Proposal: Multi-hop for ebMS V3 (V0.30)

April 15, 2009~~March 19, 2009~~

Editor(s):
Jacques D. / Sander F. / Pim v.d. E.

Abstract:
Proposal draft for Multi-hop section..

Status:
Draft for discussion.

1 Multihop Messaging

1.1 Background

The core specification of ebMS version 3.0 defines how the message exchange between two parties takes place when they do communicate directly point-to-point. A much found situation when several organizations exchange messages with each other however is the use of intermediaries which are responsible for the message delivery and which may provide additional services. The intermediaries are in charge of routing functions that make it possible for communicating parties to ignore the destination details of their messages (e.g. the URL of the ultimate MSH).

1.2 Terminology

The following definitions are used throughout this section:

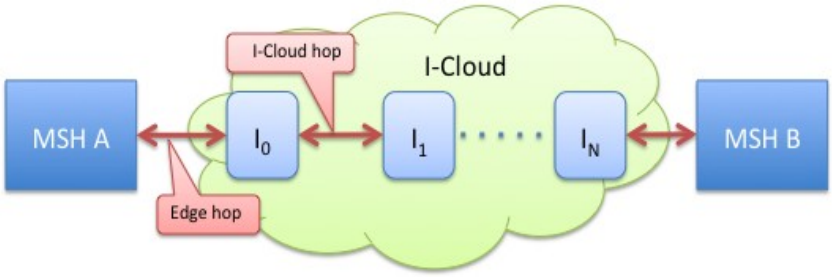
Forwarding role: The ebMS V3 Core specification previously defined two roles in which a MSH can act: “Sending” and “Receiving”. In the messaging model extended to multi-hop, an ebMS V3 MSH can also act in a new role called “**Forwarding**”. In short, a MSH acting in the Forwarding role is able to forward a received ebMS message based on its ebms header content to another MSH without modifying the message, i.e. without altering any header attribute. This new role is defined in section 19.

Intermediary MSH (or ebMS Intermediary): An MSH acting in the new Forwarding role and configured for doing so for at least some messages, in a network of MSHs. ebMS Intermediaries support routing functions that determine the MSH a message should be forwarded to. The routing functions should be based on the meta data provided by the ebMS message header. The requirements imposed on ebMS intermediaries are detailed in section 17.

Endpoint MSH: An MSH that is able to act either in the Sending role or in the Receiving role, and that is configured for doing so for at least some messages, in a network of MSHs. The ability to act as a Sender or Receiver in a multi-hop transfer imposes certain requirements on the MSH which are detailed in section 25.

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

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



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

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

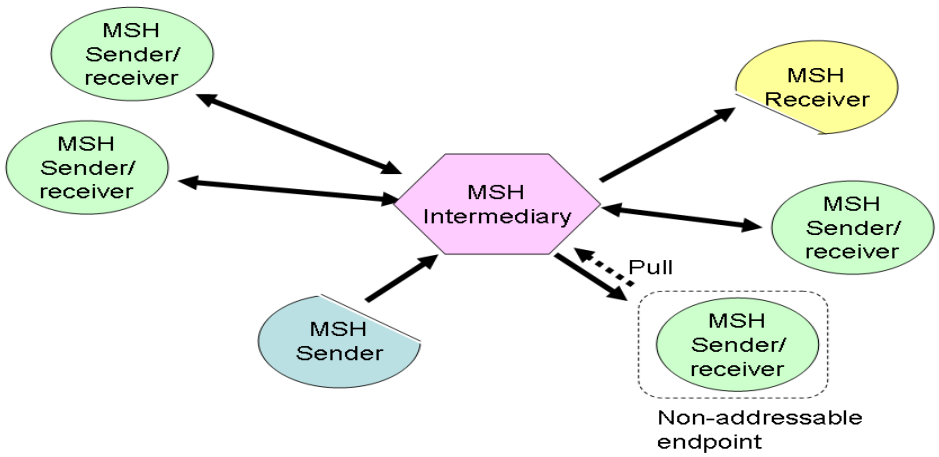
The hops that relate the Endpoint MSHs to the I-Cloud (i.e. hop: MSH A - Intermediary 0, and hop Intermediary N – MSH B) are called **Edge-hops**, while the hops over the I-Cloud are called **I-Cloud hops**. The ebMS Intermediaries that participate in the Edge-hops (I0 and IN in the Figure) are called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

1.3 Multi-hop Topologies

1.3.1 Hub-and-Spoke

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

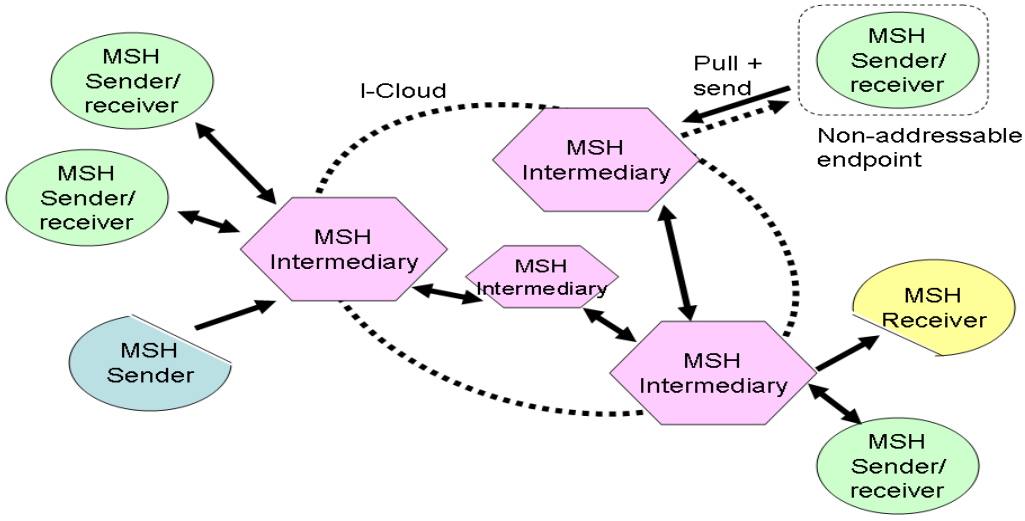
The **Hub-and-spoke** model



1.3.2 Interconnected Hubs

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

The **Interconnected Hubs** model



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

1.3.3 Bridged Domains

In this topology several private ebMS sub-networks are related by Intermediaries called Gateways. Indeed, the internal addresses within an I-Cloud can have private IP addresses and DNS names that are not publicly reachable or resolvable in the Internet.

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

1.3.4 Assumptions

The following assumptions are made about ebMS Intermediaries, that ~~restrict~~supports the multi-hop model described here in a way that is considered acceptable for the large majority of situations:

- ~~The multi-hop mode considered here – reiterated in 1.4.1 (principle 1) – if of **transparent** multi-hop: the core assumption is that ebMS Intermediaries do NOT modify in any way the messages they are forwarding.~~
- The topologies considered here all involve ebMS intermediaries, not exclusive of other non-ebMS nodes. Other nodes (SOAP nodes, HTTP proxies, etc.) may be involved in transferring ebMS messages over multi-hop paths, but they are not considered as ebMS intermediaries in the sense that they are not required to understand any of the ebMS metadata available in the headers and are not supposed to behave depending on this data. Their presence is orthogonal to the definition of ebMS multi-hop topologies.
- An ebMS Intermediary is able to support pulling (i.e. to process a received PullRequest) at least over Edge-hops, i.e. from Endpoint MSHs.
- The same MSH may play different roles for different multi-hop paths: it can be an Intermediary for some messages, a destination Endpoint for others, and an origin Endpoint for others. The multi-hop model described here must support this, although in practice many topologies will restrict the roles that an MSH can play. For simplicity we will assume that in a Hub-and-Spoke model as well as in the Interconnected-Hubs model, the Endpoints are not acting as Intermediaries.

1.4 Usage Requirements

1.4.1 Operation Principles

The following principles are overarching to the design and operation of ebMS multihop messaging:

Principle 1: The I-Cloud does not modify the ebms message units. In other words, it does not modify the eb:Messaging header nor the related message payloads - SOAP Body or

attachments s ~~(“transparent” multi-hop). It does not add or remove SOAP headers. Its intermediaries have to parse and understand some (ebMS) headers for routing purpose.~~

Principle 2: The message transfer over a multihop path, is controlled by two disjoint entities in multihop messaging:

(a)PMode for controlling the communication over edge-hops (origin Endpoint to ICloud, or I-Cloud to destination Endpoint).

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

1.4.2 Multihop modes

Two multihop modes are supported by this specification:

(1) **Transparent multihop**. In this mode, the SOAP message itself is never modified by the I-Cloud: no SOAP header is added or removed over the multi-hop path. This does not prevent intermediaries from parsing and understanding some of these headers, primarily for routing purpose.

(2) **Serviceable multihop**. In this mode, some intermediaries may be configured for adding or removing SOAP headers related to quality of service: reliable messaging and security.

The SOAP role associated with these headers MUST have a value that is same as or extend the following URI:

```
http://www.w3.org/2003/05/soap-envelope/role/ebint"
```

Serviceable multihop may be used in the "bridged domains" multihop topology, where some intermediaries act as gateways to private domains, and handle some security and reliability functions required by traffic over the public area of the I-Cloud but not over the private area.

Both modes comply with the two principles defined in the previous section (Operation Principles).

The rest of this document is describing multihop mostly in the case of transparent mode. Most of it however applies to the serviceable mode as well in a straightforward way.

1.4.1 Connectivity and Addressability constraints

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

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

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

1.4.2 QoS of Exchanges

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

It must be possible to configure a multi-hop topology so that end-to-end reliable transfer of a message is possible, i.e. over a single reliable messaging sequence.

When message forwarding does not involve pulling, and when there is no other connectivity impediment, an Intermediary must be able to use streaming to forward a message without having to persist any part of it.

(Also: WSI Conformance where feasible - especially with respect to WSI RSP policy that WS-RX Reliable Messaging headers be signed, including WS-Addressing elements where used within ReliableMessaging)

1.4.3 Intermediary Configurability and Change management

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

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

(Spoke configuration to services within an I-Cloud should have simplicity of configuration change management.

Specifically, rerouting to services (within the I-Cloud) can be accomplished without spoke configuration changes”)

1.4.4 Compliance with the SOAP Intermediary Model

It is assumed that ebMS Intermediaries are also SOAP intermediaries. By default all nodes in the multihop path act in the “next” role (http://www.w3.org/2003/05/soap-envelope/role/next). In the SOAP processing model for intermediaries, any header that is understood and processed must be removed from the SOAP message. If one wants the headers to be available to further

SOAP processors on the message path the headers have to be “reinserted” in the SOAP message. When a SOAP intermediairy does not process a header it may relay this header to the next SOAP node, I.e. not remove the header from the message.

As the the routing function of the ebMS Intermediairy requires processing of SOAP headers these headers are

not subject to “relay” but to “reinsertion” (SOAP1.2), The relaying function does not apply to the ebms headers, and the @relay=”true” attribute needs not be set. The ebms routing function is then a “header reinsertion” case.

NOTE: because of security issues, it is strongly RECOMMENDED that intermediaries do not alter in any way the SOAP header of routed messages, i.e. implementations must consider the “reinsert” operation as a virtual extraction and reinsertion of header blocs. An intermediary implementation SHOULD NOT rebuild the SOAP header of routed messages by actually extracting then reinserting header blocks.

NOTE: set the @role of all headers to “next”, as that does not seem to be a default in SOAP1.2 ?

1.5 Message Exchange Patterns

1.5.1 MEPs and Channel Bindings

Section 2.2 of the Core V3 Specification defines the notion of ebMS message exchange patterns. These MEPs represent how messages are exchanged between partners. The Core Specification defines two MEPs,

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

Also the concept of MEP Bindings is introduced in section 2.2 of the Core Specification. Such an MEP binding defines how the abstract MEP is bound to the underlying transport.

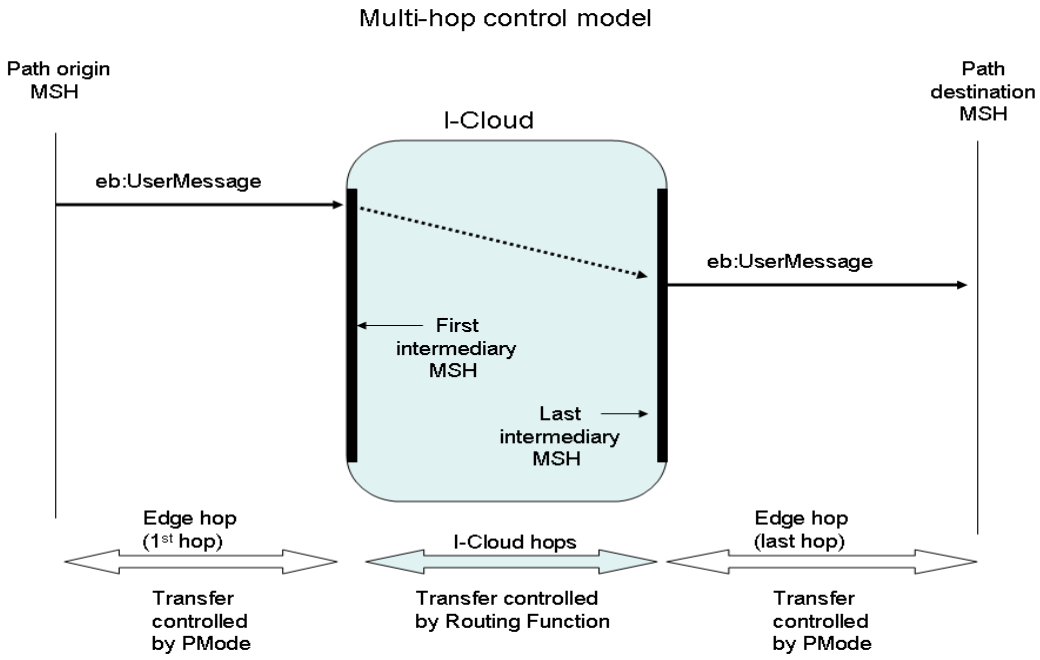
The above MEPs between two partners are independent from the network topology as the MEP represents the exchange pattern between the (application-level) Producer and Consumer of the message. Therefore two partners evolving from a point-to-point topology toward a multihop topology would still use the same message exchanges patterns (One-Way, Two-Way) as defined in the Core specification (V3)..

The way these MEPs bind to the underlying transport protocol does change however, as the transfer is now divided into multiple hops. This implies that the binding of MEPs to the underlying transport (“channel binding”, see 2.2.3 in Core V3) may vary in a way that is not covered by the Core specification.

Message transfer over a multihop path (including the way the underlying transport protocol is used) is controlled by two different means depending on the hop of the path:

- For the **edge hops** the transfer is primarily controlled by the PMode deployed on the endpoint MSHs;
- Transfers within the I-Cloud, i.e. on the **I-Cloud hops** are controlled by the routing function deployed on each Intermediary. The routing function of intermediaries is defined in section 1.6

The following figure illustrates the control of multihop transfers and the related partitioning of multihop paths,



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

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

The following subsections describe multihop MEP bindings while making abstraction of the channel binding inside the I-Cloud.

These multihop edge-bindings will not be defined by new names or URI values for the PMode.MEPbinding parameter. Instead as explained in sub-sections 1.5.2 and 1.5.3, they will result from the combination of point-to-point MEP bindings for both edge hops. In other words, while the same PMode was deployed by each endpoint to control a point-to-point exchange, slightly different PModes on each endpoint may be deployed for controlling the same exchange

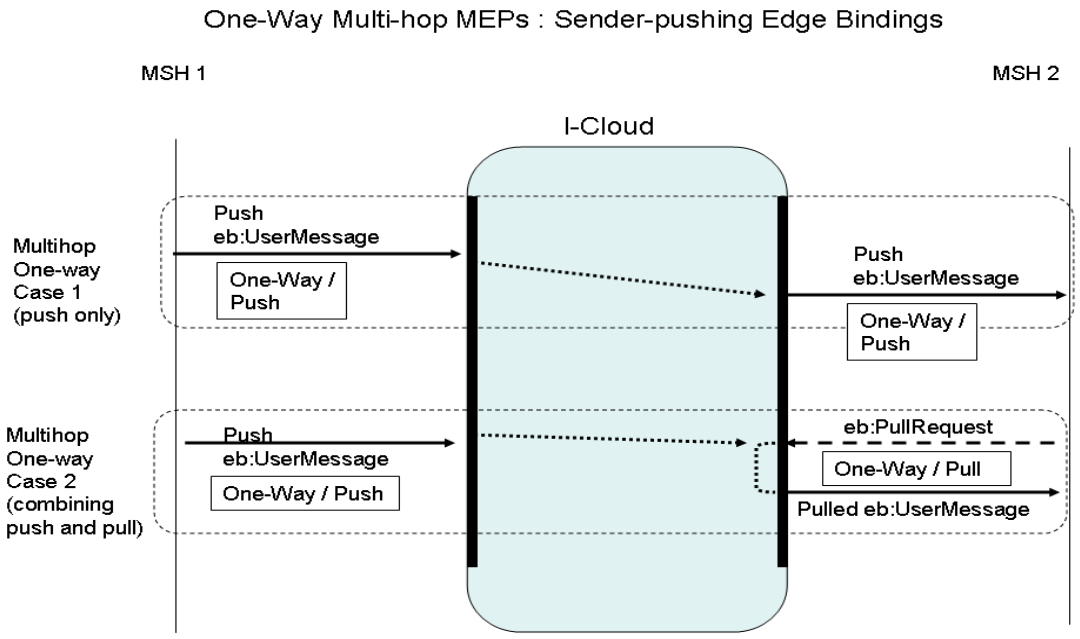
over a multihop path. Section 1.8 will describe these changes, one of them being the PMode.MEPbinding parameter value which may now differ on both ends, as the first and last hops (edge hops) may be channel-bound quite differently over a multihop path.

1.5.2 Edge-bindings for Multi-hop One-Way

NOTE: in the following, the path origin MSH is also called the Sender, and the path destination MSH, the Receiver.

1.5.2.1 Pushing Messages from the Sender Endpoint

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



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

P-Mode MEP configuration:

- Edge hop Sender side:
  - Sender as PMode.Initiator

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

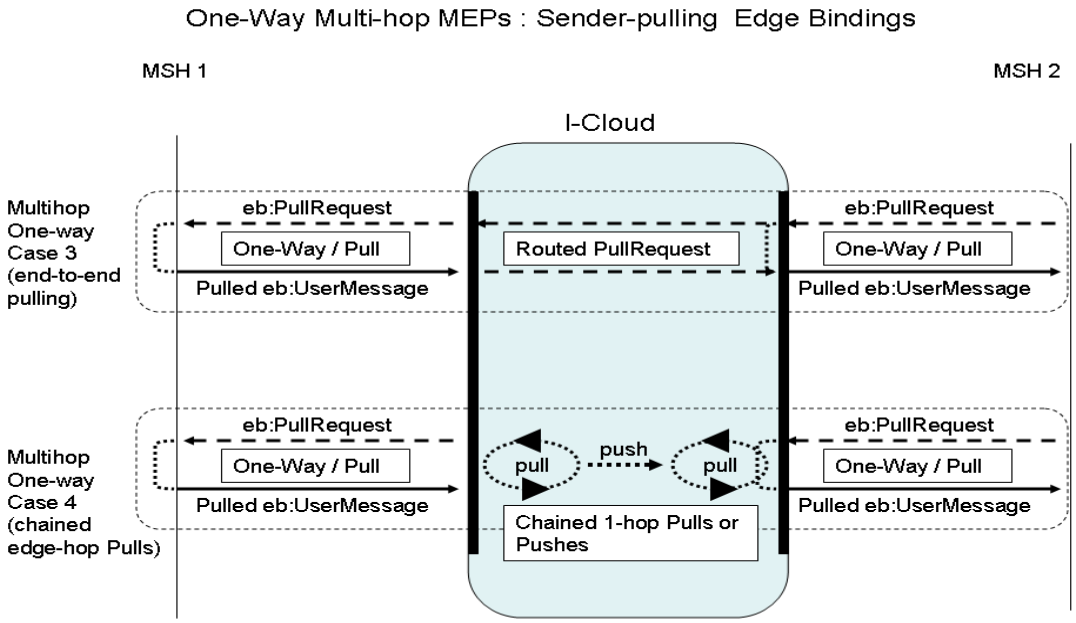
Case 2: This edge-binding will be used when both Sender and Receiver endpoints are not addressable, or when these Endpoints are not willing to receive incoming requests.

P-Mode MEP configuration:

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

1.5.2.2 Pulling Messages from the Sender Endpoint

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



In both Case 3 and Case 4 above, the edge bindings are quite similar: all of them are One-way / Pull. However, the difference lies in the way the pulling is propagated across the I-Cloud.

Case 3: In this MEP binding, the PullRequest signal is generated by the Receiver endpoint MSH, and routed all the way by the I-Cloud as any other message, to the Sender MSH. In other words, the same PullRequest message is used in both edge-hops. The implication of such multi-hop pulling, is that each Intermediary on the path must keep its transport connections open. The main advantage is that there is a single authorization point for the pulling: the Sender MSH to which the Pull signal is intended. The I-Cloud has no responsibility in authorizing the pulling and has no authorization information (passwords or certificates) to maintain. The routing function will decide of the behavior of the Intermediary: if there is an entry for the routing of PullRequest messages intended for a particular MPC, then the Intermediary must hold its connection open until it gets a response for this PullRequest. This is the behavior that supports Case 3. If there is no routing entry, then the PullRequest is intended to the Intermediary itself, and the pulled message is one that has already been posted (or forwarded) to this Intermediary for this MPC. The latter is supporting Case 4.

P-Mode MEP configuration:

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

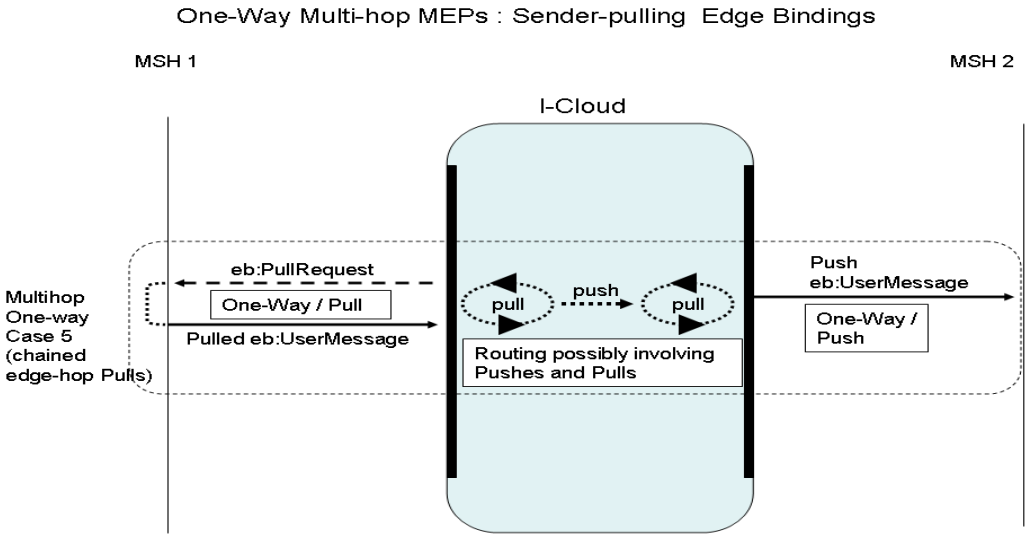
- MEP and binding = One-way / Pull

Case 4: In this MEP binding, the PullRequest signal is not routed, and only goes over a single hop. The PullRequest signals over each edge-hop are different messages, which could have different authorization credentials (the Pull signal is authorized for the next node only). These edge pulls are relayed by the I-Cloud in unspecified ways – the pulled message could be pushed and/or pulled across the I-Cloud. The MPC that is pulled from is the only common point between these decoupled pulls, called chained pulls. The Intermediary configuration for pulling some MPCs while pushing to other MPCs is part of the routing function, so the difference between Case 4 and Case 3 is not much in the P-Mode configuration, but rather in the I-Cloud routing function.

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

The difference between Case 3 and Case 4 is not apparent in P-Mode parameters: it is only apparent in how the routing function is defined – i.e. whether or not it is configured for routing PullRequest signals. However, there is a difference in endpoint behavior in the way reliable messaging is supported, which will be controlled by a new P-Mode parameter: Pmode[1].Reliability.AtleastOnce.ReliablePull

(see section 1.9 “Details on Reliable Multihop”)



Case 5: In this MEP binding, the first edge-hop is pulled, while the last edge-hop is pushed. These edge pulls are relayed by the I-Cloud in unspecified ways – the pulled message could be pushed and/or pulled across the I-Cloud. The difference between Case 5 and Case 4 is in the last edge-hop binding.

P-Mode MEP configuration:

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

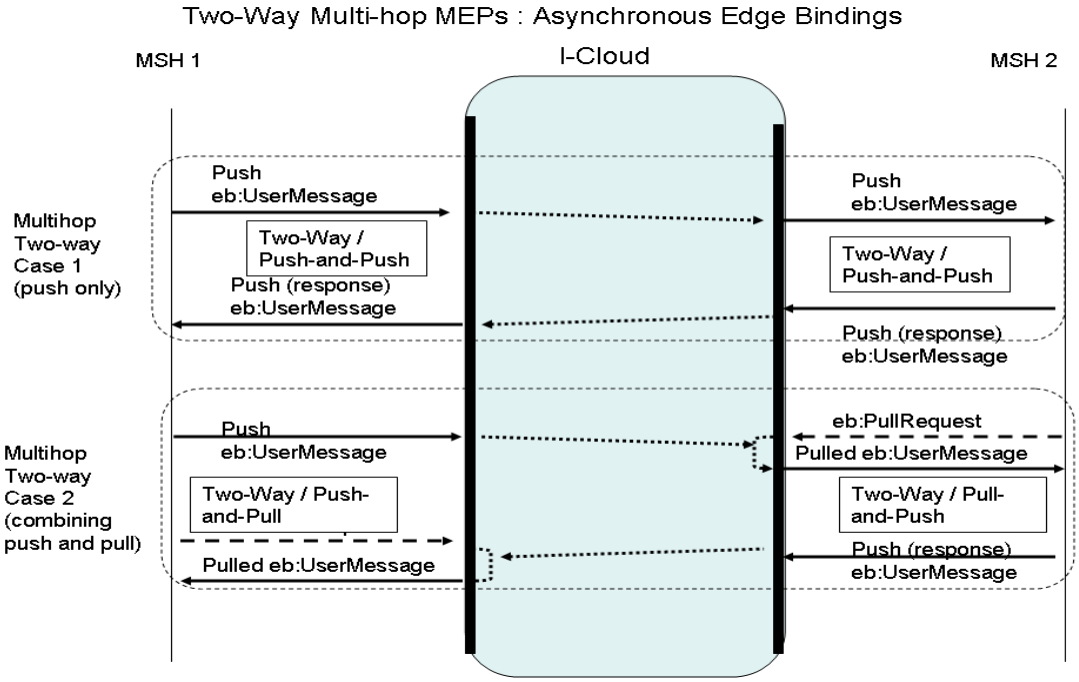
Other combinations of edge-bindings are expected to be used and supported by Intermediaries that are not described here although they will automatically be supported by Intermediaries that already support the edge-binding combinations specified here.

1.5.3 Edge-bindings for Multi-hop Two-Way

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

1.5.3.1 Asynchronous Edge-bindings

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



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

P-Mode MEP configuration:

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

Case 2: This case applies when both endpoint MSHs are not addressable: both are pulling messages from the I-Cloud.

P-Mode MEP configuration:

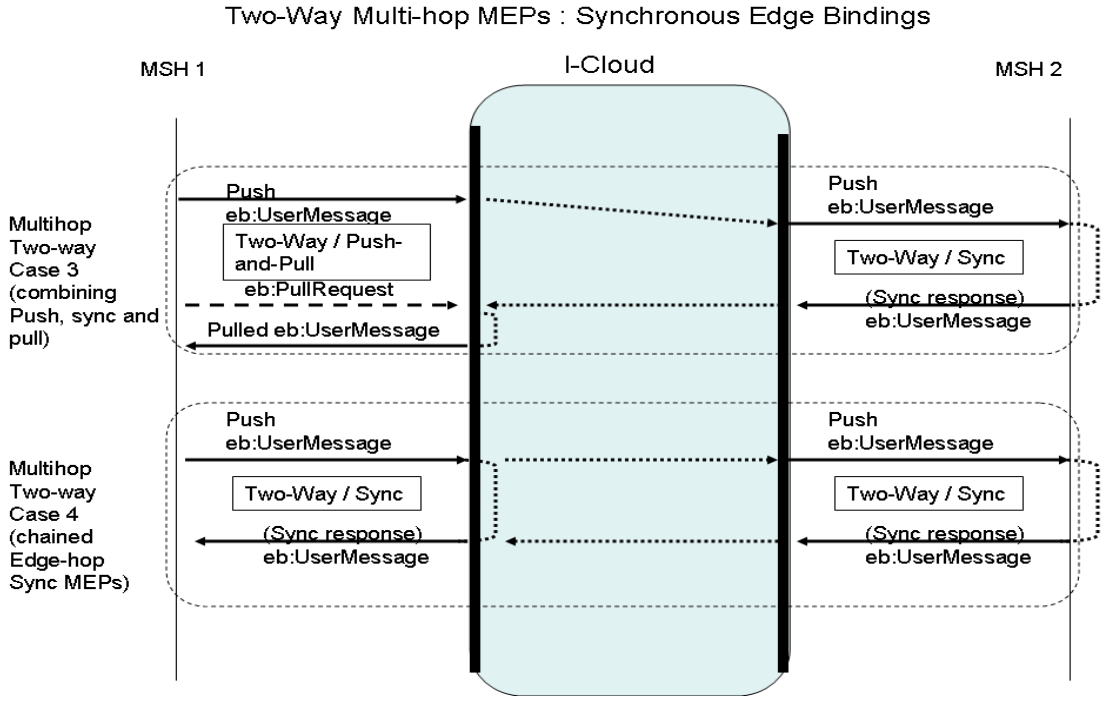
- Edge hops on Request Sender side:
  - Request Sender as PMode.Initiator (Intermediary as PMode.Responder)
  - MEP and binding = Two-way / Push-and-Pull
- Edge hops on Request Receiver side:

- Request Receiver as PMode.Responder (Intermediary as PMode.Initiator)
- MEP and binding = Two-way / Pull-and-Push

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

1.5.3.2 Synchronous Edge-bindings

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



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

P-Mode MEP configuration:

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

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

P-Mode MEP configuration:

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

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

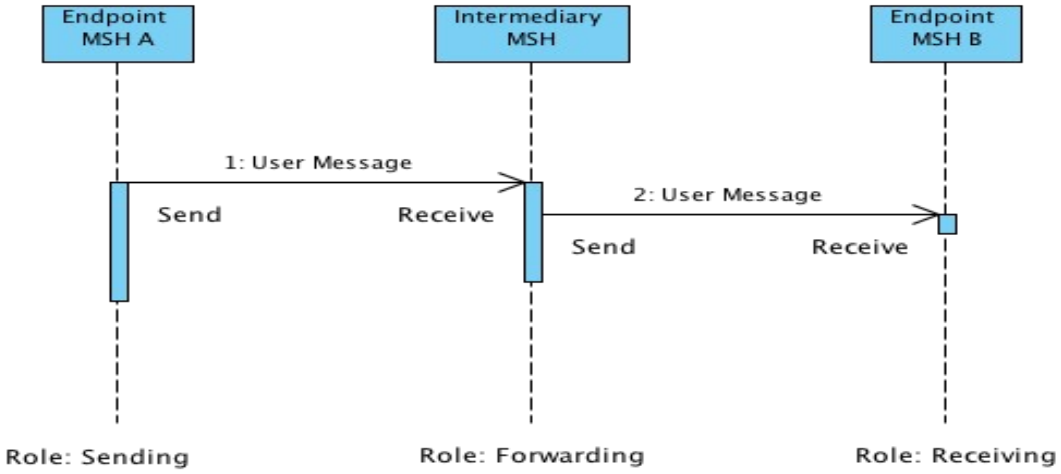
1.6 The Intermediary MSH

1.6.1 Intermediary Functions

An ebMS intermediary is expected to support the following functions:

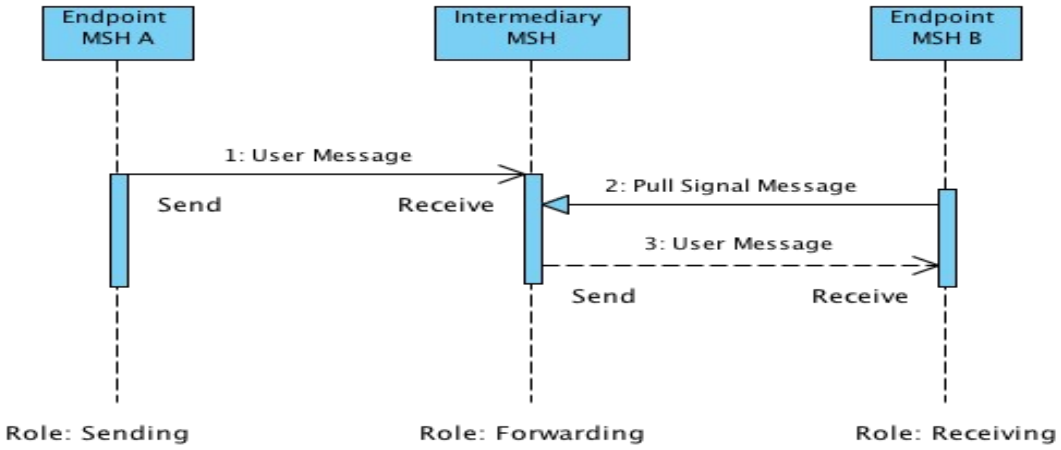
Message forwarding. Besides the Send and Receive operations defined in the model of V3 Core specification, a new operation called Forward is added to the messaging model. It is defined as: sending (operation Send) a message that has been received (operation Receive) without altering it, and without external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

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



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

MEP bridging. Forwarding also implies the ability to bridge different MEP channel bindings, although in a limited way (see Section …). For example, a message bound to a One-way / Push MEP when received, may be bound to a One-way / Pull after forwarding, as illustrated in the following Figure..



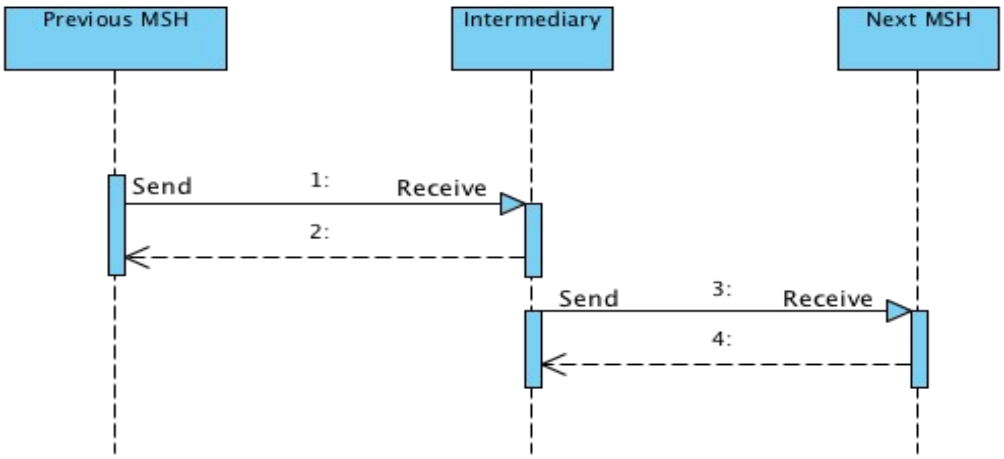
Message Routing. As an intermediary is not an endpoint of the multi-hop path, every message received must be forwarded to another MSH. The routing function of an intermediary defines to which MSH a message must be forwarded based on metadata carried by the received message.

Message Pulling support. This implies some partial knowledge of PModes that determine a push vs. a pull channel. In the case of pulling, some scalability feature (sub-channels) will also be supported and configured with authorization info. In addition to assigning (a) eb:UserMessages to Pull channels, an Intermediary MUST also be able to assign (b) eb:SignalMessages and (c) non-ebMS, RM signals such as wsrm:CreateSequence or Sequence Acknowledgments, so that these signals become available for pulling by an endpoint MSH. In case (c), the pulled signal MUST be combined with an eb:Error element with code: EBMS:0006 (EmptyMessagePartitionChannel). Consequently, the response to a Pull request is always an ebMS message.

Error Handling. An Intermediary must be able to generate at least two eb:Errors: (1) EBMS:0006 (EmptyMessagePartitionChannel), when getting PullRequest for MPCs that configured for pulling and that are empty of any User Message, (2) EBMS:0020 (RoutingFailure), when the routing function cannot perform as expected on a particular message (see next section).

1.6.2 Message Forwarding Models

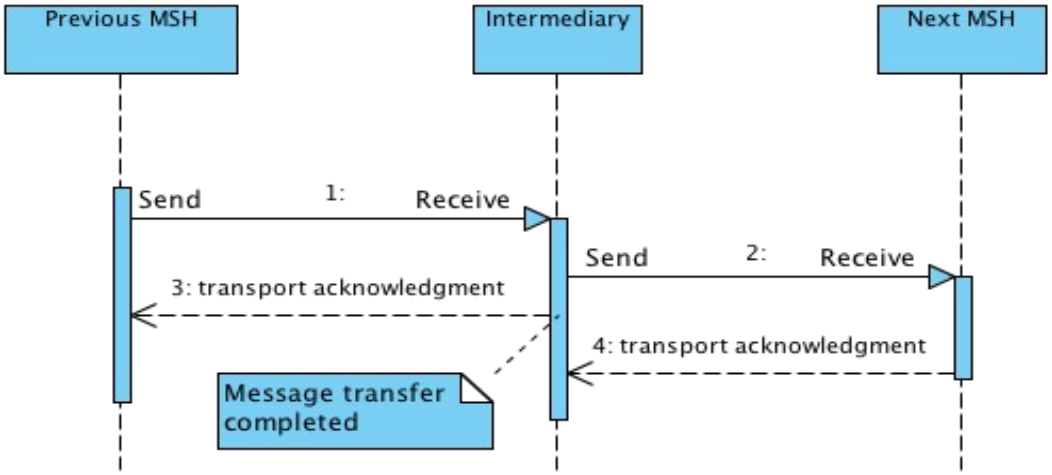
For the new abstract operation Forward two implementation models exist. First is the store-and-forward model in which the Send operation only starts after the Receive operation has successfully been completed. So the Send and Receive are executed sequentially and the successful execution of the Receive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



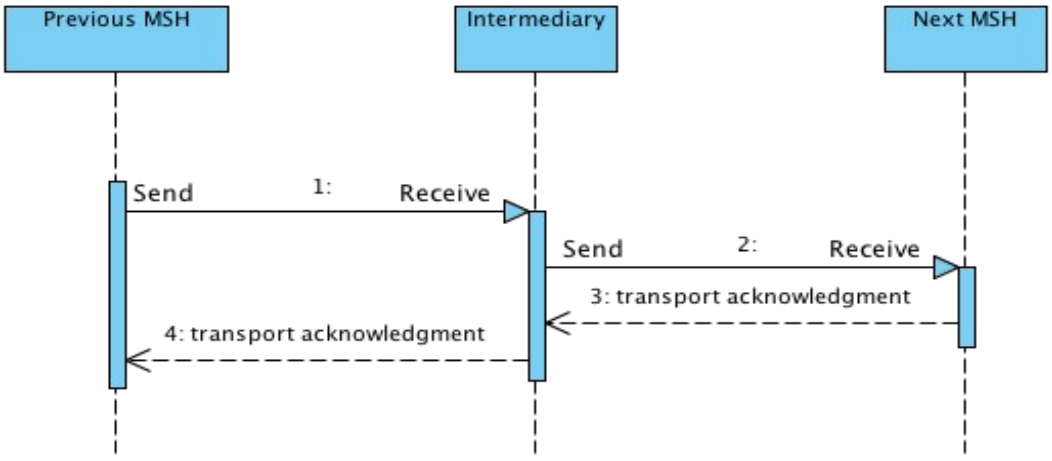
Opposite to this model is the streaming model in which both operations are executed in parallel, i.e. data is directly transferred to the next hop. In this model the Send operation starts as soon as possible after the Receive operation does. Because the next hop will only be known after the

message header has been received the Send operation will always start some time after the start of the Receive operation.

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



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



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

Advantage of both streaming models over the store-and-forward model is that is requires less storage space because there’s no need to store complete messages. Also the end-to-end latency is shorter.

The main disadvantage of this model is that it requires more bandwidth as the Receive operation only ends after the corresponding Send finishes, so incoming connections stay open until the outgoing one is closed. If the outgoing connection doesn't have at least the same bandwidth as the incoming connection the streaming model will lead to congestion on the intermediary. Therefore the streaming model should only be used when bandwidth is known to be available.

1.6.3 MEP Bridging

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

- **Forwarding pattern (a) (or "push-push"):** Message pushed to the Internediary over MPC x, and forwarded in push mode by this same Intermediary to the next node.
- **Forwarding pattern (b)(or "push-pull"):** Message pushed to the Internediary over MPC x, and forwarded in pull mode, i.e. the message will be pulled from MPC x in the Intermediary by the next node MSH (Intermediary or endpoint).
- **Forwarding pattern (c)(or "pull-push"):** Message pulled by the Internediary from MPC x, and forwarded in push mode by this same Intermediary to the next node.
- **Forwarding pattern (d)(or "pull-pull"):** Message pulled by the Internediary from MPC x, and forwarded in pull mode i.e. the message will be pulled from MPC x in the Intermediary by the next node,

This specification does not define which ones of these forwarding patterns must be supported by an intermediary: this is to be determined by future conformance profiles.

The routing function MUST specify which forwarding pattern is in use for each MPC it handles. It also includes authorization credentials associated with pulling – either inbound or outbound – when some forwarding pattern involves pulling.

1.6.4 Sub-channels

An intermediary - and in particular an edge intermediary - may split the flow of user messages to be forwarded, so that these messages are forwarded to different destinations even if they are sent over the same Message Partition Channel (i.e. have same `eb:Messaging/eb:UserMessage/@mpc` value).

An example of such a use case is:

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

Sub-channels are a means to support the previous use case. A sub-channel is associated with an existing channel or "root" MPC, and plays the role of a new MPC in case of a "pull" forwarding, with its specific authorization data distinct from other related sub-channels. Forwarding by an intermediary is only possible over the same MPC as the one over which the message was received, or over a sub-channel from the MPC over which the message was received,

The identifier of a sub-channel is an extension of the identifier of the related MPC. If the MPC identifier is an URI of the form:

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

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

```
http://sender.example.com/mpc123?id=(unique string)
```

where {unique-String} MUST be replaced by a string that is unique for this root MPC, so that every sub-channel of this MPC can be associated with different authorization means. It is RECOMMENDED to use a UUID value as defined by RFC4122 [UUID].

The forwarding patterns (b) (push-pull) and (c) (pull-pull) may then be used to forward a message from an MPC over a sub-channel of this MPC. This behavior is determined by the routing function associated with the intermediary. The original MPC value in the message (e.g. `http://sender.example.com/mpc123`) is is not altered: the message is still considered as sent over this MPC. However, when the routing function of the intermediary has forwarded a message to a sub-channel of its original MPC (i.e. assigned it to a sub-channel MPC for subsequent pulling) then pulling this message can only be done by pulling from this sub-channel, and not by pulling from its root MPC.

Form the receiver viewpoint (in the above use case, the ultimate recipient), sub-channels are used exactly in the same way as channels or MPCs: sub-channel identifiers are to be used in P-Modes in the same way as other MPC identifiers. A PullRequest message for a sub-channel will specify the sub-channel ID in its @mpc value - though the pulled message will still have the root MPC identifier in its @mpc value.

1.6.5 The Routing Function

A function every intermediary MUST implement is the routing function that defines to which MSH a received message is forwarded. The input to this routing function, called the **routing input**, is a set of meta data elements taken from the received message.

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

The routing function can be modeled as:

F(RoutingInput) → either a URL or a pull channel assignment

The RoutingInput data is a subset of the header content of ebMS User Messages, more precisely the **eb:UserMessage** element. The routing function of an Intermediary MUST be able to parse and use the following meta data as its routing input:

- The mpc attribute, when present.
- The eb:PartyInfo element and its subelements.
- The eb:CollaborationInfo element and its subelements

The routing function SHOULD be able to parse and use:

- The eb:MessageProperties element and its subelements,
- The eb:PayloadInfo element and its subelements

An Intermediary MAY parse and use the eb:MessageInfo element.

The RoutingInput data may also be extracted from a PulRequest signal, in case the routing function must route such signals (as in 1.5.2.2, Case 3 – “end-to-end pulling”). In that case, and in the absence of any other header usable for routing (i.e. RoutingInput reference parameter header) the routing input MUST be reduced to the MPC attribute eb:PullRequest/@mpc and the routing function MUST be able to parse the eb:SignalMessage element, in order to access this MPC value.

When the routing input (or Effective routing input, as defined later) is reduced for some messages to the sole MPC value – or in other words, when the I-Cloud routing function uses only the MPC value (@mpc) for some routings – then each one of these MPC values resolved by the routing function MUST NOT be shared by different destination endpoints.

NOTE: an Intermediary MUST NOT fault a message with an eb:UserMessage element that does not contain and eb:MessageInfo element, and MUST consider such eb:UserMessage as a valid input for the routing function.

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

(a)ebMS signal messages which are not also ebMS user messages;

(b)non-ebMS messages that are involved in facilitating the transfer and quality of service of ebMS messages (such as various signals supporting reliable messaging).

Such messages however do not contain an eb:UserMessage element with the input needed for the routing function. In order to provide the routing function with proper routing input these messages MUST contain a WS-Addressing endpoint reference parameter that includes the routing input – except for PullRequest signals in case the routing can be done based on MPC alone. This reference parameter is defined as an element named **RoutingInput** and which

contains exactly one child **UserMessage** element. For non ebMS user messages like (a) and (b) above the SOAP header must contain the following WS-A reference parameter:

```
<ebint:RoutingInput wsa:IsReferenceParameter='true'>
                  <ebint:UserMessage>...</ebint:UserMessage>
</ebint:routinginput>
```

The embedded ebint:UserMessage element is conforming to the XML schema defined for the similar element in the packaging section of Core V3, except for the eb:MessageInfo element which can be absent here.

NOTE: The UserMessage element in the reference parameter is in a different namespace than the UserMessage element from the Core V3 specification because it allows for the MessageInfo element to be absent and therefore must be redefined for use in the reference parameter.

An ebMS Intermediary MUST be able to obtain the routing input from a message in two ways:

(1) By parsing the eb:Messaging header in an ebMS user message, and by extracting its eb:UserMessage child element.

(2) By parsing the wsa reference parameter header ebint:RoutingInput and by extracting its ebint:UserMessage child element.

It is possible that an ebMS user message also contains a WS-Addressing reference parameter in the SOAP header. In such a case where both the ebint:RoutingInput and the eb:Messaging/eb:UserMessage elements are present in the same message, the intermediary MUST use the WS-Addressing reference parameter as input for the routing function.

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

A routing function will generally use only a subset of the eb:UserMessage element or of the ebint:RoutingInput element. This subset is called the **Effective** routing input, as opposed to the **Available** routing input represented by the entire eb:UserMessage element or the ebint:RoutingInput element.

1.6.6 Handling of Routing Failures

ebMS Errors (eb:Error) generated by endpoints are subject to the same reporting options as errors in point-to-point communication. AT least a couple of routing patterns for eb:Error must be supported: (a) the error destination URL is known (e.g. obtained from the PMode) and does not require any multihop routing, or (b) the eb:Error signal is routed using the same routing functions as those used for User messages.

An ebMS Errors may be generated by Intermediaries. A new type of error must be supported by intermediaries:

- error ID: EBMS:0020,
- short description: RoutingFailure
- severity: failure
- description: whenever an Intermediary is unable to route an ebMS message, it must generate such an error and stop processing the message.

The reporting of the error must follow one of these three patterns: (a) the error is logged locally to the Intermediary, for later analysis, (b) the error is sent to a fixed destination provided to the Intermediary at configuration time, (c) the error is reported to an explicit URL present in a wsa header (wsa:ReplyTo or wsa:FaultTo) of the message in error.

1.7 Endpoints requirements

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

For ebMS user messages this poses no additional requirements on endpoints: the routing input is included in the eb:Messaging SOAP header block. For other messages (ebMS Signal messages, or non-ebMS messages) the endpoint MUST insert one or more WS-Addressing headers and use these to carry routing input. The following requirements must be observed by Endpoints.

1.7.1 Usage and Interpretation of WS-Addressing EPRs and Message Headers

1.7.1.1 Endpoint References

WS-Addressing EndPoint References (EPR) are used in this specification to identify endpoint MSHs. However, part of the rationale of ebMS multihop topologies is that he Internet address (URL) of destination endpoints is either not resolvable directly by the original sender, or even not to be published (e.g. they may change overtime, or be meaningful only for a local network). Consequently both the wsa:Address and wsa:ReferenceParameters values of the EPR may be used by the routing function, when no other routing input is available in the message header.

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

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

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

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

This EPR indicates the need to rely exclusively on the reference parameter eb:RoutingInput to route the messages intended to such an endpoint. The “icloud” URI must then appear in the wsa:To header of any message intended to this destination Endpoint, along with the <ebint:RoutingInput> header block that represents the reference parameter. The minimum set of wsa headers that must be present in a message intended to an Endpoint described by the above EPR, is illustrated below:

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

1.7.1.2 Use of wsa Headers

Reliance on WS-Addressing headers other than those supporting the routing function (i.e. headers containing reference parameters) is not required for ebMS multihop processing, However, in order to comply with the SOAP binding of message addressing properties defined in [REC-ws-addr-soap-20060509.htm] (http://www.w3.org/TR/ws-addr-soap) when a reference parameter header block is present – indicating a destination endpoint described by an EPR - at minimum the wsa:To and wsa:Action headers must be present, with values corresponding to those in the EPR definition.

About wsa:Action:

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

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

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

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

In case the message contains an `eb:Receipt` for an ebMS user message, it is RECOMMENDED that the `wsa:Action` value has a similar value based on whether the whether the received message is One

Way message or the request or a response in a Two Way message exchange, again with "`.receipt`" as suffix:

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

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

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

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

Note that the wsa:Action attribute MUST be an IRI as defined in RFC3987 and has basically the same structure as an URL. I think this can be a problem when constructing the Action value from P-Mode property values as the concatenation might not form a valid IRI.

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

1.7.2 Routing Support for Initiating Messages

"Initiating messages" are here defined as messages that are not responding to previous messages.

1. Case of ebMS User Messages: Such messages are either the "request" leg of an ebMS Two-way or the "oneway" leg of an ebMS One-way. The eb:UserMessage header element contains all routing input. The content of these headers is determined jointly by the PMode configuration of the Sending endpoint and/or by the submitting message Producer.

2. Case of ebMS Signal Messages: The main case of "initiating" ebMS signal, is the eb:PullRequest message (see Case 4 of On-way edge-bindings in section …). Some eb:Errors may be generated as initiating messages, i.e. not as responses to faulty messages. The ebms header of such messages does not contain an eb:UserMessage element. However,

the PMode associated with the messages to be pulled (i.e. defining a One-way / Pull MEP) is shared between partners, along with the EPR of the sender of the pulled message. From this EPR is extracted the eb:routinginput reference parameter, added as WS-Addressing header to the PullRequest signal.

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

1.7.3 Routing Support for Response Messages

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

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

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

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

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

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

2. Case of ebMS Signal Messages: These are either eb:Errors or eb:Receipts sent in response of a previously received ebMS message. The ebms header of such messages does not contain an eb:UserMessage element. However, either one of these conditions MUST be met and decided per agreement between communicating parties:

(b1)the wsa:ReplyTo header was present in the request message, and if its EPR value contains the ebint:routinginput element. In that case (and unless the EPR also contains a reachable URI – see Section 1.7.3) this element is used to generate a corresponding wsa header that will be

used by the I-Cloud routing function. For eb:Error, the wsa:FaultTo if present must take precedence over wsa:ReplyTo.

(b2)the PMode associated with the request message is shared between partners, and the sender of the response message extracts the EPR of the initial sender from this PMode (even if it has an anonymous URI). It inserts the eb:routinginput reference parameter obtained from this EPR in the header of the response message. This header will be used by the I-Cloud routing function.

3. Case of non-ebMS messages: Such messages do not contain any eb:Messaging header (unless they are piggybacked on ebms Messages in which case the routing rules for these messages apply). The same rules as for above case (2) apply: the routing is based on WS-Addressign reference parameter header ebint:routinginput (unless the EPR also contains a reachable URI – see Section 1.7.3) An exception is made for RM protocol messages (such as Acknowledgements, or sequence management messages): In that case – and assuming that end-to-end reliable messaging is used - the EPR of the request sender MUST be specified in the wsrm:AcksTo element provided when requesting or accepting an RM sequence creation. Either one of the following conditions MUST be met:

(c1)This EPR contains the eb:routinginput element suitable for routing across the I-Cloud back to the initial message sender.

(c2)The wsa:Address element in this EPR contains a URI identifying the request message sender so that it can be used as routing input by Intermediaries or used to directly reach the Internet destination, bypassing the I-Cloud and not relying on any ebMS-related routing function.

(c3)This EPR has an wsa:Address element that contains a URI identifying the Intermediary used by the request message sender, as well as some MPC parameter, so that the message can be pulled by the request sender on this MPC, piggybacked on a EmptyMessagePartitionChannel warning (EBMS:0006).

4. Inferred RoutingInput for the reverse path:

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

-The eb:From and eb:To values should be swapped.

-The eb:Service is the same, but eb:Action is appended with “.receipt” (in case an eb:Receipt is sent back) or “.error” (in case an eb:Error is sent back).

-In case an MPC is mentioned, derive a new MPC Id by concatenating “.receipt” or “.error”. E.g. on the first hop, a One-way / Push is sent to MPC “abc123”. The eb:Receipt is automatically routed back to MPC “abc123.receipt. Similarly for a related error: it would be sent back on the MPC “abc123.error” Inferring a new MPC for these response allows the original Sender to automatically know which MPC it can then pull for these Receipts and Errors, in case this Sender is not addressable for Callbacks.

1.8 PModes for Multihop

1.8.1 PMode Extension for EPRs and Routing Input

When it is necessary to insert reference parameters in a message, a Sending endpoint needs to get this information from the PMode. The PMode governing the sending messages SHOULD specify whether a reference parameter must be added and what should be included in the parameter. This requires an extension of data model for PModes as presented in the core specification. A new PMode composed parameter named EPR is introduced to specify the WS-Addressing EPR value. It contains two main sub-elements:

[PMode...].EPR.Address
[PMode...].EPR.RoutingInput

The **EPR.Address** element indicates the address URI associated with the endpoint. In a multihop context where the routing function is used, its value is expected to be: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/icloud. (When sending a message to such an EPR, the wsa header wsa:To must reflect this value.)

The **EPR.RoutingInput** element represents the value of the RoutingInput reference parameter associated with this EPR. This composed parameter includes a subset of parameters found in the User Message header:

[PMode...].EPR.RoutingInput.Initiator.Party

[PMode...]. EPR.RoutingInput. Initiator.Role

[PMode...]. EPR.RoutingInput.Responder.Party

[PMode...]. EPR.RoutingInput. Responder.Role

[PMode...]. EPR.RoutingInput.BusinessInfo.Service

[PMode...]. EPR.RoutingInput.BusinessInfo.Action

[PMode...]. EPR.RoutingInput.BusinessInfo.Properties[]

[PMode...]. EPR.RoutingInput.BusinessInfo.MPC

Notes:

-Not every element has to be present under the RoutingInput parameter node, as long as the remaining elements are sufficient for routing.

-RoutingInput.Initiator.{ Party, Role} and RoutingInput.Responder.{ Party, Role} when present, must have same values as respectively PMode.Initiator.{ Party, Role} and PMode.Responder. { Party, Role}.

-RoutingInput.BusinessInfo concerns the destination endpoint. There could be several such elements (meaning several sets {Service, Action, Properties, MPC}) associated with each endpoint, as reflected by various PModes associated with this endpoint. Or, it is possible to select just one RoutingInput.BusinessInfo value to be used for routing to this endpoint, even if the endpoint supports several pairs Service/Action, or several MPCs. For example, even if an

endpoint supports several Service values, it is possible that only one such value is known of the I-Cloud routing function. In that case, all PModes associated with this endpoint will use this same value even though each one of them may use a different Service / Action pair in its PMode. BusinessInfo parameter. In such cases the RoutingInput reference parameter must always be used even for routing User Messages, as the routing function would be unable to use various Service / Action pairs supported by this endpoint.

This EPR element may be repeated at different places in the PMode, where it is necessary to indicate a multihop destination.

In particular, it may appear in the PMode as follows:

PMode.Initiator.EPR

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

Note: If both the EPR.Address PMode sub-parameter and the PMode[].Protocol.Address are present, the latter is taking precedence. Hence if routing must be used to send messages to the Initiator, the PMode[].Protocol.Address must not be set so that EPR.Address is used.

NOTE: in case this EPR parameter only contains the Address element and no RoutingInput element, the Address value MUST either (1) be sufficient to directly address the endpoint (meaning “response” messages sent by the Responder are not routed via the I-Cloud) , or (2) be sufficient for the routing function to resolve the multihop path leading to the Initiator endpoint (using the wsa:To header containing this address).

PMode.Responder.EPR

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

Note: If both the EPR.Address PMode sub-parameter and the PMode[].Protocol.Address are present, the latter is taking precedence. Hence if routing must be used to send messages to the Initiator, the PMode[].Protocol.Address must not be set so that EPR.Address is used.

PMode[1].Errorhandling.Report.SenderErrorsTo.EPR

To specify the destination – including the routing reference parameter value - to be used for routing any Error generated by the UserMessage-sending endpoint for this PMode leg. The default is the EPR associated with the Sending MSH for this PMode leg (either PMode.Initiator.EPR or PMode.Responder.EPR)

PMode[1].Errorhandling.Report.ReceiverErrorsTo. EPR

To specify the destination – including the routing reference parameter value - to be used for routing any Error generated by the UserMessage-receiving endpoint for this PMode leg. The default is the EPR associated with the Receiving MSH for this PMode leg (either PMode.Initiator.EPR or PMode.Responder.EPR)

PMode[1].Security.SendReceipt.EPR

To specify the destination – including the routing reference parameter value - to be used for routing any eb:Receipt generated in response to a User Message send over this PMode leg. If not present, Receipts will be routed using the EPR associated with the UserMessage-sending endpoint (either PMode.Initiator.EPR or PMode.Responder.EPR)

NOTE: when PMode[1].Reliability.AtLeastOnce.Contract is set to “true” then Acknowledgements as well as other RM signals sent by the RMD MUST be sent back to the Sending party for this PMode leg. In case the PMode[1].Reliability.AtLeastOnce.Contract.AcksTo parameter is set, then its value MUST either be of an EPR associated with the RMS of the sending endpoint (this EPR could contain an address that does not need be resolved by the I-Cloud routing function, meaning these RM signals are directly sent back to the MSH without multihop routing), or simply contain a URI as described in ebMS V3 Appendix D.3.5., that resolves to the RMS associated with the sending endpoint, in which case these signals will be directly addressed to this RMS without multihop routing.

If PMode[1].Reliability.AtLeastOnce.Contract.AcksTo is not present, then RM signals will be routed using the EPR associated with the endpoint sending messages for this RM sequence (either PMode.Initiator.EPR or PMode.Responder.EPR)

The use of the EPR.RoutingInput extension will generally be required for non-ebMS messages (e.g. RM protocol messages) and may be require for eb signals (eb:Receipt, eb:Error) although the reverse routing of these messages may rely on information provided in the WSA header of the related request message (wsa:ReplyTo) as described in Appendix [].

1.8.2 Migrating From Point-to-Point to Multihop

An endpoint migrating to a multihop exchange may have to modify its behavior compared with a point-to-point exchange. For example, depending on the multihop channel binding (see section 1.5) it may have to pull eb:Receipt messages related to pushed User messages, whereas in a point-to-point exchange the eb:Receipt message could be received over the same connection as the User message (e.g. on the HTTP response).

Another example is, when using reliable messaging (WS-RM) the routing of RM signal messages such as CreateSequence, requires the addition of reference parameters (conforming to WS-Addressing) in the SOAP header.

Often, when two partners who were communicating point-to-point need to migrate to a multi-hop configuration, it is expected they will derive a PMode for multihop from the existing PModes that govern their point-to-point exchanges (security, QoS, Error reporting…).

The PMode governing a multihop MEP is actually divided in two parts or “units”, each one of these is actually structured as a separate PMode:

- The PMode unit that governs the Initiator edge-hop (linking the Initiator endpoint to the first ebMS Intermediary on a multihop path originating in the Initiator). The ID of this PMode (PMode.ID) must be of the form: “<ID>.init”, where <ID> is a valid value for PMode.ID. This unit is called the “**init**” PMode for the multihop MEP.

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

In order to minimize the impact over an already deployed endpoint MSH when upgrading a point-to-point exchange into a multihop exchange, the existing PMode model as defined in ebMS core V3, does not need to extend to new MEP binding values in order to implement the new multihop bindings (edge bindings) defined in 1.5.

The following rules will apply to automatically extend to a multi-hop semantics the PMode parameters values specified in Core V3 for PMode.MEP and PMode.MEPbinding:

- (Rule 1) Whereas these parameters were meant to define the exchange between an Initiator and a Responder endpoints in a point-to-point context, they are meant to define the exchange between the **Initiator** MSH and the **first Intermediary** MSH in an “init” PMode unit

- (Rule 2) Whereas these parameters were meant to define the exchange between an Initiator and a Responder endpoints in a point-to-point context, they are meant to define the exchange between the **last Intermediary** MSH and the **Responder** MSH in a “resp” PMode unit.

- (Rule 3) These PMode parameters have no effect on the mode of transfer (either pulling or pushing) over each hop of the portion of the multi-hop path inside the I-Cloud, as this is governed by the routing function which also dictates the mode of transfer associated with an MPC, separately for inbound and outbound traffic over this MPC.

The above rules ensure that a pair of point-to-point partners can migrate to a multi-hop context with minimal changes or no changes in their PMode configuration.

1.8.3 Aligning the PMode units deployed on Initiator and Responder Endpoints

As the two edge hops of a same multihop path used by an MEP instance are controlled by different PMode units, these two PMode units MUST be aligned for proper interoperability.

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

PMode parameters that MUST be common between two related PMode units:

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

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

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

1.9 Details of Reliable Multihop

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

As required in Core ebMS V3 specification, an endpoint MSH MUST be able to control which reliable (RM) sequence is used by the User Message it sends, if only because different reliability QoS may be associated with different reliable sequences. This includes the ability to initiate RM sequences as required by the PModes of messages to be sent, and to associate such sequences with any further message sent for this PMode. The main difference between a multi-hop environment and a peer-to-peer environment, is the need to add sufficient routing input to RM Lifecycle Messages (CreateSequence, CloseSequence, TerminateSequence, Acknowledgements) in a multi-hop environment. This routing input is obtained from References Parameters associated with the destination EPR (of which the Address element is generally not supportive of a direct addressing mechanism, justifying the use of routing). Such a Reference Parameter value SHOULD be specified in the EPR.RoutingInput PMode parameter associated with the destination.

The following rules apply when creating reliable sequences for multihop:

•An end-to-end reliable (RM) sequence is always associated with a unique destination MSH, although it MAY be used for several types of message exchanges governed by different PModes, if all these messages require the same level of reliability QoS (see the PMode parameters: AtLeastOnce, AtMostOnce, ExactlyOnce). It is however RECOMMENDED that all messages sent over the same sequence, are sent over the same MPC.

•When a CreateSequence message contains a RoutingInput reference parameter (as a SOAP header block), and if the MPC value (@mpc attribute) is specified (`soap:Header/ebint:RoutingInput/eb:UserMessage/@mpc`) then all User messages sent reliably over this sequence MUST be intended for the same MPC, i.e. have same `ebint:UserMessage/@mpc` value.

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

The PMode parameter **PMode[1].Reliability.Correlation** can help enforce the assignment of all messages with same Effective routing input to the same RM sequence. For example, if the routing is determined by the following Effective routing input:

•eb:CollaborationInfo/eb:Service

•eb:CollaborationInfo/eb:Action

•eb:PartyInfo/eb:To/eb:PartyId

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

PMode[1].Reliability.Correlation ="ebint:UserMessage/ eb:CollaborationInfo/eb:Service, ebint:UserMessage/eb:CollaborationInfo/eb:Action, ebint:UserMessage/eb:PartyInfo/eb:To/eb:PartyId"

The response to RM lifecycle messages (CreateSequence, CloseSequence, TerminateSequence, AckRequested).is routed according to cases 3 or 4 in section 1.7.3 (Routing support for response Messages) .

Reliable message pulling:

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

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

1.**End-to-end pulling** (as described in section 1.5.2.2, “pulling messages from the Sender endpoint”, Case 3 – “end-to-end pulling”). This case is handled in the same way as reliable point-to-point, from a reliable messaging viewpoint. The PullRequest message itself SHOULD be sent reliably (AtLeastOnce contract) in order to support AtLeastOnce contract for pulled messages, as the resending of PullRequest messages will provide – for some request-response transports such as HTTP – a back-channel for resending pulled messages that may have already been sent. The behavior of sending PullRequest signals reliably is normally assumed when ensuring reliable One-Way / Pull, as in point-to-point.

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

(a)Receiving-side pulling and Sending-side pulling, or one-hop pull for each edge-hop. The case is illustrated in section 1.5.2.2, “pulling messages from the Sender endpoint”, Case 4

(b)Receiving-side pulling and Sending-side pushing. This case is illustrated in Section 1.5.2.1 (Case 2), involving pushing from the Sending endpoint (first edge hop) and Pulling from Receiving endpoint (last edge hop).

In both cases, the routing function inside the I-Cloud may be configured for any combination of pull and push along the multihop path, and any pull is only a one-hop pull (Pull may be relayed, as in the forwarding pattern (d), section 1.6.5.) , For this reason, there is no use for sending PullRequest signals reliably from the initiating endpoint, as the resending MSH is not the one receiving the resent PullRequest. PullRequest signals SHOULD NOT be sent reliably, in spite

of the response expected to be sent reliably. This is controlled by a new P-Mode parameter that overrides the default behavior for One-Way / Pull MEPs:

Pmode[1].Reliability.AtleastOnce.ReliablePull. If “false”, the sending of PullRequest will not be made reliably by the initiating endpoint.

Consideration about reliable messaging specifications:

The Core V3 specification allows for binding with either WS-Reliability or WS-ReliableMessaging standards, in order to ensure reliable messaging. Both provide the same level of reliability and can be used for end-to-end multihop reliability. The following minor differences must be noted:

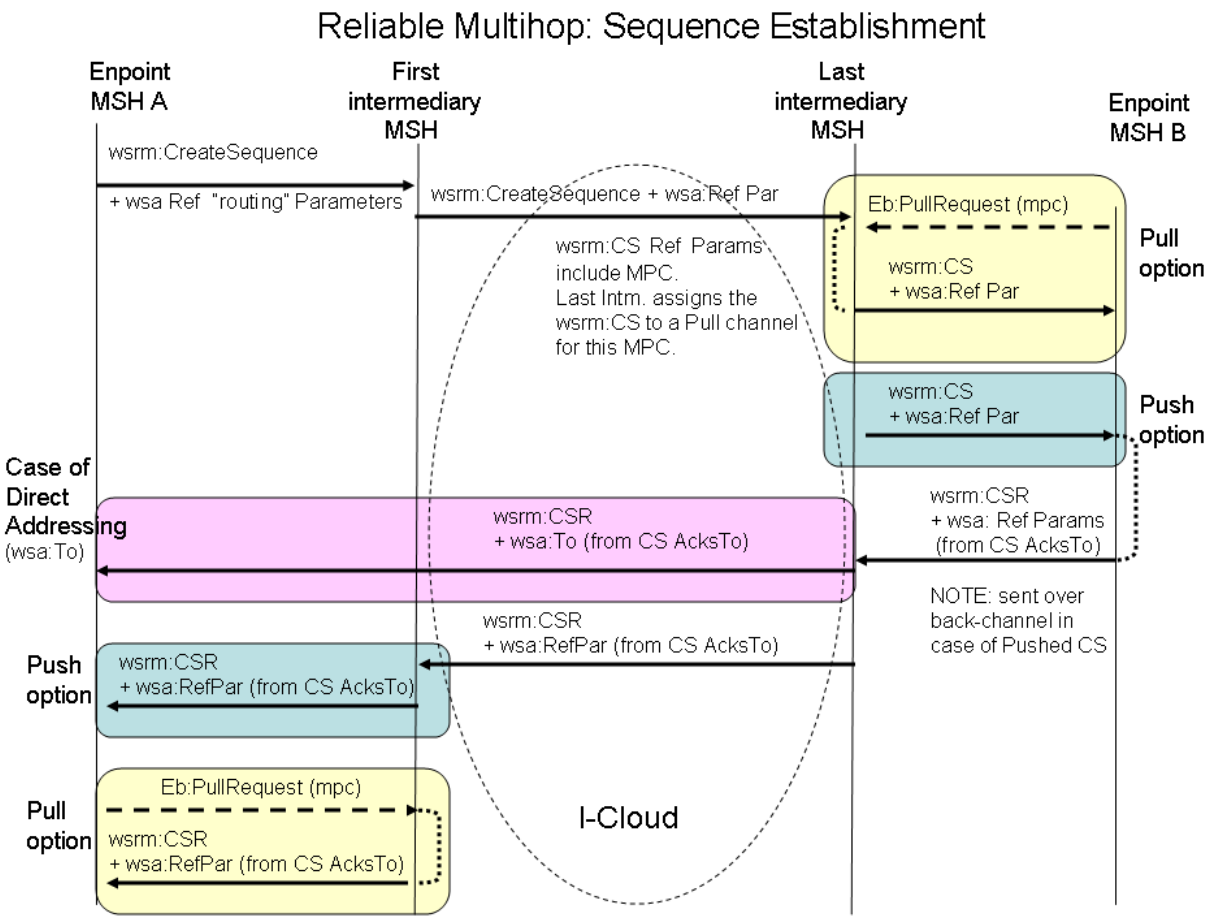
The routing of RM signals (CreateSequence, etc) and the need to add routing input headers to these, are mostly a requirement tied to the use of WS-ReliableMessaging, as WS-Reliability does not use such signals to manage sequences. Initiating and closing a reliable sequence with WS-Reliability requires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

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

1.10 Appendix A: Flow Diagrams for Multi-hop

1.10.1 Reliable Sequence Establishment

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



- **Step 1**: The Sending party submits a User message to its endpoint MSH A. The MSH resolves which PMode / CPA is associated with this message so that it knows its processing mode (which level of security, reliability, MEP…). In this case, the PMode requires that its related messages need be sent reliably, i.e. that an RM sequence needs be initiated for these.
- **Step 2**: The Sending endpoint MSH A determines where to send this message – here to an ebMS Intermediary – by getting the Intermediary URL from its PMode / CPA configuration
- **Step 3**: MSH A is initiating a wsrm:CreateSequence message to the Intermediary (unless an RM sequence already exists for this PMode / CPA). Because the PMode also contains a wsa EPR for the destination, including the Reference Parameter eb:routinginput that replicates a significant subset of the ebMS header data associated with this PMode (data that will be common to all User Messages sent for this PMode), the MSH piggybacks this Reference Parameter on the wsrm:CreateSequence message for routing purpose.
    - NOTE: because the destination URL is unknown (dynamically resolved by I-Cloud routing), the destination EPR is set to the I-Cloud URI: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/icloud, which has no special meaning in case of a SOAP Request. The header wsa:To is present in the wsrm:CS message and contains the I-Cloud URI.

- **Step 4:** The First Intermediary receives the wsrm:CreateSequence message, and forwards it using a routing function that uses ebMS header data (regardless whether this data is wrapped into an ebMS header or into the eb:routinginput wsa Ref Parameters headers). In this case the routing data is in wsa Reference Parameters.
- **Step 5**: At the end of the routing path, the Last Intermediary gets the wsrm:CreateSequence message. Two options must be supported depending on the MEP required by the destination endpoint (MSH B):
  - **Push MEP**: The Last Intermediary keeps routing the wsrm:CS in a push mode to the destination endpoint (MSH B). The content of wsrm:AcksTo will determine how the wsrm:CSR is sent back.
  - **Pull MEP**: The Last Intermediary is aware that the reference parameter associated with the wsrm:CS calls for a Pull mode. Among these parameters, is a mention of the MPC where messages for this sequence will be pulled from. The Intermediary MAY piggyback on the wsrm:CS a dummy eb:Header with a non-effective Service value (`http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service`). The intermediary MUST assign the wsrm:CS message to the related MPC so that the message becomes subject to pulling using eb:PullRequest targeted to this MPC.
    - ***EDITOR NOTE:*** *an alternative could have used wsmc:MakeConnection to pull such signals. However, the use of MC would not be able to use the most standard cases (pulling based on sequence ID, and pulling based on wsa:Address) due to the multihop context. Consequently MC extensibility points would have to be used, to pull based on MPC. Because of this level of customization there is little advantage in using the wsmc standard, and it becomes simpler and more reliable to rely on a unique pulling mechanism – eb:PullRequest – for all messages related to a sequence*.

- **Step 6**: The RM module in MSH B takes knowledge of the wsrm:AcksTo element contained in the wsrm:CS message. The wsrm:AcksTo element value indicates the EPR where the wsrm:CSR must be sent back. Indeed, in the ebMS multihop model, RM lifecycle response messages such as CSR must be sent to the same destination as RM acknowledgment. The wsrm:AcksTo element is itself an EPR that sufficiently identifies the initial Sending MSH so that the I-Cloud can route the wsrm:CSR back to MSH A. The following cases need to be supported, depending on how the wsrm:CS was transmitted:
  - **Pushed wsrm:CS:** The wsrm:CSR is sent back based on the wsrm:AcksTo content. If it were an anonymous URI, the CSR is sent over the backchannel of the wsrm:CS, from the RM module of MSH B to the Last Intermediary. In all cases, the eb:routinginput Reference Parameter, if present in the AcksTo EPR, is added to the wsrm:CSR message. The routing of wsrm:CSR is based on this reference parameter. In a special case where the AcksTo gives the URL of the destination (MSH A) and this destination can directly be resolved without ebMS-level routing, the RM module of MSH B can directly send it back to MSH A.
  - **Pulled wsrm:CS:** The wsrm:CSR is sent over a new HTTP connection to the I-Cloud. The eb:routinginput Reference Parameter in the AcksTo EPR is added to the wsrm:CSR for routing.

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

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

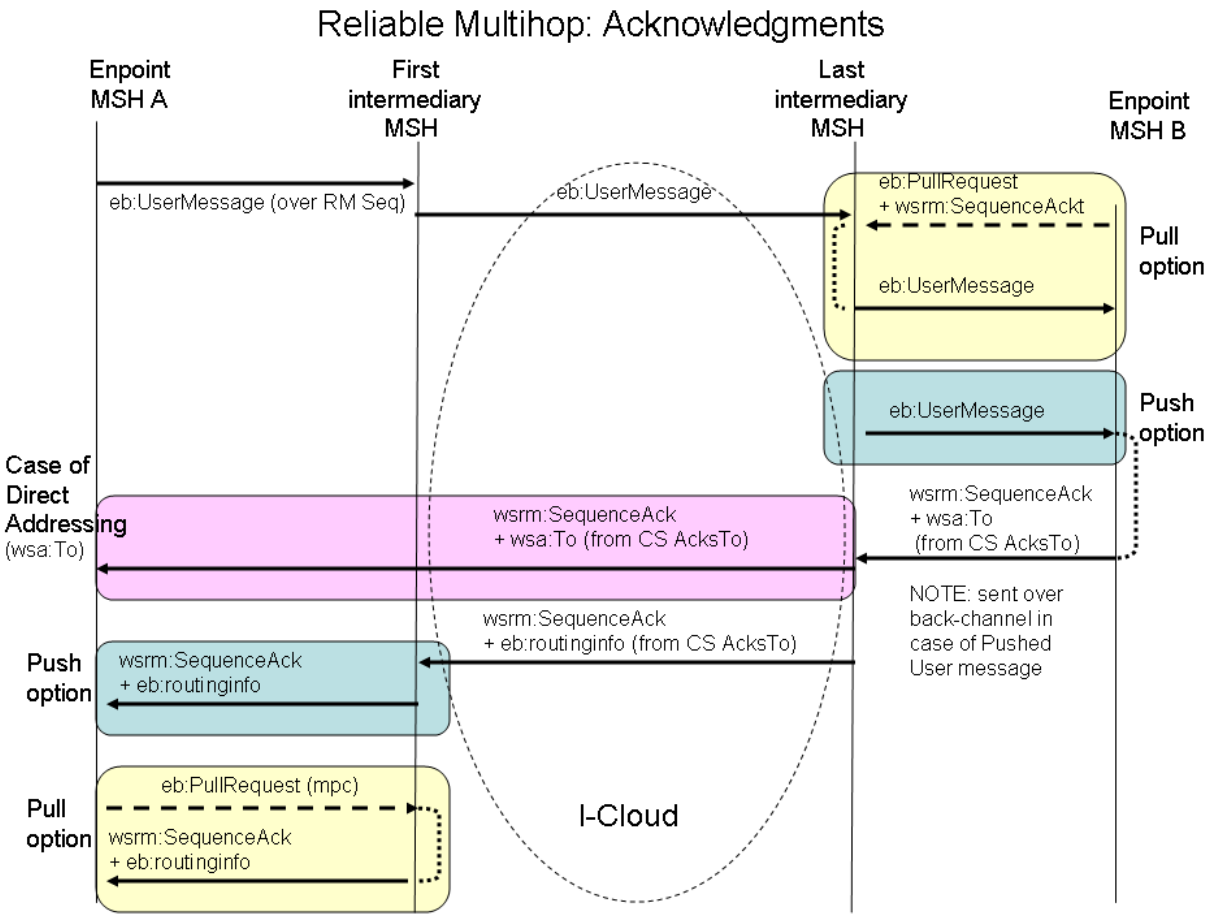
In the above process, it must be noted that:

-the endpoint MSHs have the ability to insert and parse WS-addressing EPRs.

-The ebMS intermediary in contact with the destination endpoint MSH, must act in accordance of the PMode unit that governs its edge-hop, especially in case of message pulling from the endpoint.

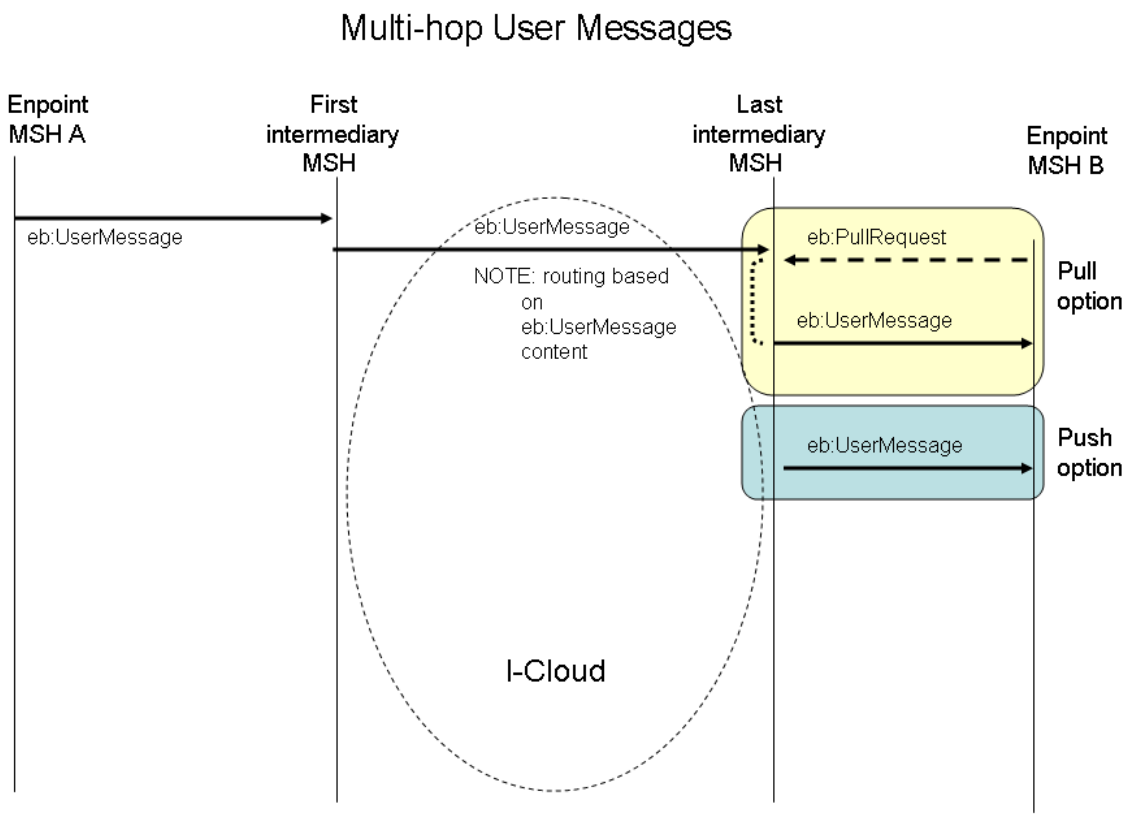
.

1.10.2 Routing RM Acknowledgments



- **Step 1**: The Sending party submits a User message to its endpoint MSH A. The MSH resolves which PMode / CPA is associated with this message so that it knows its processing mode (P-Mode) In this case, the P-Mode is already associated with a Reliable Messaging (RM) sequence.
- **Step 2**: The Sending endpoint MSH A determines where to send this message – here to an ebMS Intermediary
- **Step 3:** The First Intermediary receives the User message, and forwards it using a routing function that uses ebMS header data.
- **Step 4**: At the end of the routing path, the Last Intermediary gets the User message. Two options must be supported depending on the MEP required by the destination endpoint (MSH B):
  - **Push MEP**: The Last Intermediary keeps routing the User message in a push mode to the destination endpoint (MSH B).
  - **Pull MEP**: The Last Intermediary is aware that the P-Mode associated with the User message calls for a Pull mode. The message is assigned to a Pull channel (MPC).
- **Step 5**: The RM module in MSH B sends back a wsrm:SequenceAcknowledgment based on the content of wsrm:AcksTo. It will be routed in the same way as the wsrm:CSR during the sequence establishment. In case where the User message was pulled from the last Intermediary, and if the resolution of wsrm:AcksTo is compatible with using this last Intermediary for routing acknowledgments, then the wsrm:SequenceAcknowledgment header may be piggybacked on the eb:PullRequest message.
- **Step 6**: In case the routing of wsrm:SequenceAcknowledgment involves the initial First Intermediary, the wsrm:SequenceAcknowledgment message is subject to the same Push / Pull alternative as in Step 4. In case of a Pull configuration, the Intermediary will assign the wsrm:SequenceAcknowledgment to a Pull channel. The acknowledgment will be associated with an eb:Error of type warning (EBMS:0006) unless it is piggybacked with another eb message for this channel.

1.10.3 Routing User Messages (One-Way MEP)



1389

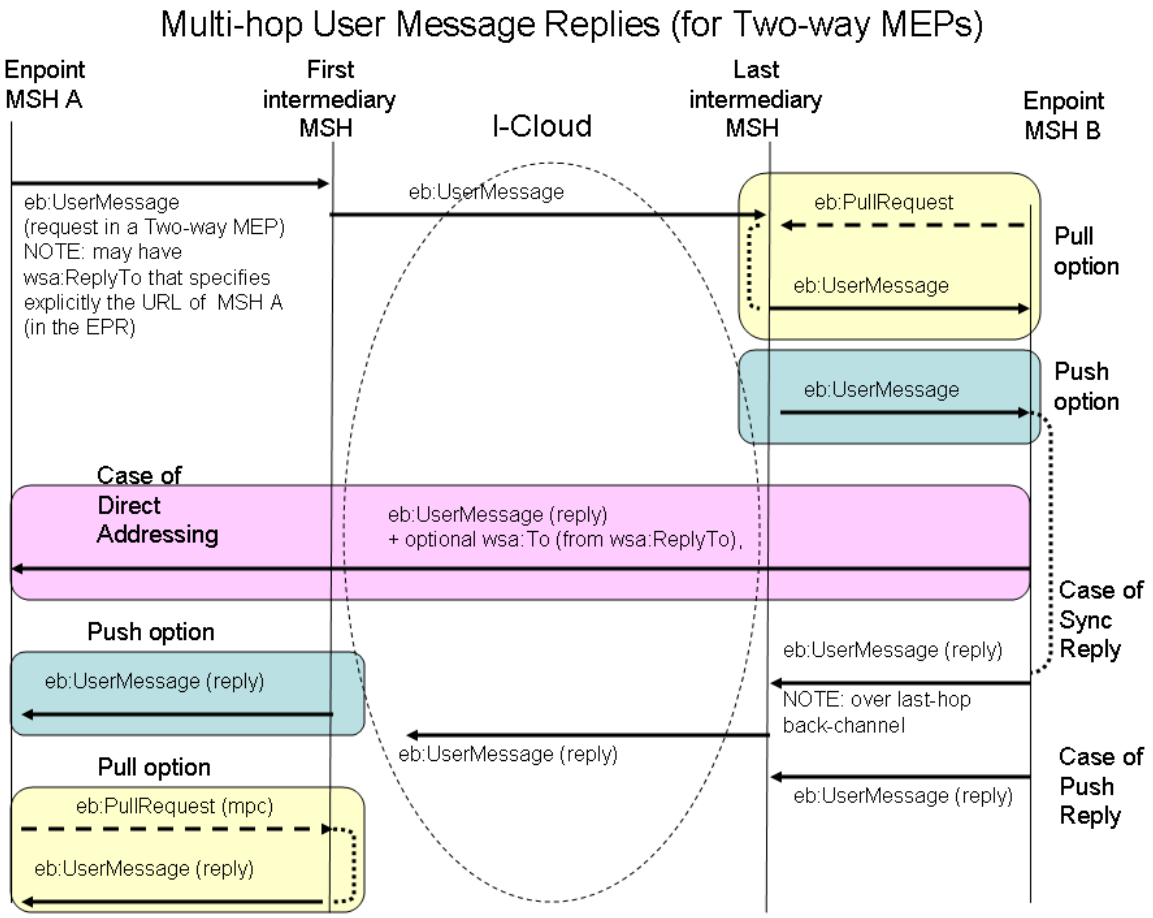
1390 The above case only concerns User messages that are pushed by the Sending MSH. This flow
1391 has been described in sufficient details in the previous sections. Routing is expected to be
1392 based on the UserMessage content. However the presence of a RoutingInput reference
1393 parameter is not excluded, in which case the latter takes precedence.

1394 The routing across the I-Cloud may involve both pushing and pulling. Both push and pull cases
1395 are illustrated on the Receiving side.

1396

1397 1.10.4 Routing ebMS Reply User Messages (Two-way MEP)

1398



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

(1) Direct addressing, by-passing the I-Cloud routing in case the response address is explicitly provided (e.g. by wsa:ReplyTo) and can be resolved without ebMS-level routing.

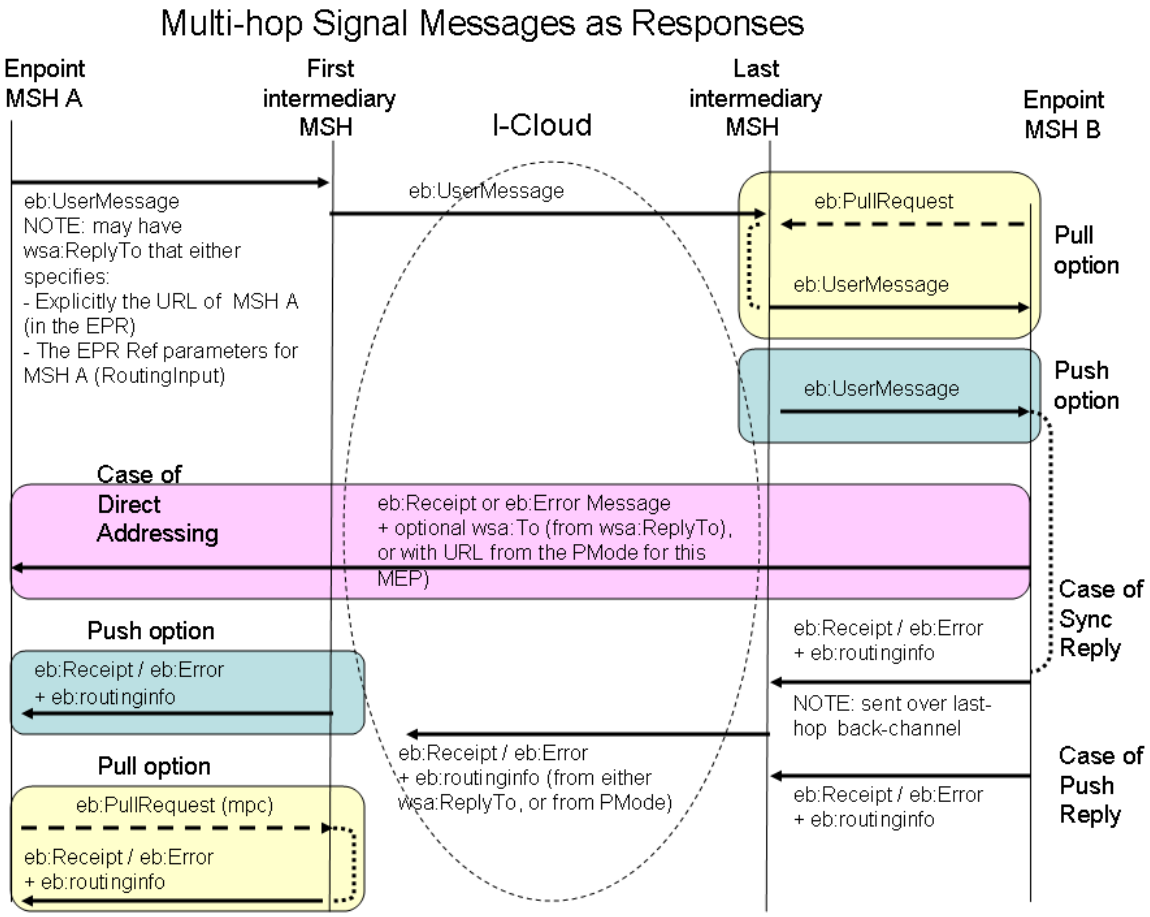
(2) Reply message sent back over the back-channel of the “request” message (case where the P-Mode.MEPbinding is “Sync”, for a request-response underlying transport such as HTTP).

(3) Reply message sent back as callback (case where the P-Mode.MEPbinding is “Push-and-Push”, e.g. over a new HTTP connection).

his flow has been described in sufficient details in the previous sections.

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

1.10.5 Routing ebMS Response Signals (Receipts, Errors)



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

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

1.11 Appendix B: Sample Multihop Message Exchanges

This annex contains some sample message exchanges across intermediaries. They highlight the issues related to intermediaries (such as the use of the `ebint:RoutingInput` and WS-Addressing headers). All example messages contain a `wsse:Security` header, but the timestamp, security token and signature values of the security header are omitted. Digest values may be incorrect due to formatting.

The following example corresponds to the flow diagram in Section 1.10.5 of Appendix A ("Routing of Response Signals"), in the "Push Reply" case.

1.11.1 One-Way Exchange with Asynchronous Receipt

This example includes a One-Way push MEP across a single intermediary, followed by a `Receipt` sent back via the same intermediary, on separate asynchronous HTTP connections. This case is described in draft 0.22 section 1.5.2.1 Multihop One-Way Case 1 (Push Only). The assumption is that both endpoints are addressable from the intermediary. The intermediary pushes all messages to next hop URLs using a routing function based on `eb3:To/eb3:PartyId`.

1.11.2 Message from Buyer to Intermediary

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

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

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

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

<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd">
    <S12:Header>
        <eb3:Messaging S12:mustUnderstand="true" id="id_5cb44655-5720-4cf4-
a772-19cd480b0ad4">
            <eb3:UserMessage mpc="e5c31ef7-d750-4db8-b4dc-13a751d80b9a">
                <eb3:MessageInfo>
                    <eb3:Timestamp>2009-05-21T10:49:28.886Z</eb3:Timestamp>
```

```
                    <eb3:MessageId>orders123@buyer.example.com</eb3:Message
Id>
                </eb3:MessageInfo>
                <eb3:PartyInfo>
                    <eb3:From>
                        <eb3:PartyId type="urn:oasis:names:tc:ebxml-
cppa:partyid-type:0002"
                            >123456789</eb3:PartyId>
                        <eb3:Role>Buyer</eb3:Role>
                    </eb3:From>
                    <eb3:To>
                        <eb3:PartyId type="urn:oasis:names:tc:ebxml-
cppa:partyid-type:0106"
                            >192837465</eb3:PartyId>
                        <eb3:Role>Seller</eb3:Role>
                    </eb3:To>
                </eb3:PartyInfo>
                <eb3:CollaborationInfo>
                    <eb3:Service>Sales</eb3:Service>
                    <eb3:Action>ProcessPurchaseOrder</eb3:Action>
                    <eb3:ConversationId>ecae53d4-7473-45a6-
ad70-61970dd7c4b0</eb3:ConversationId>
                </eb3:CollaborationInfo>
                <eb3:PayloadInfo>
                    <eb3:PartInfo href="cid:a1d7fdf5-d67e-403a-
ad92-3b9deff25d43@buyer.example.com"
                    />
                </eb3:PayloadInfo>
            </eb3:UserMessage>
        </eb3:Messaging>
        <wsse:Security S12:mustUnderStand="true">
            <wsu:Timestamp>
                <!-- details omitted -->
            </wsu:Timestamp>
            <wsse:BinarySecurityToken wsu:Id="buyerkey">
                <!-- details omitted -->
            </wsse:BinarySecurityToken>
            <ds:Signature>
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#id_5cb44655-5720-4cf4-
a772-19cd480b0ad4">
                        <ds:Transforms>
                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>qnmr/SB5vgVCRnd1V7QJjkFXab4=</ds:Di
gestValue>
                    </ds:Reference>
                    <ds:Reference URI="#id_f8aa8b55-
b31c-4364-94d0-3615ca65aa40">
```

```
                        <ds:Transforms>
                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>ktCOmnQtHVKJIiXTu7TJJOxA2F8=</ds:Di
gestValue>
                    </ds:Reference>
                    <ds:Reference URI="cid:a1d7fdf5-d67e-403a-
ad92-3b9deff25d43@buyer.com">
                        <ds:Transforms>
                            <ds:Transform
                                Algorithm="http://docs.oasis-
open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-
Transform"
                            />
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>iWNSv2W6SxbOYZliPzZDcXAxrwI=</ds:Di
gestValue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    <!-- details omitted -->
                </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#buyerkey"
                            ValueType="http://docs.oasisopen.org/wss/2004/0
1/oasis-200401-wss-x509-token-profile-1.0#X509v3"
                        />
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="id_f8aa8b55-b31c-4364-94d0-3615ca65aa40" />
</S12:Envelope>

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

UNB+UNOA:2+ABSENDER901234567890123456789OABCDE:CD-A:WEITERLEITNG-
A+EMPFAENGER:CD-E:WEITERLEITNG-E+940812:0235+T940812023504A++0026-
Anwendung'
UNH+M940812023504A+ORDERS:D:93A:UN:EAN007'
BGM+220+5211229'
DTM+002:940815'
NAD+SU+4004353000000::9'
NAD+BY+4306517005214::9'
LIN+1++4004353054099:EN'
```

```
QTY+21:9'
UNS+S'
UNT+51+M940812023504A'
UNZ+3+T940812023504A'
--f1fad5ca-f6b1-4c1b-ba46-099321af7cbe
```

1.11.3 Message from Intermediary to Seller

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

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

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

1.11.4 Message from Seller to Intermediary

Based on P-mode, Seller sends an `eb3:SignalMessage` containing an eb3:`Receipt` for the message displayed in 1.11.2. The receipt has `ds:Reference`s for all structures signed by the WS-Security header in the received message. To be routed through the I-Cloud, an `ebint:RoutingInput` element is added. In this response message, WS-Security is used to sign the `ebint:RoutingInput` and `eb3:Messaging` structures, as well as the empty `S12:Body` element.

```
POST /reroute?mode=async HTTP/1.1
Host: intermediary.example.com:4081
Content-Type: application/soap+xml; action=''; charset="utf-8"

<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:eb3="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:ebint="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/multihop/200902/"
    xmlns:ebbp="http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <S12:Header>
        <wsa:To>http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/part2/200811/icloud</wsa:To>
        <wsa:Action>http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/oneWay.receipt</wsa:Action>
        <ebint:RoutingInput wsa:IsReferenceParameter="true"
            id="id_ccd050c7-a1a5-4c31-8c01-e3c2534609ab">
            <ebint:UserMessage mpc="e5c31ef7-d750-4db8-
b4dc-13a751d80b9a.receipt">
                <ebint:PartyInfo>
```

```
                    <eb3:From>
                        <eb3:PartyId type="urn:oasis:names:tc:ebxml-
cppa:partyid-type:0106"
                            >192837465</eb3:PartyId>
                        <eb3:Role>Seller</eb3:Role>
                    </eb3:From>
                    <eb3:To>
                        <eb3:PartyId type="urn:oasis:names:tc:ebxml-
cppa:partyid-type:0002"
                            >123456789</eb3:PartyId>
                        <eb3:Role>Buyer</eb3:Role>
                    </eb3:To>
                </ebint:PartyInfo>
                <ebint:CollaborationInfo>
                    <eb3:Service>Sales</eb3:Service>
                    <eb3:Action>ProcessPurchaseOrder.receipt</eb3:Action>
                    <eb3:ConversationId>ecae53d4-7473-45a6-
ad70-61970dd7c4b0</eb3:ConversationId>
                </ebint:CollaborationInfo>
            </ebint:UserMessage>
        </ebint:RoutingInput>
        <eb3:Messaging S12:mustUnderstand="true" id="id_393bb2e2-
df86-4b4f-b682-f7a684316b3d">
            <eb3:SignalMessage>
                <eb3:MessageInfo>
                    <eb3:Timestamp>2009-05-22T14:33:11.735Z</eb3:Timestamp
>
                    <eb3:MessageId>orderreceipt@seller.example.com</eb3:Me
ssageId>
                    <eb3:RefToMessageId>orders123@buyer.example.com</eb3:R
efToMessageId>
                </eb3:MessageInfo>
                <eb3:Receipt>
                    <ebbp:NonRepudiationInformation>
                        <ebbp:MessagePartNRInformation>
                            <ds:Reference URI="#id_5cb44655-5720-4cf4-
a772-19cd480b0ad4">
                                <ds:Transforms>
                                    <ds:Transform
                                        Algorithm="http://www.w3.org/2001/
10/xml-exc-c14n#"/>
                                </ds:Transforms>
                                <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                                <ds:DigestValue>qnmr/SB5vgVCRnd1V7QJjkFXab
4=</ds:DigestValue>
                            </ds:Reference>
                        </ebbp:MessagePartNRInformation>
                        <ebbp:MessagePartNRInformation>
                            <ds:Reference URI="#id_f8aa8b55-
b31c-4364-94d0-3615ca65aa40">
                                <ds:Transforms>
                                    <ds:Transform
                                        Algorithm="http://www.w3.org/2001/
10/xml-exc-c14n#"/>
```

```
                                </ds:Transforms>
                                <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                                <ds:DigestValue>ktCOmnQtHVKJIiXTu7TJJOxA2F
8=</ds:DigestValue>
                            </ds:Reference>
                        </ebbp:MessagePartNRInformation>
                        <ebbp:MessagePartNRInformation>
                            <ds:Reference
                                URI="cid:a1d7fdf5-d67e-403a-
ad92-3b9deff25d43@buyer.example.com">
                                <ds:Transforms>
                                    <ds:Transform
                                        Algorithm="http://docs.oasis-
open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-
Transform"
                                    />
                                </ds:Transforms>
                                <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                                <ds:DigestValue>iWNSv2W6SxbOYZliPzZDcXAxrw
I=</ds:DigestValue>
                            </ds:Reference>
                        </ebbp:MessagePartNRInformation>
                    </ebbp:NonRepudiationInformation>
                </eb3:Receipt>
            </eb3:SignalMessage>
        </eb3:Messaging>
        <wsse:Security S12:mustUnderStand="true">
            <wsu:Timestamp>
                <!-- details omitted -->
            </wsu:Timestamp>
            <wsse:BinarySecurityToken wsu:Id="sellerkey">
                <!-- details omitted -->
            </wsse:BinarySecurityToken>
            <ds:Signature>
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#id_ccd050c7-a1a5-4c31-8c01-
e3c2534609ab">
                        <ds:Transforms>
                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>zHsWMe5iRfxbfe74CAsRJih1Dug=</ds:D
igestValue>
                    </ds:Reference>
                    <ds:Reference URI="#id_393bb2e2-df86-4b4f-b682-
f7a684316b3d">
                        <ds:Transforms>
```

```
                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>DAZle0qOJQnxy+1XmGX3672AOZQ=</ds:D
igestValue>
                    </ds:Reference>
                    <ds:Reference URI="#id_861ff127-
f930-4934-8ad0-5b91bd6dbc1e">
                        <ds:Transforms>
                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
                        <ds:DigestValue>NSfMvBI1BaM/7Mv0bKnvkozFLqw=</ds:D
igestValue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    <!-- details omitted -->
                </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#sellerkey"
                            ValueType="http://docs.oasisopen.org/wss/2004/
01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
                        />
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </S12:Header>
    <S12:Body wsu:Id="id_861ff127-f930-4934-8ad0-5b91bd6dbc1e"/>
</S12:Envelope>
```

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

1.11.5 Receipt from Intermediary to Buyer

Again, the intermediary only rewrites the HTTP header:

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

1.12 Appendix C: Using WS-Addressing Headers

In case more wsa headers are used than required by the previous multihop functions and use cases, it is useful to clarify their expected semantics in a multihop environment. The following sections also serve as best practices when using additional wsa headers.

1.12.1 Use of wsa:ReplyTo Header

The wsa:ReplyTo header may be used in case it is not expected that the responding endpoint has enough routing information for the response message.

- For an ebMS message, this could be the case if the PMode is incomplete or not deployed on the destination endpoint
- The wsa:ReplyTo header should not be necessary for non-ebms messages that are used by the reliable messaging function (CreateSequence, CloseSequence, TerminateSequence, AckRequested). Indeed, this specification requires that all RM responses be sent back to the same EPR as the EPR specified in /wsrm:CreateSequence/wsrm:AcksTo . Therefore, if the wsa:ReplyTo is present on any of these messages, it should contain an EPR that resolves to the same MSH as the AcksTo EPR.

The value of wsa:ReplyTo has to be set to the appropriate reply EPR, as dictated by the PMode. See the section “PMode Extension for EPRs and Routing Input”.

Case of anonymous wsa:ReplyTo: In principle the absence of wsa:ReplyTo, or the presence of wsa:ReplyTo with an anonymous URI, is of no significance for the Intermediary MSH. Because the actual destination of the message is beyond the Intermediary, the Intermediary SHOULD NOT interpret this header as a requirement to send the response message over the same HTTP connection (when using HTTP). The destination MSH must do this interpretation. However, in many cases when using anonymous wsa:ReplyTo, the sending endpoint will expect the same addressing behavior from the I-Cloud as from a point-to-point partner: the response is expected over an HTTP response (back-channel). It must be noted that this HTTP response is not necessarily part of the same HTTP connection that carried the request message, as allowed by WS-I BP2.0.
Consequently, the use of wsa:ReplyTo anonymous is only compatible with the multihop edge bindings described in section 1.5.3.2 (cases 3 and 4). The routing function implemented in the intermediary participating in the first edge-hop, will decide of which case (3 or 4) is enforced. This behavior must be aligned with the P-Mode deployed on the endpoint. In case 3, an MPC for pulling responses is associated with the routing function handling these responses.
The behavior of other MSHs on the multihop path is as follows:

- The destination endpoint MUST comply by sending the response over the underlying protocol back-channel (HTTP response, if HTTP), as expected in point-to-point setting.
- The other intermediaries in the I-Cloud, are not restricted to forward the response message over the underlying protocol back-channel (HTTP response, if HTTP). Their routing function will decide of the transfer mode. Routing input will be provided by the message header, which could be reference parameters associated with the EPR provided as the wsa:ReplyTo value.

Case of non-anonymous wsa:ReplyTo: The presence of wsa:ReplyTo with an EPR containing a non-anonymous URI falls into the following categories:

(a) The URI can be resolved as a reachable Internet address. This indicates that the response message is not subject to I-Cloud routing. It should instead be directly addressed to the specified URL. A subcase is when the URI does not identify the ultimate destination of the response, but instead identifies the edge Intermediary used by the request message sender (i.e. Intermediary involved in the first hop of the request message, and assumed to also be the last Intermediary of the response). In that case, and if the message is not an eb:UserMessage, the EPR MUST also contain the eb:RoutingInput Reference parameter, so that the message can be routed to its final destination, e.g. pulled by the request sender on the MPC specified in the reference parameter (piggybacked on a EmptyMessagePartitionChannel warning - EBMS:0006 - in case of a non-ebMS signal, as explained in 1.6.1).

(b) The URI is the “icloud” URI. This indicates the need to rely on the I-Cloud routing function. Routing of the reply message in turns relies either on message header data for routing input, or on the reference parameter ebint:RoutingInput contained in the wsa:ReplyTo EPR, as described in 1.7.3.

1.12.2 Other wsa Headers

An Endpoint MSH that inserts a wsa:ReplyTo header with a soapMustUnderstand=”1” attribute, must comply with the use of WS-Addressing as required by WS-I Basic Profile 2.0. This means:

(1) Along with the wsa:ReplyTo header, the message must also contain a wsa:MessageID header, in addition to wsa:To and wsa;Action headers. In eb UserMessages, the wsa;Action value must be same as the eb:Action value.

(2) Any message responding to a message containing a wsa:ReplyTo header, must also contain a wsa:RelatesTo header with attribute @RelationshipType = 'http://www.w3.org/2005/08/addressing/reply', and with an element value set to the wsa:MessageID of the message responded to. The response message must also include headers for reference parameters specified in the EPR value used in wsa:ReplyTo.

About wsa:FaultTo:

This header may be set to the Error reporting EPR. (see the section “PMode Extension for EPRs and Routing Input”.)