I suggest that the current sections 5 and 6 be deleted. Section 6.3 should be moved into section 4.4. The following text should be used as a new section 5 and other sections should be re-numbered accordingly.

5. Concept of Operation for Message Servicing

5.1 Scope

The ebXML Message Service [ebMS] defines the message enveloping and header document schema used to transfer ebXML messages over a communications protocol such as HTTP or SMTP and the behavior of the software that sends and receives ebXML messages[1]. The ebMS is defined as a set of layered extensions to the base Simple Object Access Protocol [SOAP] and SOAP Messages with Attachments [SOAPATTACH] specifications. The ebMS provides security and reliability features necessary to support international electronic business. These security and reliability features are not provided in the SOAP or SOAPATTACH specifications.

The ebXML infrastructure is composed of several independent, but related, components. Specifications for the individual components are fashioned as stand-alone documents. The specifications are totally self-contained; nevertheless, design decisions within one document can and do impact the other documents. Considering this, the ebMS is a closely coordinated definition for an ebXML message service handler [MSH].

The ~~ebXML Messaging Service [~~ebMS~~]~~ component provides the message packaging, routing, and transport facilities for the ebXML infrastructure. The ebMS is not defined as a physical component, but rather as an abstraction of a[2] process. An implementation of this specification could be delivered as a wholly independent software application or an integrated component of some larger business process.

5.1 Background and Objectives

Traditional business information exchanges have conformed to a variety of standards-based syntaxes. These exchanges were largely based on electronic data interchange (EDI) standards born out of mainframe and batch processing. Some of the standards defined bindings to specific communications protocols. These EDI techniques worked well; however, they were difficult and expensive to implement. Therefore, use of these systems was normally limited to large enterprises that possessed mature information technology capabilities.

The proliferation of XML-based business interchanges served as the catalyst for defining a new global paradigm that ensured all business activities, regardless of size, could engage in electronic business activities. The prime objective of ebMS is to facilitate the exchange of electronic business messages within an XML framework. Business

[1] David Burdett comment, phrase added
[2] Chris Ferris comment, phrase added

messages, identified as the 'payloads' of the ebXML messages, are not necessarily expressed in XML. XML-based messages, as well as traditional EDI formats, are transported by the ebMS. Actually, the ebMS payload can take any digital form—XML, ASC X12, HL7, AIAG E5, database tables, or binary image files.

The ebXML architecture requires that the ebXML ~~transport mechanism be free to communicate~~Message Service protocol be capable of being carried over any available transport protocol. Therefore, the ebMS does not mandate use of a specific transport protocol. This version of the specification provides bindings to HTTP and SMTP, but other protocols can and reasonably will be used[3].

The ebXML Requirements Specification [ebRS] mandates the need for secure, reliable communications. The ebXML work focuses on leveraging existing and emerging technology—attempts to create new protocols are discouraged. Therefore, the ebMS defines security within the context of existing security standards and protocols. Those requirements that can be satisfied with existing standards are specified in the ebMS, others must be deferred until new ~~technology~~ technologies or standards are available, for example encryption of individual message header elements[4].

Reliability requirements defined in the ebRS relate to delivery of ebXML messages over the communications channels. The ebMS provides mechanisms to satisfy the ebRS requirements. The reliable messaging elements of the ebMS supply reliability to the communications layer; they are not intended as business-level [5]acknowledgments to the applications that are supported by the ebMS. This is an important distinction. Business processes often [6]anticipate responses to messages they generate. The responses may[7] take the form of a simple acknowledgment of message receipt by the application that received the message [8]or a companion message that reflects action on the original message. Those messages are outside of the requirements defined for the MSH. The acknowledgment defined in this specification does not indicate that the payload of the ebXML message was syntactically correct. It does not acknowledge the accuracy of the payload information. It does not indicate business acceptance of the information or agreement with the content of the payload. The ebMS is designed to provide the sender ~~is~~ with the confidence that the receiving MSH has received the message intact[9].

The underlying architecture of the MSH assumes that messages are exchanged between two ~~ebXML~~ebMS-compliant MSH nodes[10]. This pair of MSH nodes provides a hop-to-hop model that is extended as required to support a multi-hop environment. The multi-hop environment allows for the final destination of the message to be an intermediary[11]

---

[3] All changes to this paragraph were derived from comments from Chris Ferris
[4] These changes reflect the combined contribution of David Burdett and Arvola Chan
[5] Arvola Chan change
[6] David Burdett change
[7] Arvola Chan change
[8] David Burdett change
[9] David Burdett and Arvola Chan both made recommended changes to this sentence
[10] Chris Ferris requested the addition of the term NODE
[11] Change requested by Bruec Pedretti to clarify sentence

MSH other than the 'receiving MSH' identified by the original sending MSH. The ~~ebXML~~ ebMS architecture assumes that the sender of the message MAY be unaware of the specific path used to deliver a message. However, it MUST be assumed that the original sender has knowledge of the final recipient of the message and the first of one or more intermediary hops. The architecture also supports a business requirement to specify an ordered-set of discrete parties to whom a message is routed. The multi-hop and ordered-set options obfuscate the acknowledgment message identified in the paragraph above. It is understood that the acknowledgment does not assure delivery of the message to the final destination, only to the receiving MSH of the MSH pair.

The MSH supports the concept of 'quality of service.' The degree of service quality is controlled by ~~a contract~~ an agreement[12] existing between the parties directly involved in the message exchange. In practice, multiple ~~contracts~~ agreements may be required between the two parties. The ~~contracts~~ agreements ~~would~~ might be tailored to the particular needs to the business exchanges. For instance, a set of business partners may have a contract that defines the message exchanges related to buying products from a domestic facility and another that defines the message exchanges for buying from an overseas facility. Alternatively, the partners might agree to follow the agreements developed by their trade association.[13] Multiple ~~contracts~~ agreements may also exist between the various parties that handle the message from the original sender to the final recipient. These ~~contracts~~ agreements could include:[14]

- ~~-~~an ~~contract~~ agreement between the MSH at the message origination site and the MSH at the f~~F~~inal destination; ~~and~~
- ~~contracts~~ agreements between the MSH at the message origination site and the MSH acting as an intermediary; and ~~and~~
- ~~a contract~~an agreement between the MSH at the final destination and the MSH acting as an intermediary. There would, of course, be ~~contracts~~ agreements between any additional intermediaries; however, the originating site MSH and final destination MSH MAY have no knowledge of these ~~contracts~~agreements.

The important point is that an ~~ebXML~~ebMS-compliant MSH shall respect the in-force ~~contracts~~ agreements between itself and any other ~~ebXML~~ebMS-compliant MSH with which it communicates. In broad terms, these ~~contracts~~ agreements are expressed as Collaborative Profile Agreements (CPA). This specification identifies the information that must be agreed. It does not specify ~~T~~the method ~~of~~ or form used to create and maintain these agreements ~~is beyond the scope of this specification~~.[15] It is assumed that, in practice, the actual content of the contracts may be contained in

[12] David Burdett objected to the term 'contract' as being too strong, all references to contract were changed to agreement
[13] Sentence added by David Burdett
[14] Bruce Pedretti recommended listing the specific examples to simplify the sentence
[15] Sentence change suggested by David Burdett

initialization/configuration files, databases, or XML documents that comply with the [ebCPP] specification.

5.3 Operational policies and constraints

The ebMS is a service that is logically positioned between one or more business applications and a communications service. This requires the definition of an abstract service interface between the business applications and the MSH. This document acknowledges the interface, but does not provide a definition for the interface. Future versions of the ebMS MAY ~~dictate~~ define[16] the service interface structure. Bindings to two communications protocols are defined in this document; however, the MSH is specified independent of any communications protocols. While early work focuses on HTTP for transport, no preference is being provided to this protocol. Other protocols may be used and future versions of the specification may provide details related to those protocols.

The ebMS relies on external ~~business and~~ [17]communications configuration information. This information is determined either through defined business processes or trading partner agreements. These data are captured for use within a collaboration protocol profile [CPP] or collaboration protocol agreement [CPA]. The ebXML Collaborative-Protocol Profile and Agreement Specification [ebCPP] provides definitions for the information constituting the agreements. The ebXML architecture defines the relationship between this component of the infrastructure and the ebMS. As regards the MSH, the information composing a CPP/CPA must be available to support normal operation. However, the method used by a specific implementation of the MSH does not mandate the existence of a discrete instance of a CPA. The CPA is expressed as an XML document. Some implementation may elect to populate a database with the information from the CPA and then use the database. This specification does not prescribe how the CPA information is derived, stored, or used: it only states that specific information items must be available for the MSH for successful operations.

This specification MUST distinguish between acknowledgments and delivery receipts. This specification restricts the term acknowledge to mean recognition that a message has been accepted into the persistent storage of the receiving MSH—not necessarily the final destination. The delivery receipt is understood by the MSH to mean that the MSH at the final destination has accepted the message into its persistent storage.

5.4 Modes of operation

This specification does not mandate how the MSH will be installed within the overall ebXML framework. It is assumed that some MSH implementations will not implement all functionality defined in this specification. For instance, a set of trading partners may

[16] David Burdett thought that 'dictate' was too strong and requested a substitution using define
[17] David Burdett recommends that only communications information be referred to here, delete the reference to business

not require reliable messaging services; therefore, no reliable messaging capabilities exist within their MSH. But, all MSH implementations shall comply with the specification with regard to the functions supported in the specific implementation and provide error notifications for functionality that has been requested but is not supported.[18] Documentation for an MSH implementation SHALL identify all ebMS requirements that are not satisfied in the implementation.

The *ebXML Message Service* may be conceptually broken down into following three parts: (1) an abstract *Service Interface*, (2) functions provided by the ~~Message Service Handler (~~MSH~~)~~, and (3) the mapping to underlying transport service(s).

Figure 1 depicts a logical arrangement of the functional modules that exist within one possible implementation of the *ebXML Message Services* architecture. These modules are arranged in a manner to indicate their inter-relationships and dependencies.

- **Header Processing** - the creation of the ebXML Header elements for the *ebXML Message* uses input from the application, passed through the Message Service Interface, information from the *Collaboration Protocol Agreement* ~~(CPA defined in [EBXMLTP])~~ that governs the message, and generated information such as digital signature, timestamps and unique identifiers.
- **Header Parsing** - extracting or transforming information from a received ebXML Header element into a form that is suitable for processing by the MSH implementation.
- **Security Services** - digital signature creation and verification, authentication and authorization. These services MAY be used by other components of the MSH including the Header Processing and Header Parsing components.
- **Reliable Messaging Services** - handles the delivery and acknowledgment of ebXML Messages sent with ***deliverySemantics*** of ***OnceAndOnlyOnce***. The service includes handling for persistence, retry, error notification and acknowledgment of messages requiring reliable delivery.
- **Message Packaging** - the final enveloping of an *ebXML Message* (ebXML header elements and payload) into its SOAP Messages with Attachments [SOAPATTACH] container.
- **Error Handling** - this component handles the reporting of errors encountered during MSH or Application processing of a message.
- **Message Service Interface** - an abstract service interface that applications use to interact with the MSH to send and receive messages and which the MSH uses to interface with applications that handle received messages.

[18] additional clause recommended by David Burdett

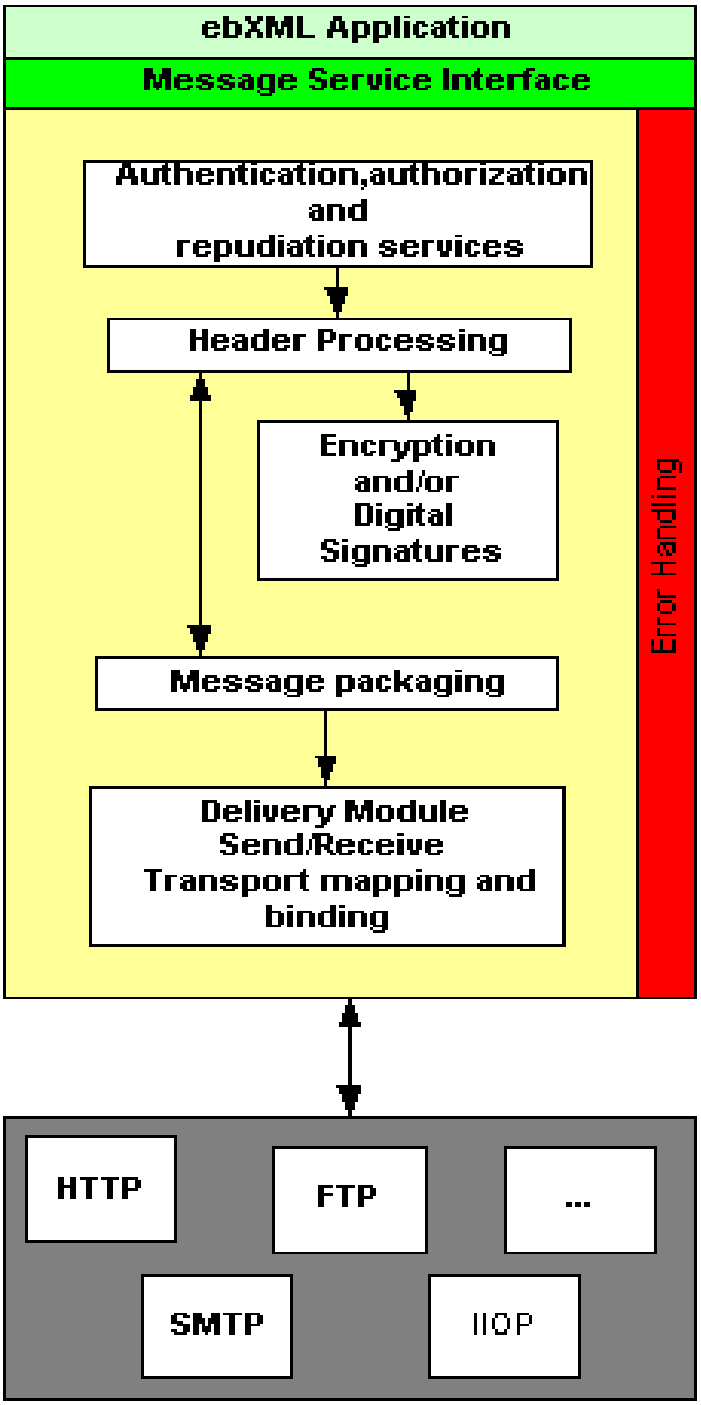

Figure -1 Typical Relationship between ebXML Message Service Handler Components

5.5 User classes and others involved