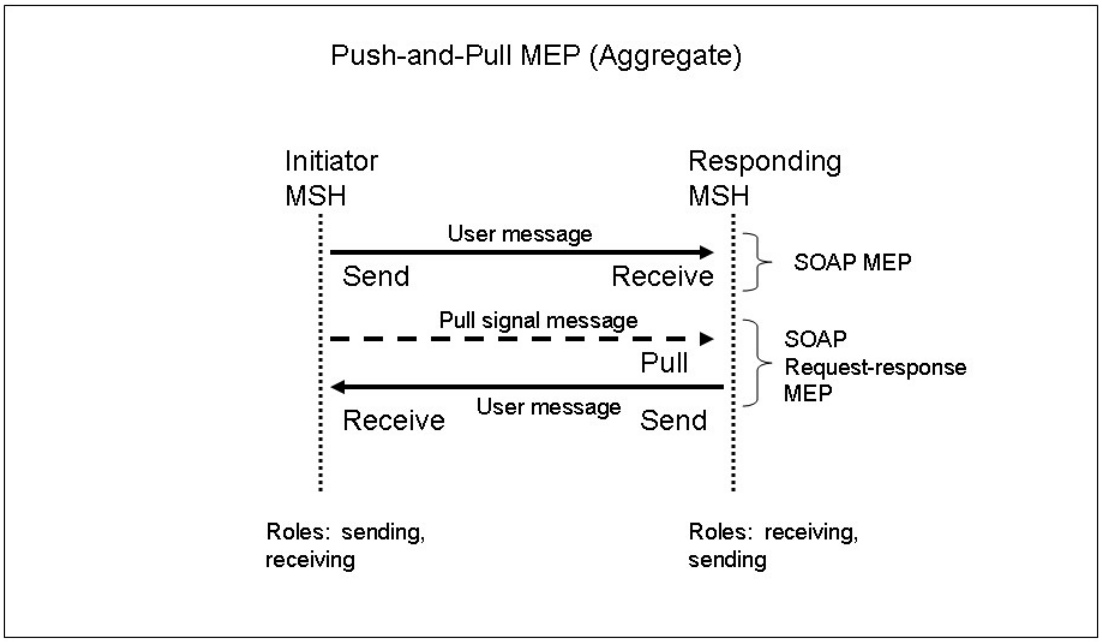


1.4.2 The Push-and-Pull Message Exchange Pattern

This MEP involves two ebMS user messages. It is composed of a One-Way Push MEP followed by a One-Way Pull MEP. The user message in the second simple MEP refers to the message ID of the user message in the first simple MEP. This aggregate MEP can be used to correlate asynchronously a response message with a request message, with both exchanges being initiated by the same MSH (here, the sender of the request). This may accommodate situations where an MSH does not allow for incoming connections, e.g. due to firewall restrictions. Figure MM6 illustrates the exchange

pattern and operations involved for this MEP.

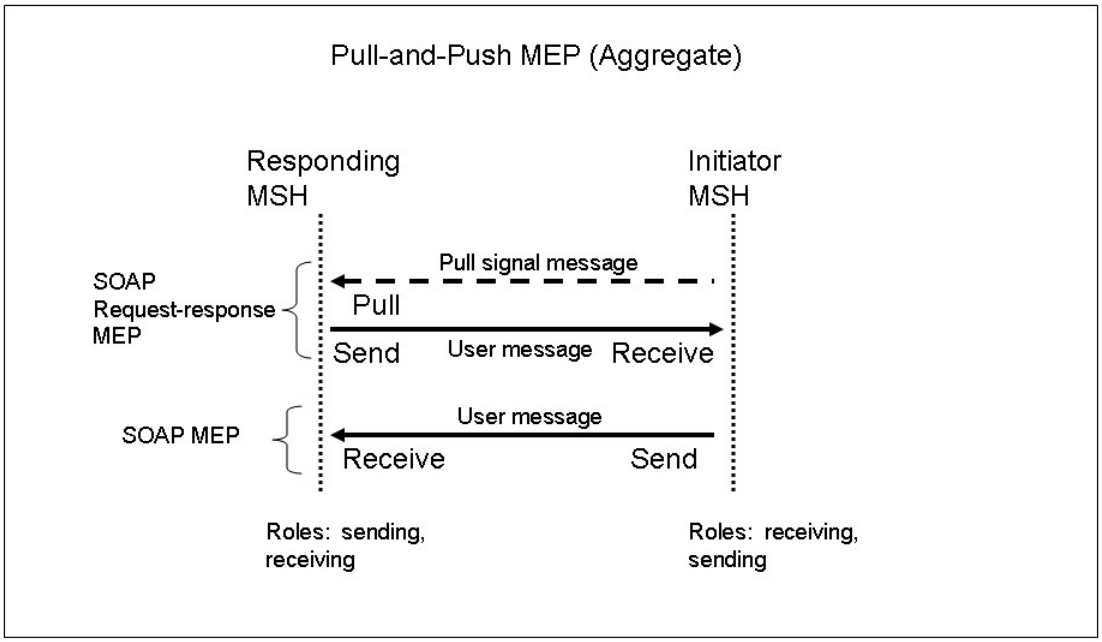
Figure MM6. Aggregate Push-and-Pull MEP.



1.4.3 The Pull-and-Push Message Exchange Pattern

This MEP involves two ebMS user messages. It is composed of a One-Way Pull MEP followed by a One-Way Push MEP. The user message in the second simple MEP refers to the message ID of the user message in the first simple MEP. This MEP can be used to correlate asynchronously a response message with a request message, with all exchanges initiated by the same MSH (here, the receiver of the request). This may accommodate situations where an MSH does not allow for incoming connections, e.g. due to firewall restrictions. Figure MM7 illustrates the exchange pattern and operations involved for this MEP.

Figure MM7. Aggregate Pull-and-Push MEP.


Pull-and-Push MEP (Aggregate)
Responding MSH
Initiator MSH
SOAP Request-response MEP
Pull signal message
Pull
Send
User message
Receive
SOAP MEP
User message
Receive
Send
Roles: sending, receiving
Roles: receiving, sending