OASIS ebXML Messaging Services
Version 3.0 Part 2 – Advanced Features
Version 0.1
Draft
22 August 2007

Specification URIs:
This Version:
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/advanced/ebms3-advanced-wd-01.html
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/advanced/ebms3-advanced-wd-01.doc

Previous Version:

Latest Version:
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/advanced/ebms3-advanced-wd-01.html
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/advanced/ebms3-advanced-wd-01.doc

Latest Approved Version:

Technical Committee:
OASIS ebXML Messaging Services TC

Chair(s):
Ian Jones

Editor(s):
Hamid Ben Malek
Jacques Durand

Related work:
This specification completes:
• Oasis ebXML Messaging Services Version 3.0: Part 1, Core Features
• Oasis ebXML Messaging Services Version 3.0: Part 1, Conformance Profiles
• Oasis ebXML Messaging Services Version 3.0: Part 1, Implementation Notes

This specification is related to:
• WS-Addressing
• WSS
• WS-Reliability
• WS-ReliableMessaging

Declared XML Namespace(s):
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
Abstract:
This specification defines a communications-protocol neutral method for exchanging electronic business messages. It defines specific enveloping constructs supporting reliable, secure delivery of business information. Furthermore, the specification defines a flexible enveloping technique, permitting messages to contain payloads of any format type. This versatility ensures legacy electronic business systems employing traditional syntaxes (i.e. UN/EDIFACT, ASC X12, or HL7) can leverage the advantages of ebXML infrastructure along with users of emerging technologies.

Status:
This document has not been yet revised or approved by the TC. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

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

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/ebxml-msg/ipr.php).
Notices

Copyright © OASIS® 2007. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

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

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

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

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

Ebms3_Advanced
Copyright © OASIS® 2007. All Rights Reserved.

22 August 2007
Page 3 of 35
# Table of Contents

1  Introduction .................................................................................................................. 6
    1.1 Terminology ........................................................................................................... 6
    1.2 Normative References ............................................................................................ 6
    1.3 Non-Normative References .................................................................................... 6

2  Messaging Model ............................................................................................................ 7
    2.1 Terminology and Concepts ..................................................................................... 7
        2.1.1 Components of the Model ............................................................................. 7
        2.1.2 Terminology .................................................................................................. 8
        2.1.3 Messaging Roles ......................................................................................... 8
        2.1.4 Abstract Messaging Operations .................................................................... 8
    2.2 Message Exchange Patterns .................................................................................... 9
        2.2.1 Aggregate ebMS Message Exchange Patterns ............................................ 9
        2.2.2 Complex ebMS Message Exchange Patterns ............................................. 9

3  Processing Modes ......................................................................................................... 12
    3.1 Case of a One-Way/Pull ....................................................................................... 13
    3.2 Case of a complex orchestration .......................................................................... 16
    3.3 PMode Packaging ................................................................................................ 19
        3.3.1 Error Reporting in PModes ...................................................................... 20
        3.3.2 Quality of Service in PModes .................................................................... 21
        3.3.3 Default Values ............................................................................................ 23

4  Message Packaging ....................................................................................................... 25
    4.1 New Signal Messages ............................................................................................ 25
        4.1.1 Ping/Pong Signal Messages ....................................................................... 25
        4.1.2 Message Status Signal Messages ............................................................... 25
        4.1.3 Flow Control Signal Messages ................................................................ 25
        4.1.4 Agreement Exchange Signal Messages .................................................... 25
    4.2 Message Bundling .................................................................................................. 25
        4.2.1 User Message Unit Bundling ..................................................................... 25
        4.2.2 UserMessage/SignalMessage Combination ............................................. 25

5  WS-Addressing Module ............................................................................................... 26

6  Payload Services Module ............................................................................................. 27
    6.1 Large Attachments ............................................................................................... 27
    6.2 Transformations ...................................................................................................... 27
    6.3 Compression ........................................................................................................... 27
    6.4 Encryption ............................................................................................................. 27

7  Orchestration/Workflow ............................................................................................... 28

8  WSDL Interfaces ........................................................................................................... 29
    8.1 WSDL Interfaces to MSH Operations .................................................................. 29
    8.2 WSDL Interface to PartyId .................................................................................. 29

9  Gateway Component .................................................................................................... 30
    A. Communication-Protocol Bindings ....................................................................... 31
    B. WS-I Compliance ................................................................................................... 32
    C. Acknowledgements ............................................................................................... 33
D. Non-Normative Text ........................................................................................................... 34
E. Revision History ............................................................................................................. 35
1 Introduction

[All text is normative unless otherwise labeled]

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in Error! Reference source not found..

1.2 Normative References

[Reference] [Full reference citation]

1.3 Non-Normative References

[Reference] [Full reference citation]
2 Messaging Model

2.1 Terminology and Concepts

This section defines the messaging model of ebms3 part 2 along with its main concepts. In part 1 of the ebms3 specification, the messaging model consisted of a single pair of MSHs, with one MSH playing the role of a sender and the other playing the role of a receiver.

The messaging model of ebms3 part 2 specification is much more complex than the one focused on by part 1. It does not consists of a pair of MSH talking to each other but rather it covers a graph of various MSHs all working together in a collaborative way along with many partyIds behind each MSH.

2.1.1 Components of the Model

The ebMS messaging model assumes the following components:

- A set of distributed MSHs, each called a "Participant". These MSHs may all be located within a small area such as a private campus network or be separated across the planet throughout the Internet.
- Each Participant may serve many partyIds behind it. A "Partyld" is any component that can invoke a "Submit" operation on a MSH or that the MSH can invoke a "Deliver" operation on it.
- The participants exchange messages with each other's according to a message workflow (or choreography) that we call "Complex MEP" and defined more precisely in Section 8 (Orchestration/Workflow)
- The message exchange choreography is always initiated (started) at only one given participant, called "Initiating Participant". Any partyld behind the initiating participant may drive a new message choreography.

The following picture illustrates the messaging model:
In the messaging model depicted above, MSH 1 plays the role of the Initiator Participant. Any partyId behind MSH 1 may start new message choreography. The choreography described above starts first by a user message transfer from MSH 1 to MSH 2, then MSH 2 sends two user messages in parallel (at the same time) to two different MSHs (one for MSH 3 and the second for MSH 4). MSH 4 transfers a user message to MSH 5 and then MSH 5 transfers another user message to MSH 3. Finally MSH 3 transfers a user message to MSH 1. The choreography does not have to be a loop (meaning that the final user message transfer does not have to be received by the Initiator Participant. Each user message transfer between any two participants corresponds to one leg of one of the seven MEPs described in part 1 of the ebms3 specification, namely “One-Way/Push”, “One-Way/Pull”, “Two-Way/Sync”, “Two-Way/Push-And-Push”, “Two-Way/Push-And-Pull”, “Two-Way/Pull-And-Push”, and “Two-Way/Pull-And-Pull”.

2.1.2 Terminology

- The words “Collaboration”, “Orchestration” and “Complex ebMS MEP” will all be used to mean the same thing. It consists of a set of messages exchanged between a graph of various distributed MSHs all collaborating in a certain way. A “Collaboration” (or “Complex MEP”) will be defined more precisely in the following sections.

2.1.3 Messaging Roles

In part 1 of the ebms3 specification, the messaging model was very simple as it consisted of only one pair of MSHs, and therefore two roles were available: sending and receiving roles. However, the above messaging model is more complex and it does not have these two roles. There is an “initiator” role and it is played by the MSH that drives (or starts) the message choreography. All the other MSHs in the model have the same role which is the “participant” role.

2.1.4 AbstractMessaging Operations

Part 1 of the ebms3 specification defined five abstract operations, namely: Submit, Deliver, Notify, Send, and Receive. The messaging model described above introduces additional operations which are the following:
2.2 Message Exchange Patterns

Part 1 of the ebms3 specification defined the concept of an “abstract” MEP and the transport channel binding. However, when we use the word MEP in this document, we mean the combination of an abstract MEP and a given transport-channel binding.

MEPs are divided into three categories. In other words, an MEP can either be a “Simple MEP”, an “Aggregate MEP”, or a “Complex MEP”. Part 1 of the specification defined Simple and Aggregate MEPs but not the “Complex MEPs” which are the subject of this document.

2.2.1 Aggregate ebMS Message Exchange Patterns

Part 1 of the ebMS 3 specification defined three “Simple MEP”: One-Way/Push, One-Way/Pull, and Two-Way/Sync. The other MEPs were called “Aggregate MEPs” and they are: Two-Way/Push-And-Pull, Two-Way/Pull-And-Push, and Two-Way/Pull-And-Pull. Aggregate MEPs will be more examined in the WS-Addressing Module section.

2.2.2 Complex ebMS Message Exchange Patterns

First let us use the BNF (Backus-Naur Form) syntax to define a complex ebMS MEP. Terminals (pieces of final strings that do not have a production rule) will be placed within quotes.

In BNF syntax, a complex ebMS MEP is defined as follows:

```
Complex_ebms_MEP ::= "("name "," Leg\{Leg\})" ;
Leg ::= "\"(n \"," message\")\" | "\"(n \"," Complex_ebms_MEP \")\" ;
n ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | etc...
message ::= 'ebMS Pull Request Message' | 'ebMS User Message' ;
```

The above definition says that a complex ebMS MEP has a given name and a collection (i.e. a set) of Legs. A Leg has always a number (positive integer) and may consist of either a message or a complex ebMS MEP. Whereas a message is either a Pull Request Message or a UserMessage. Note that a message (be it a pull request message or a user message) is directional, meaning that it comes from an MSH and is destined to another MSH. In other words, to completely define a message, it is not sufficient to know whether it is a pull request or a user message but to also know which MSH it comes from and to what MSH it goes to. In mathematical form, this can be expressed as a triple (From-MSH, message, To-MSH). The name of the MEP carries this missing information that tells us the from-MSH and to-MSH of each leg in the MEP. The number of any given leg indicates its order in time. A leg with number 1 means that it occurs first in time (the initiating leg). If two legs have the same number, it means that they occur at the same time. Also if a message flows from an MSH-A to an MSH-B, and there was a previous message that was traveled from in the other direction (from MSH-B to MSH-A), then the message traveling from MSH-A to MSH-B must have an eb:RefToMessageld that refers to that previous message. In other words,
a message must refer to some previous message that traveled in the opposite direction (if such a message exists of course).

To illustrate the above definition, let us examine the basic seven MEPs described in part 1 of the ebms 3 specification in terms of the new BNF definition above:

- **One-Way/Pull:** This MEP, if expressed in terms of the above BNF definition, could simply be written as (“One-Way/Pull”, {(1, p), (2, u)}). This means that the name of the MEP is “One-Way/Pull”, and that it is a collection of two legs, one called “p” (representing a pull request message from MSH A to MSH B) and the other called “u” (representing a user message from MSH B to MSH A). The leg that is called “p” has number 1 because it happens first in time and the other leg has number 2 because it happens after the first leg. There is nothing in the BNF notation above that says that the pull request message (p) is from MSH A to MSH B, while the user message (u) is from MSH B to MSH A. This missing information is contained within the name of the MEP itself. In other words, because the above MEP is called “One-Way/Pull”, this means the first leg (p) is from MSH A to MSH B and that the second leg (u) should be from MSH B to MSH A.

- **One-Way/Push:** This MEP can be represented as (“One-Way/Push”, {(1, u)})

- **Two-Way/Sync:** This MEP can be represented as (“Two-Way/Sync”, {(1, u), (2, u)}). Again, there is no information in this BNF representation that says that the second leg should be the back channel of the first leg. However this information is supposed to be contained within the name of the MEP itself, which is in this case “Two-Way/Sync”.

- **Two-Way/Push-And-Push:** This MEP can be represented as (“Two-Way/Push-And-Push”, {(1, u), (2, u)}). The fact that the second leg in this MEP flows within a separate channel (not the back channel of the first leg) is not contained within the BNF representation but in the name of the MEP itself.

- **Two-Way/Push-And-Pull:** This MEP can be represented as (“Two-Way/Push-And-Pull”, {(1, u), (2, p), (3, u)}). Here the first and second leg both have the same from-MSH and to-MSH and the third leg has the opposite direction and flows in the back channel of the second leg. This information is not contained per say in the syntax itself, but carried by the name of the MEP itself which tells us the from/to MSH of each leg and whether a leg flows on the back channel or in its own separate transport channel.

- **Two-Way/Pull-And-Push:** This MEP can be represented as (“Two-Way/Pull-And-Push”, {(1, p), (2, u), (3, u)}). Here the second leg flows in the back channel of the first leg, while the third leg flows in its own separate transport channel.

- **Two-Way/Pull-And-Pull:** This MEP can be represented as (“Two-Way/Pull-And-Pull”, {(1, p), (2, u), (3, p), (4, u)}). Here the second leg flows in the back channel of the first leg, while the fourth leg flows in the back channel of the third leg. The first and fourth legs have the same direction (i.e. same from-MSH and to-MSH) while the second and third legs have the same direction which is the opposite of the first one.

Now that we have represented the seven known MEPs (mentioned more or less in part 1 of the specification), let us give an example of a complex MEP and see how it can be represented using the BNF syntax. The following picture illustrates one example of such a complex MEP (there are infinitely many possible complex MEPs by the way):
The above picture shows a complex MEP that consists of the following: First MSH 1 initiates a user message push to MSH 2. MSH 2 may or may not consume some of the payloads, and then issues two different sync (i.e. Two-Way/Sync) at the same time. The Leg between MSH 2 and MSH 3 has number 2, while the leg between MSH 2 and MSH 4 has the same number 2. This means these two legs may happen in parallel at the same time (in practice they may occur at the same time or may occur one after the other, but what this means is that it does not matter which one occurs first or if they occur in parallel). MSH 2 then receives two independent responses, one from MSH 3 and the other from MSH 4. Finally MSH 1 makes a pull request to MSH 2 to request the result which may be some combination of the responses received by MSH 2.

How can we define the above complex MEP with the BNF representation? First we should notice that our complex MEP has four legs numbered respectively 1, 2, 2, and 3. The first leg representing a One-Way/Push, the second and third leg both represents a Two-Way/Sync, while the fourth leg represents a One-Way/Pull. Thus the above complex MEP can be represented as follows:

```
("Some Name", {(1, mep1), (2, mep2), (2, mep3) (4, mep4)})
```

where “Some Name” is the name we gave to our complex MEP, mep1, mep2, mep3, and mep4 are separate MEPs defined as:

- mep1 = ("One-Way/Push", {(1, u)})
- mep2 = ("Two-Way/Sync", {(1, u), (2, u)})
- mep3 = ("Two-Way/Sync", {(1, u), (2, u)})
- mep4 = ("One-Way/Pull", {(1, p), (2, u)})
3 Processing Modes

It was mentioned earlier that the messaging model in this document consisted of a graph of various MSHs collaborating together by exchanging messages. This collaboration is captured by a “Complex MEP” (also called “Orchestration”) and therefore it can be played many times over and over again. Each time the collaboration is replayed, everything looks the same as the previous replay except for the payloads themselves which may change. In order to play a given collaboration, every MSH involved in this collaboration (these MSHs are called “Participants”) needs to have a representation of all the details of the collaboration. Each Participant MSH would need to have the same representation of the collaboration that will be played. Such a representation (which may take many forms such as a pure XML document, or a CPA document, or any other type of possible format) is completely encapsulated in what we call a PMode (Processing Mode). Therefore, a PMode contains all the necessary information with all the details that enable a set of Participant MSHs to play a well defined collaboration (as many times as necessary).

Let us define a PMode using an XML representation for example. A PMode is defined by a name and a binding. Every PMode has a given name and a Binding object, and thus can be represented as follows:

```xml
<PMode name="MyFirstPMode" binding="SomeBinding" />
<Binding name="SomeBinding">
  …
</Binding>
```

The binding object of a PMode can be specified as an attribute (as in the example above), or it can be directly specified within the PMode tag itself as in the following example:

```xml
<PMode name="MyFirstPMode">
  <Binding name="SomeBinding">
    …
  </Binding>
</PMode>
```

If the PMode tag contains many binding elements inside it, then it must specify which binding is the one that defines the PMode, as illustrated in the following example:

```xml
<PMode name="MyFirstPMode" binding="mainBinding">
  <Binding name="SomeBinding">
    …
  </Binding>
  <!--this is the binding that defines the PMode and is referred to in the PMode attribute -->
  <Binding name="mainBinding">
    …
  </Binding>
</PMode>
```

A binding object has a name and defines an orchestration that specifies the properties of each leg and how it binds to the transport channel.
3.1 Case of a One-Way/Pull

Let’s illustrate how a PMode can be defined in XML for the simplest orchestration possible: a simple One-Way/Pull involving a single pair of MSHs.

The following example is a representation of a binding object:

```xml
<Binding name="SomeBinding">
  <Orchestration mep="One-Way/Pull">
    <Leg number="1" mpc="orders">
      <Endpoint address="http://foo.com/ebms3/soapVersion="1.2"/>
      <Authorization type="UsernameToken" username="bob" password="secret" />
    </Leg>
    <Leg number="2" userService="barUserService" producer="foo" messageLabel="orderMsg"/>
  </Orchestration>
</Binding>

<UserService name="barUserService">
  ...
</UserService>

<Producer name="foo">
  <PartyId>uri:foo.com</PartyId>
  <Role>Buyer</Role>
</Producer>
```

The above example represents the binding for a partyId (“uri:bar.com”) that has a UserService called “barUserService” and would like to consume user messages by pulling them from a remote partner called “uri:foo.com” on the mpc called “orders”. The remote partner requires a UsernameToken in the PullRequest in order to authorize the pulling (the username token would be placed within a wss:Security element that has “ebms3” role).

- The value of `Binding/Orchestration@mep` attribute can be one of the following seven values: One-Way/Pull, One-Way/Push, Two-Way/Sync, Two-Way/Push-And-Push, Two-Way/Pull-And-Push, and Two-Way/Pull-And-Pull. This attribute is optional if the orchestration is complex (meaning that its core MEP cannot be reduced to one of the seven defined MEPs -- Simple and Aggregate MEPs).
- The attribute `Binding/Orchestration/Leg@number` is required and must be a positive integer (>=1). The first leg must have number="1". This attribute indicates the occurrence in time (with 1 meaning the first leg to occur).
- The optional attribute `Binding/Orchestration/Leg@mpc` indicates which MPC (Message Partition Channel) to use for that leg. For example, if the leg represents an eb:PullRequest message, then the mpc value would go as an attribute to the PullRequest element. If however, the leg represents a UserMessage, the mpc value would go as an attribute to the eb:UserMessage element.
- The optional attribute `Binding/Orchestration/Leg@producer` indicates what values should go in the from party (eb:PartyInfo/eb:From element) of the current leg. In the example above, the pulled user message would have uri:foo.com in the eb:From/eb:PartyId.
- The optional attribute `Binding/Orchestration/Leg@userService` specifies the destination service (also called “UserService”): eb:PartyInfo/eb:To, eb:CollaborationInfo/eb:Service, and eb:CollaborationInfo/eb:Action.
The element Binding/Orchestration/Leg has other optional attributes which are the following: "binding", "messageLabel", "soapAction", "wsaAction". If the attribute Binding/Orchestration/Leg@binding is present then only the "number" attribute should be present and all the other attributes should be absent. The Binding/Orchestration/Leg@binding attribute is used when defining complex MEPs (i.e. collaboration/orchestration whose core MEP cannot be reduced to the one of the well know seven MEP). The attributes Binding/Orchestration/Leg@soapAction and Binding/Orchestration/Leg@wsaAction indicate what values to use respectively for soapAction and wsa:Action in the current leg. These two attributes specify the minimum binding with the WS-Addressing module (more on this, see WS-Addressing Module Section).

Now let's define the UserService called "barUserService":

```xml
<UserService name="barUserService">
  <ToPartyInfo>
    <PartyId>uri:bar.com</PartyId>
    <Role>Seller</Role>
  </ToPartyInfo>
  <CollaborationInfo>
    <Service>CarService</Service>
    <Action>Lease</Action>
  </CollaborationInfo>
  <MessageProperties>
    <Property name="year" type="date" description="The year the car was made" required="true" />
  </MessageProperties>
  <PayloadInfo>
    <Message label="orderMsg" maxSize="3000">
      <SoapBody schemaLocation="http://bar.com/order.xsd" />
      <Part cid="agr" mimeType="application/pdf" description="signed agreement"/>
    </Message>
  </PayloadInfo>
</UserService>
```

The above example shows the details of the UserService that will consume the pulled user message. The elements within the "UserService" element are self-explanatory:

MH-1 (serving partyId "uri:bar.com" playing the role of a seller) will send an eb:PullRequest message to MH-2 serving partyId "uri:foo.com" along with a UsernameToken in a wss:Security with "ebms3" as role. MH-2 will authorize the pull request based on the username token, and a user message would be sent back on the back channel of the pull request. The ebMS headers of this user message will be populated as follows:

- UserService/ToPartyInfo/PartyId will go in the eb:PartyInfo/eb:to
- UserService/CollaborationInfo/Service will go in the eb:CollaborationInfo/eb:Service.
- UserService/CollaborationInfo/Action will go in the eb:CollaborationInfo/eb:Action.
- The pulled user message should have a message property with the name "year" and value of type date that indicates the year the car was made. This message property would be represented as an element within the eb:UserMessage/eb:MessageProperties element.
- The payload of the pulled user message would consist in an XML document in the SOAP body and an attachment in the form of a pdf document. The XML document within the SOAP body should conform to the XML schema located at http://bar.com/order.xsd and the total size of both the XML document and the pdf attachment should not exceed 3000 kilobytes.
Now that we have explained the various pieces, we can list the whole PMode definition in the following example:

```xml
<PMode name="MyFirstPMode">
  <Producer name="foo">
    <PartyId>uri:foo.com</PartyId>
    <Role>Buyer</Role>
  </Producer>

  <UserService name="barUserService">
    <ToPartyInfo>
      <PartyId>uri:bar.com</PartyId>
      <Role>Seller</Role>
    </ToPartyInfo>
    
    <CollaborationInfo>
      <Service>CarService</Service>
      <Action>Lease</Action>
    </CollaborationInfo>

    <MessageProperties>
      <Property name="year" type="date" description="The year the car was made" required="true" />
    </MessageProperties>

    <PayloadInfo>
      <Message label="orderMsg" maxSize="3000">
        <SoapBody schemaLocation="http://bar.com/order.xsd" />
        <Part cid="agr" mimeType="application/pdf" description="signed agreement" />
      </Message>
    </PayloadInfo>

  </UserService>

  <Binding name="SomeBinding">
    <Orchestration mep="One-Way/Pull">
      <Leg number="1" mpc="orders">
        <Endpoint address="http://foo.com/ebms3/" soapVersion="1.2" />
      </Leg>

      <Leg number="2" userService="barUserService" producer="foo" messageLabel="orderMsg" />
    </Orchestration>
  </Binding>
</PMode>
```

To summarize, the above PMode definition defines one PMode called “MyFirstPMode” between two MSHs: MSH-1 serving a partyId called “uri:bar.com” playing a seller role, and MSH-2 serving a partyId called “uri:foo.com” playing a buyer role.

- The first partyId (uri:bar.com) has a UserService called “barUserService” that would like to consume user messages by pulling them remotely from the address “http://foo.com/ebms3/” (the address of MSH-2) on the mpc called “orders”. PartyId “uri:bar.com” must provide a Username token in the pull request (with username=bob and password=secret) in order for MSH-2 (or partyId “uri:foo.com” itself) to authorize the pull request. This username token is placed within a wss:Security element with “ebms3” as role (this security element is not removed after the pull request goes through the Security Module because it is not intended for the Security Module itself but is intended for the ebMS Module which may do the authorization himself or forward it to the final partyId to perform the authorization).

- The pulled user message that is returned in the back channel of the pull request should have its ebMS headers populated as follows:
  - The eb:UserMessage/eb:PartyInfo/eb:From is populated from the PMode/Producer element.
  - The eb:UserMessage/eb:PartyInfo/eb:To is populated from the PMode/UserService/ToPartyInfo element.
The eb:UserMessage/eb:CollaborationInfo is populated from the
PMode/UserService/CollaborationInfo element.

The eb:UserMessage/eb:MessageProperties should contain a message property with the
name "year" and a value whose type is date; because the UserService called
"barUserService" says so.

The payload data in the pulled user message should not exceed 3000 Kilobytes as
indicated in the definition of the UserService that will consume the pulled user message.
Also this payload data should consist of an XML document placed in the SOAP body
which should conform to the XML schema located at http://bar.com/order.xsd and the
attachment data should consist of a PDF document (representing some signed
agreement between the buyer and seller).

Now let us consider an example of a complex orchestration and see how its PMode can be defined in
XML:

3.2 Case of a complex orchestration

Let us consider the example of a complex orchestration that we mentioned earlier in previous sections
and which is illustrated by the following picture:

The above picture assumes the following:

- MSH-1 is the initiator Participant of the orchestration. The partyId behind MSH-1 (call it p1.com)
can start the orchestration at any time by pushing a user message to MSH-2.
- MSH-2 may consume some of the payloads by giving them to the partyId behind MSH-2 (call it
p2.com). Then MSH-2 pushes two different user messages (one intended for MSH-3 and the
other intended for MSH-4) in a synchronous way (two independent Two-Way/Sync MEPs in
parallel). PartyId “p3.com” (the one behind MSH-3) consumes the payload and returns its
response on the back channel. Similarly, partyId “p4.com” (the one behind MSH-4) consumes the
payload and returns its response on the back channel.
- MSH-2 gets back two user messages are responses (one from MSH-3 and the other from MSH-4), and may combine them into one single user message (the sum of the payloads returned in both responses) which is placed on a mpc called “results”.

- MSH-1 then sends a PullRequest message to MSH-2 in order to pull the user message from the “results” mpc.

The PMode that describes such an orchestration could be represented by the following XML document:

```xml
<PMode name="MyComplexPMode" binding="main-binding">
  <!-- this is a partyId behind MSH-1 -->
  <Producer name="p1">
    <PartyId>p1.com</PartyId>
  </Producer>

  <!-- this is a partyId behind MSH-2 -->
  <Producer name="p2">
    <PartyId>p2.com</PartyId>
  </Producer>

  <!-- this is a partyId behind MSH-3 -->
  <Producer name="p3">
    <PartyId>p3.com</PartyId>
  </Producer>

  <!-- this is a partyId behind MSH-4 -->
  <Producer name="p4">
    <PartyId>p4.com</PartyId>
  </Producer>

  <!-- this represents the user service of p1 that consumes the pulled user message -->
  <UserService name="p1UserService">
    <ToPartyInfo>
      <PartyId>p1.com</PartyId>
    </ToPartyInfo>
    <CollaborationInfo>
      <Service>p1-service</Service>
      <Action>p1-action</Action>
    </CollaborationInfo>
  </UserService>

  <!-- this represents the user service of p2 that consumes the pushed user message -->
  <UserService name="p2UserService">
    <ToPartyInfo>
      <PartyId>p2.com</PartyId>
    </ToPartyInfo>
    <CollaborationInfo>
      <Service>p2-service</Service>
      <Action>p2-action</Action>
    </CollaborationInfo>
  </UserService>

  <!-- this represents the user service of p3 that consumes the user message sent by MSH 2 -->
  <UserService name="p3UserService">
    <ToPartyInfo>
      <PartyId>p3.com</PartyId>
    </ToPartyInfo>
    <CollaborationInfo>
      <Service>p3-service</Service>
      <Action>p3-action</Action>
    </CollaborationInfo>
  </UserService>

  <!-- this represents the user service of p4 that consumes the user message sent by MSH 2 -->
  <UserService name="p4UserService">
    <ToPartyInfo>
      <PartyId>p4.com</PartyId>
    </ToPartyInfo>
    <CollaborationInfo>
      <Service>p4-service</Service>
      <Action>p4-action</Action>
    </CollaborationInfo>
  </UserService>
</PMode>
```
The binding element whose name is “main-binding” has four legs (representing the four legs of the whole orchestration), two of which have the same number (the second and third leg have both number attribute equal to 2). This is because these two legs are independent and are authorized to happen in parallel in the implementation can enforce that, otherwise it does not matter which one occurs first.

The above example was only to illustrate how to represent a complex orchestration with an XML document. It can be noted however that the quality of service was not mentioned (none of the legs had a reference to the quality of service). Also, error reporting was absent from the above PMode representation. This was done on purpose to first illustrate the concept, but the XML language used here...
allows for such things such as specifying certain quality of service for certain legs only and for defining
how errors should be reported in certain legs if problems were found with the messages they carry. The
next subsection will define the absent elements.

### 3.3 PMode Packaging

In this section we will give a complete reference of the XML language used to represent a PMode and
show how to package many PModes inside one single document.

A PMode can be represented alone in one XML document or many PModes can all be packed in one
single XML document.

If the XML document contains only one PMode, then such an XML document would look like the
following:

```xml
<PMode name="MyPMode" binding="myMainBinding">
    <Producer name="producer-1"> ... </Producer>
    <Producer name="producer-2"> ... </Producer>
    ...
    <UserService name="userService-1"> ... </UserService>
    <UserService name="userService-2"> ... </UserService>
    <UserService name="userService-3"> ... </UserService>
    ...
    <Binding name="binding-1"> ... </Binding>
    <Binding name="binding-2"> ... </Binding>
    <Binding name="myMainBinding"> ... </Binding>
</PMode>
```

If the XML document contains many PModes, then such an XML document may look like the following:

```xml
<PModes>
    <Producer name="producer-1"> ... </Producer>
    <Producer name="producer-2"> ... </Producer>
    ...
    <UserService name="userService-1"> ... </UserService>
    <UserService name="userService-2"> ... </UserService>
    <UserService name="userService-3"> ... </UserService>
    ...
    <Binding name="binding-1"> ... </Binding>
    <Binding name="binding-2"> ... </Binding>
    ...
    <Binding name="myMainBinding-1"> ... </Binding>
    <Binding name="myMainBinding-2"> ... </Binding>
    ...
    ...
    <PMode name="MyPMode-1" binding="myMainBinding-1" />
    <PMode name="MyPMode-2" binding="myMainBinding-2" />
    <PMode name="MyPMode-3" />
    <Producer name="producer-10"> ... </Producer>
    <Producer name="producer-11"> ... </Producer>
    ...
    <UserService name="userService-10"> ... </UserService>
    <UserService name="userService-11"> ... </UserService>
    <Binding name="myMainBinding-3"> ... </Binding>
</PModes>
```
The following rule applies to all elements:

- Any element that is referred to using an attribute can also be written inline as a child within the element making the reference. For example, the `<PMode>` element has an attribute called "binding" that refers to some `<Binding>` element whose name attribute equals to the value of the binding attribute. Therefore, a `<Binding>` element could either reside as a child within the `<PMode>` element or it can reside outside the `<PMode>` element.

For example, the `<Leg>` element has an attribute called "userService" that refers to some `<UserService>` element. Therefore, a `<UserService>` element could be defined outside or it can also be defined as a child of the `<Leg>` element as illustrated by the following two examples:

```
...<Binding name="some-binding">
  <Orchestration mep="One-Way/Push">
    <Leg number="1" producer="producer-1">
      <UserService> ... </UserService>
    </Leg>
  </Orchestration>
</Binding>
...```

In the above example, the element `<UserService>` was defined inline as a child within the `<Leg>` element.

In the following example, an attribute reference is used and the `<UserService>` is defined outside:

```
...<Binding name="some-binding">
  <Orchestration mep="One-Way/Push">
    <Leg number="1" producer="producer-1" userService="userService-2">
      ... </UserService>
    </Leg>
  </Orchestration>
</Binding>
<UserService name="userService-2"> ... </UserService>
```

### 3.3.1 Error Reporting in PModes

How can the PMode XML representation specify error reporting policies for specific legs of a given orchestration? The `<Leg>` element has an optional child element called "ErrorAtSender" which specifies the behavior of error reporting if the error occurs at the sender side. The following example shows the use of such an element:

```
<PMode name="MyPmode">
  ...
</PMode>

<Binding name="mainBinding">
  <Orchestration mep="Two-Way/Push-And-Push">
    <Leg number="1" mpc="orders" userService="user-2" producer="producer-1">
      <Endpoint address="http://msh2.com/" soapVersion="1.2"/>
      <Authorization type="UsernameToken" username="bob" password="secret"/>
      <ErrorAtSender notifyProducer="true" notifyConsumer="true"
    </Leg>
    <Leg number="2" userService="user-1" producer="producer-2" mpc="billing"/>
  </Orchestration>
```
The above example describes a simple orchestration involving a single pair of MSHs where the core MEP of the orchestration is a Two-Way/Push-And-Push. The <ErrorAtSender> element is specified in the first leg of the orchestration (that is the first push from MSH-1 to MSH-2). This element says that if the error occurs at the sender side (which is MSH-1 trying to push the first user message to MSH-2), then the producer (which is some partyId behind MSH-1) should be notified and also the consumer (which is some partyId behind MSH-2) should be notified and finally that the two external entities defined by their address http://msh3.com and http://msh4.com should also be notified. The “reportTo” attribute may contain a comma-separated list of addresses representing some external entities that we desire to notify.

Similarly the <Leg> element can contain an optional child element called “ErrorAtReceiver” to specify how to report errors if a problem occurs while receiving the message. The “ErrorAtReceiver” element has the same attributes as the “ErrorAtSender” element.

### 3.3.2 Quality of Service in PModes

This subsection explains how to specify the quality of service for certain legs of a given orchestration. There are two quality-of-services: Reliability and Security.

There are only six QoS (Quality Of Service) in Reliability which are defined as follows:

```xml
<Reliability name="ORDER-ANON">
  <AtMostOnce>true</AtMostOnce>
  <AtLeastOnce>true</AtLeastOnce>
  <InOrder>true</InOrder>
  <AckReply>true</AckReply>
  <RetransmissionInterval>40000</RetransmissionInterval>
  <ExponentialBackoff>false</ExponentialBackoff>
  <MaximumRetransmissionCount>5</MaximumRetransmissionCount>
</Reliability>

<Reliability name="ORDER-CALLBACK">
  <AtMostOnce>true</AtMostOnce>
  <AtLeastOnce>true</AtLeastOnce>
  <InOrder>true</InOrder>
  <AckReply>Response</AckReply>
  <RetransmissionInterval>40000</RetransmissionInterval>
  <ExponentialBackoff>false</ExponentialBackoff>
  <MaximumRetransmissionCount>5</MaximumRetransmissionCount>
</Reliability>

<Reliability name="ORDER-POLL">
  <AtMostOnce>true</AtMostOnce>
  <AtLeastOnce>true</AtLeastOnce>
  <InOrder>true</InOrder>
  <AckReply>Poll</AckReply>
  <RetransmissionInterval>40000</RetransmissionInterval>
  <ExponentialBackoff>false</ExponentialBackoff>
  <MaximumRetransmissionCount>5</MaximumRetransmissionCount>
</Reliability>

<Reliability name="ANON">
  <AtMostOnce>true</AtMostOnce>
  <AtLeastOnce>true</AtLeastOnce>
  <InOrder>false</InOrder>
  <AckReply>true</AckReply>
  <RetransmissionInterval>40000</RetransmissionInterval>
  <ExponentialBackoff>false</ExponentialBackoff>
  <MaximumRetransmissionCount>5</MaximumRetransmissionCount>
</Reliability>

<!DOCTYPE swa:CallbackClass SYSTEM sample.swa.client.MyCallback.dtd>
<RetransmitCallbackClass>sample.swa.client.MyCallback</RetransmitCallbackClass>
```
If a Leg wants to specify a Reliability Quality Of Service, all what needs to be done is to refer to the quality by its name as an attribute of the <Leg> element, as follows:

```xml
<PMode name="MyPmode">
  ...

  <Binding name="mainBinding">
    <Orchestration mep="Two-Way/Push-And-Push">
      <Leg number="1" mpc="orders" userService="user-2" producer="producer-1"
        Reliability="POLL">
        ...
      </Leg>

    </Orchestration>
  </Binding>

</PMode>
```

In the example above, the leg specified that the reliability quality of service should be “POLL”. The “POLL” quality was defined above as:

```xml
<Reliability name="POLL">
  <AtMostOnce>true</AtMostOnce>
  <AtLeastOnce>true</AtLeastOnce>
  <InOrder>false</InOrder>
  <AckReply>Poll</AckReply>
  <RetransmissionInterval>40000</RetransmissionInterval>
  <ExponentialBackoff>false</ExponentialBackoff>
  <MaximumRetransmissionCount>5</MaximumRetransmissionCount>
</Reliability>
```

This means the following:

- “AtMostOnce" semantic is on (i.e. Duplicate Elimination),
- Ordering is turned off (no ordering is required on the receiver side),
- that “AtLeastOnce” is on (i.e. guaranteed delivery)
- The reliability acknowledgment will be pulled.
- The Reliability Module at the sender side will retransmit (resend) non-acknowledged messages every 40 seconds.
• ExponentialBackoff is turned off (if ExponentialBackoff was on, that means that the retransmission interval will double each time).

• The Reliability Module at the sender side will try resending non-acknowledged messages for a maximum of 5 tries.

If some value in a reliability quality of service needs to be customized, then the <Leg> element can include the reliability element as a child instead of referring to it from an attribute, as the following example shows:

```xml
<PMode name="MyPmode">
  ...
  <Binding name="mainBinding">
    <Orchestration mep="Two-Way/Push-And-Push">
      <Leg number="1" mpc="orders" userService="user-2" producer="producer-1">
        <Reliability name=" Polly">
          <AtMostOnce>true</AtMostOnce>
          <AtLeastOnce>true</AtLeastOnce>
          <InOrder>false</InOrder>
          <AckReply>Poll</AckReply>
          <RetransmissionInterval>40000</RetransmissionInterval>
          <ExponentialBackoff>false</ExponentialBackoff>
          <MaximumRetransmissionCount>10</MaximumRetransmissionCount>
        </Reliability>
      </Leg>
      ...
    </Orchestration>
    ...
  </Binding>
  ...
</PMode>
```

In the example above, the <Reliability> element was included as a child of the <Leg> element in order to customize the "MaximumRetransmissionCount" parameter.

TODO: Show how the Security configuration is expressed in the legs of a PMode.

3.3.3 Default Values

Many elements and attributes are optional in the XML representation of PModes. When these attributes or child elements are absent, certain default values are assumed. This subsection explains what those default values are:

• If the "mpc" attribute is missing from a <Leg> element, that leg is assumed that it will be using the default mpc. There is always a default mpc, even if it does not have a name.

• If the <Endpoint> child element is absent from a <Leg> element, and that leg is not the back channel of another leg, then it is assumed that the address that will be used will be taken from the WS-Addressing wsa:ReplyTo header. For example, the previous example represents a Two-Way/Push-And-Push orchestration between MSH-1 and MSH-2 where the second leg did not specify an <Endpoint> element (supposed to be the address of MSH-1). In this case, it is assumed that the first leg should have a WS-Addressing header wsa:ReplyTo that would indicate to MSH-2 where to send the asynchronous response.

• If the attribute "userService" is absent from the second leg of a two-way MEP (does not matter if it's sync, push-and-push, pull-and-push, or pull-and-pull), then it is recommended that the response user message contains the same ebMS headers as the corresponding request (except that the eb:From and eb:To parties would be switched, and that the response would contain a refToMessageId).
To Be Continued….
4 Message Packaging

4.1 New Signal Messages

4.1.1 Ping/Pong Signal Messages

4.1.2 Message Status Signal Messages

4.1.3 Flow Control Signal Messages

4.1.4 Agreement Exchange Signal Messages

In this subsection, we describe how an MSH may send a special signal message to another MSH to inquire about the PModes of that MSH. MSH can exchange their PModes and agree on a certain configuration on the fly by exchanging these new signal messages....

4.2 Message Bundling

4.2.1 User Message Unit Bundling

4.2.2 UserMessage/SignalMessage Combination
5 WS-Addressing Module
6 Payload Services Module

6.1 Large Attachments

6.2 Transformations

6.3 Compression

6.4 Encryption
7 Orchestration/Workflow
8 WSDL Interfaces

8.1 WSDL Interfaces to MSH Operations

8.2 WSDL Interface to PartyId
9 Gateway Component

This section describes the “Gateway Component” of an MSH. This component plays the role of the final consumer of user message and is highly configurable and flexible component. It can route user messages to various applications, webservices, workflows, ESBs, and so forth… It has a web interface throughout the administrator can configure it and register consumers (be it regular applications, webservices, an ESB, a workflow, etc…)
A. Communication-Protocol Bindings
B. WS-I Compliance
C. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Hamid Ben Malek, Fujitsu Computer Systems <hbenmalek@us.fujitsu.com>
Jacques Durand, Fujitsu Computer Systems <jdurand@us.fujitsu.com>
Ric Emery, Axway Inc <remery@us.axway.com>
Kazunori Iwasa, Fujitsu Limited <kiwasa@jp.fujitsu.com>
Ian Jones, British Telecommunications plc <ian.c.jones@bt.com>
Rajashekar Kailar, Centers for Disease Control and Prevention <kailar@bnetal.com>
Dale Moberg, Axway Inc <dmoberg@us.axway.com>
Sacha Schlegel, Individual <sacha@schlegel.li>
Pete Wenzel, Sun Microsystems <pete.wenzel@sun.com>

[Participant Name, Affiliation | Individual Member]
# E. Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Editor</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>08/26/2007</td>
<td>Hamid Ben Malek</td>
<td>Initial setup of table of contents, Complex MEP and PMode definitions, Messaging Model</td>
</tr>
</tbody>
</table>