OASIS ebXML Messaging Services 3.0 - Core

Review of Working Draft 06, 21 October 2005

Editors:
Hamid Ben Malek, Fujitsu Software Corp.

Introduction (lines 217-319)

- Remove line 218
- End of line 226, beginning of line 227: replace “XML framework” by “SOAP framework”.
- Remove the two paragraphs (lines 233 to 238), delete “For example,” at the beginning of line 239, and capitalize the letter I (It instead of it).
- Remove “– either at SOAP or ebMS level –“ in line 248 (because it does not any meaning).
- The last part of the sentence in line 248 (“and experiencing restrictions in initiating a message transfer”) is not clear and suggest to delete it.
- Replace the beginning of line 250 (“The choice of an XML messaging framework…”) by “The choice of a SOAP messaging framework…”
- In line 254, replace “communications protocol” by “transport protocol”.
- In line 255, add SMTP to the list of protocol supported in Part 1 of ebMS-3. In other terms, replace “This version of the specification provides bindings to HTTP (Core Messaging Part), SMTP and FTP (advanced Messaging Part)” by “This version of the specification provides bindings to HTTP, SMTP (Core Messaging Part), FTP (advanced Messaging Part)”.
- In line 272, replace “the domains of reliability and security” by “the domains of reliability, security, and addressing”.
- Remove paragraph (lines 289 to 290).
- Remove section (lines 291 to 294)
- Remove section (lines 295 to 319): Rules for normative interpretation should part of a “Conformance” chapter (the “Conformance” chapter should be an Appendix, and not a real chapter of the core spec).

Messaging Model (lines 320 to 563)

- Definition of ebMS MSH in line 326 is using the term “ebMS message” while this term is defined only at later time (in line 344).
- In lines 338-339: “… a programming object combining payload and header elements, a SOAP message, etc.” A SOAP message is also a programming object, so the sentence is not well formed because it does not say much… There is no SOAP message concept in software that is not an object.
- Figure 1 on page 11: The picture is misleading. A message producer could be a queue or some storage area, and the “submit” operation in this case is an act initiated by the MSH itself and consists in fetching (or picking up) a message from the queue or storage area. The arrow representing the “submit” operation on the figure suggests that it is the producer who invokes this operation on the MSH. The submit operation could be invoked by the MSH itself (the MSH could be configured to periodically fetch messages from a queue or storage space).
- Lines 347-351: The definitions provided for the terms “ebMS Signal Message” and “ebMS User Message” are using semantics only. A definition should only be based on syntax. Semantics could be added to a definition at a second step only to add illustration and provide intuition (not as part of the definition itself). For example, the term “ebMS Signal Message” should be defined as being a SOAP message that contains at least one eb:Message/eb:SignalMessage element inside its SOAP header block. An ebMS User Message should be defined as a SOAP message that contains only one (one and only one instance of) element eb:Message/eb:UserMessage within its SOAP header and does not contain the element eb:Message/eb:SignalMessage.
- Lines 347-351: Maybe should also provide a new term for an ebMS message that contains many user messages (user messages together with their payloads all bundled in one SOAP message). And another term to call a SOAP message that is both a user message and a signal message.
- In lines 360-362: It talks about the “receiving” role that could be played by an MSH. How about when sending an asynchronous response (a response correlated to a previous request)? Is the MSH considered in the “receiving” role when it sends an asynchronous response?
- Lines 371-373: The “Notify” operation does not always exist. Consider the deployment scenario where an MSH is a JAX-RPC handler among other handlers. In this deployment scenario, the “producer” becomes the JAX-RPC handler that is deployed just before the MSH (that is the handler that processes messages before the MSH does). In this example, the MSH has no way of “notifying” the producer as this later does not even exist most of the time (it is instantiated and called by the stack runtime only

when a message is to be processed, afterwards it is destroyed by the runtime).

- Lines 407 to 410 should be written as a footnote.
- In line 413, RMP is used instead of MSH. It should be MSH instead of RMP. The same thing is happening at lines 422 and 424.
- From line 424 to 427, the definition is general enough to seem to include the asynchronous Request/Response MEP. Is this true? If yes, it should be clearly stated.
- Line 439: “This MEP involves a single ebMS user message.” This should also include a batched message (many user messages bundled together in one SOAP message). The line should be modified to something like “This MEP involves a single ebMS user message or a batched message.”
- Line 442: “… the latter must NOT contain an ebXML user message.” This statement is not very precise. Consider the case where a response SOAP message contains data (i.e. SOAP envelope not empty) and yet the response does not contain the “eb” namespace in its SOAP header (in other terms, the response is not an ebMS User Message). Furthermore, the term “ebXML user message” has never been defined.
- Figure 2: The “Send” operation is not only about a “user message”. A batched user message should also be mentioned.
- Line 460: “The ebMS reply refers to the message ID of the ebMS request via the eb:RefToMessageId header element”. Is this really necessary? Normally we don’t need to correlate a response to a request when a synchronous Req/Resp MEP is used. Only in the case of an asynchronous Request/Reply that the correlation might be necessary. The described MEP is not about the asynchronous Request/Response.
- Line 462: “The eb:syncresp attribute MUST be set to ‘true’ on the request user message.” The “MUST” is maybe a too strong statement. There are situations when an MSH is deployed as one JAX-RPC handler among others, and in this case, the MSH has no knowledge whether the MEP is a SOAP Oneway or a Req/Resp. The spec should rather say something like: “The MSH MUST set the eb:syncresp attribute to true only when it knows that an ebMS Req/Resp MEP is being used.
- Line 474: “mode of transfer…” Mode of transfer is a term that has not been introduced before (only MEPs have been introduced without using the expression “mode of transfer”.
- Line 475: “… is only dependent on implementations…” It does also depend on the “transfer mode”, not just implementations.
- Lines 482-483: This is not very clear. Persistence maybe understood, but not recovery and security.
- Lines 484-488: Message Queues or Message Bags are part of the definition of a pipe. In other terms, each pipe is associated with a queue or bag of messages (the queue or bag may be logical only or of a real

persisted nature). What is out of scope of this spec is how the bag/queue is implemented and associated with a pipe. The queue/bag concept exists and is associated with a pipe (one-to-one relationship between queues and pipes). The sentence “Support for assigning…” in line 484 suggests that associating a queue/bag with a pipe is optional.

- Line 490: “… ore more generally an IRI.” There is no need for a pipe’s name to be an IRI.
- End of line 495, beginning of line 496: “if it is defined” This comment does not make any sense. When we talk about the default pipe, it is assumed that we are talking about a pair of MSHs one of which we know the address of. If both MSH do not have addresses or don’t know the address of the other, there is no default pipe in between.
- Lines 536 and 537: The request/response MEP talked about may be synchronous or asynchronous. It is better to state it clearly that both cases are considered.
- Line 542: How about the first leg of a PullRequest MEP? What pipe does it use?
- Line 552: “The RIT mode is intended…” Need to stick to one terminology only (Pull or RIT).

Concept of Operation (lines 564 to 691)

- Line 574: “It may be construed…” Should be “constructed” not “construed”.
- Line 582: “… into five main entities…” There are six operation contexts not five. And if we add an operation context for Addressing, the total would be seven instead.
- The following comments apply to section 3 (section 3 starts at line 564): First, there is a terminology issue. Need a name for the set of operation contexts. For example, “Operation Mode” could be used to the collection of the operation contexts (the six operation contexts). Another term for “Operation Mode” could be “ebMS Agreement” for example, though “Operation Mode” may be better. Second, replace the title of section 3 (title is “Concept of Operation”) by “Operation Mode” or ebMS Agreement (whatever term that is chosen to denote the collection of the six operation contexts).
- Line 627: “The Default Operation mode between two MSHs is an operation context characterized by” This should be “The Default Operation Mode between two MSHs is an operation mode characterized by”.
- Figure 5 is an obsolete figure that is inherited from ebMS version 2. In order to keep that figure, the following corrections must be made to it: (1) remove the layer called “Message Transceiver/Transport Liaison”. (2) the

layer “Digital Signature/Cryptography processing” need to be replaced by a layer that has “WSS Processing” for title instead. (3) change the layer “ebMS Packaging & Payload Services” to just “ebMS Packaging”. (4) Replace the layer “Authentication, Authorization & Non-Repudiation” by a layer with “Addressing” as title. (5) Replace the layer “ebXML Application” by “Application or SOAP processor”.

- Remove lines 654 to 657.
- Line 658: “ebXML Message …” The term ebMS Message has been defined and used before but not the term ebXML Message. Need to stick to one terminology only.
- Remove lines 664 to 666.
- Line 670: section 3.4 (Examples of Supported Topologies) should be a subsection of “Messaging Model” section.

Message Packaging (lines 692 to 1300)

- Lines 695, 700, 703: the term “ebXML Message” is used. Need to stick to one terminology only (either ebMS Message or ebXML Message).
- Line 715 uses the term “ebXML User Message”. Again, need to stick to one term only (ebMS User Message or ebXML User Message).
- After line 714, there should be a listing (sample example) of an ebMS User Message, another one for a Signal Message illustrating a PullRequest, and a third listing for an example of an Error Message (a particular case of a Signal Message). Examples are very important and should be given right away. This is the first thing implementers look at. For example, page 31 provides a listing of an example of a User Message. This kind of listing should be given right away after line 714 and should be well indented.
- Figure 7 (page 25): eb:ErrorList should be replaced by eb:Error. The packaging of errors has changed from ebMS-2 to ebMS3. In ebMS-2, errors were embedded in an outer element called “eb:ErrorList”. In ebMS-3, there is no outer element eb:ErrorList.
- Lines from 745 to 844 should be removed.
- Lines from 868 to 882 should be removed.
- The listing example on page 31 should be well indented.
- The listing example (lines 1110-1114) should be well indented.
- The listing example (lines 1119-1123) should be well indented.
- Example on lines 1220-1224 should be well indented.
- The packaging synctax of the eb:PayloadInfo element need to be simplified. The following is suggested:

```
<eb:PayloadInfo>
```

```
    <eb:Partref eb:id="…" eb:idref="…">
      <eb:Description>…</eb:Description>
    </eb:Partref>
</eb:PayloadInfo>
```

The child eb:Schema is removed and the xml:lang attribute is removed from the eb:Description element.

Error Handling Module (lines 2039 to 2281)

This section on the Error Module introduced a new concept called “Error Reporting”. However this term was not well defined and did not cover other concepts that the Error Module intended to introduce. Some parts were intended as definitions (lines 2059-2075), however, they were perceived differently when read by other people. All of this makes lots of confusion for a new implementer reading the Error Module section. The eb:Error element introduced too many attributes and ended up being overly complex and overloaded. For example, to mention a few, the eb:Error/eb:errorDetail element was not a realistic option as it would be impossible for most (if not all) implementations to include such error detail since an exception details can hardly be captured and then included in the SOAP header packaging (this would assume total control of the stack runtime for an MSH, an assumption that is almost impossible. Therefore, the section on Error Module needed to be re-written in a way to address some of the issues and confusions that were unintentionally introduced in the way it was drafted. The new version of the section on Error Module is written as a separate document and not included in this review.