



Creating A Single Global Electronic Market

Collaboration-Protocol Profile and Agreement Specification Version 0.95

ebXML Trading-Partners Team

04/19/01 5:33 PM

1 Status of this Document

This document specifies an ebXML WORK IN PROGRESS for the eBusiness community.

Distribution of this document is unlimited.

The document formatting is based on the Internet Society's Standard RFC format.

This version:

<http://www.ebxml.org/specdrafts/cpa-cpp-spec-0.95.pdf>

Latest version:

<http://www.ebxml.org/specdrafts/cpa-cpp-spec-0.95.pdf>

Previous version:

http://www.ebxml.org/project_teams/trade_partner/private/cpa-cpp-spec-0.93.pdf

2 ebXML Participants

The authors wish to recognize the following for their significant participation to the development of this document.

David Burdett, CommerceOne
Tim Chiou, United World Chinese Commercial Bank
Chris Ferris, Sun
Scott Hinkelman, IBM
Maryann Hondo, IBM
Sam Hunting, ECOM XML
John Ibbotson, IBM
Kenji Itoh, JASTPRO
Ravi Kacker, eXcelon Corp.
Thomas Limanek, iPlanet
Daniel Ling, VCHEQ
Henry Lowe, OMG
Dale Moberg, Cyclone Commerce
Duane Nickull, XMLGlobal Technologies
Stefano Pogliani, Sun
Rebecca Reed, Mercator
Karsten Riemer, Sun
Marty Sachs, IBM
Yukinori Saito, ECOM
Tony Weida, Edifecs

3 Table of Contents

1	Status of this Document	1
2	ebXML Participants	2
3	Table of Contents	3
4	Introduction.....	5
4.1	Summary of Contents of Document.....	5
4.2	Document Conventions.....	5
4.3	Version of the Specification.....	6
4.4	Definitions.....	6
4.5	Audience	6
4.6	Assumptions.....	6
4.7	Related Documents	6
5	Design Objectives	8
6	System Overview	9
6.1	What This Specification Does.....	9
6.2	Forming a CPA from Two CPPs.....	10
6.3	How the CPA Works.....	12
6.4	Where the CPA May Be Implemented.....	13
6.5	Definition and Scope.....	14
7	CPP Definition	15
7.1	Globally-Unique Identifier of CPP Instance Document.....	16
7.2	SchemaLocation Attribute	16
7.3	CPP Structure.....	17
7.4	CollaborationProtocolProfile element.....	17
7.5	PartyInfo Element	18
7.5.1	PartyId element	18
7.5.2	PartyRef element.....	19
7.5.3	CollaborationRole element.....	20
7.5.4	ProcessSpecification element.....	22
7.5.5	Role element	25
7.5.6	ServiceBinding element	25
7.5.7	Override element.....	26
7.5.8	Certificate element	27
7.5.9	DeliveryChannel element.....	28
7.5.10	Characteristics element	29
7.5.11	Transport element.....	31
7.5.12	Transport protocol.....	31
7.5.13	Endpoint element	33
7.5.14	Transport protocols	33
7.5.15	Transport security.....	35
7.6	DocExchange Element.....	36
7.6.1	docExchangeId attribute.....	37
7.6.2	ebXMLBinding element.....	37
7.6.3	version attribute.....	37
7.6.4	ReliableMessaging element.....	37
7.6.5	NonRepudiation element.....	39
7.6.6	DigitalEnvelope element	40
7.6.7	NamespaceSupported element	41
7.7	Packaging element	41
7.7.1	ProcessingCapabilities element.....	42
7.7.2	SimplePart	42
7.7.3	SimplePart element	42
7.7.4	CompositeList element.....	42

7.8 ds:Signature element	44
7.9 Comment Element.....	44
8 CPA Definition	46
8.1 CPA Structure	46
8.2 CollaborationProtocolAgreement Element	46
8.3 Status Element.....	47
8.4 CPA Lifetime	48
8.4.1 Start element	48
8.4.2 End element.....	48
8.5 ConversationConstraints Element	49
8.5.1 invocationLimit attribute.....	49
8.5.2 concurrentConversations attribute.....	49
8.6 PartyInfo Element	50
8.6.1 ProcessSpecification element.....	50
8.7 ds:Signature Element	50
8.7.1 Persistent Digital Signature.....	51
8.8 Comment element	52
8.9 Composing a CPA from Two CPPs	52
8.9.1 ID Attribute Duplication	53
8.10 Modifying Parameters of the Process-Specification Document Based on Information in the CPA	53
9 References.....	54
10 Conformance	56
11 Disclaimer	57
12 Contact Information	58
Copyright Statement	59
Appendix A Example of CPP Document (Non-Normative).....	60
Appendix B Example of CPA Document (Non-Normative).....	62
Appendix C DTD Corresponding to Complete CPP/CPA Definition (Normative).....	65
Appendix D XML Schema Document Corresponding to Complete CPP and CPA Definition (Normative)	68
Appendix E Formats of Information in the CPP and CPA (Normative)	74
Appendix F Composing a CPA from Two CPPs (Non-Normative).....	75

4 Introduction

4.1 Summary of Contents of Document

As defined in the ebXML Business Process Specification Schema[ebBPSS], a *Business Partner* is an entity that engages in *Business Transactions* with another *Business Partner(s)*. Each *Partner's* capabilities (both commercial/*Business* and technical) to engage in electronic *Message* exchanges with other *Partners* MAY be described by a document called a *Trading-Partner Profile (TPP)*. The agreed interactions between two *Partners* MAY be documented in a document called a *Trading-Partner Agreement (TPA)*. A *TPA* MAY be created by computing the intersection of the two *Partners' TPPs*.

The *Message-exchange* capabilities of a *Party* MAY be described by a *Collaboration-Protocol Profile (CPP)* within the *TPP*. The *Message-exchange* agreement between two *Parties* MAY be described by a *Collaboration-Protocol Agreement (CPA)* within the *TPA*. Included in the *CPP* and *CPA* are details of transport, messaging, security constraints, and bindings to a *Process-Specification* document that contains the definition of the interactions between the two *Parties* while engaging in a specified electronic *Business Collaboration*.

This specification is a draft standard for trial implementation. This specification contains the detailed definitions of the *Collaboration-Protocol Profile (CPP)* and the *Collaboration-Protocol Agreement (CPA)*.

This specification is a component of the suite of ebXML specifications. An overview of the ebXML specifications and their interrelations can be found in the ebXML Technical Architecture Specification[ebTA].

This specification is organized as follows:

- Section 5 defines the objectives of this specification.
- Section 6 provides a system overview.
- Section 7 contains the definition of the *CPP*, identifying the structure and all necessary fields.
- Section 8 contains the definition of the *CPA*.
- The appendices include examples of XML *CPP* and *CPA* documents (non-normative), the DTD (normative), an XML Schema document equivalent to the DTD (normative), formats of information in the *CPP* and *CPA* (normative), and composing a *CPA* from two *CPPs* (non-normative).

4.2 Document Conventions

Terms in *Italics* are defined in the ebXML Glossary of Terms[ebGLOSS]. Terms listed in ***Bold Italics*** represent the element and/or attribute content of the XML *CPP* or *CPA* definitions.

In this specification, indented paragraphs beginning with "NOTE:" provide non-normative

43 explanations or suggestions that are not required by the specification.

44
45 References to external documents are represented with BLOCK text enclosed in brackets, e.g.
46 [RFC2396]. The references are listed in Section 9, "References".

47
48 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD
49 NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be
50 interpreted as described in [RFC 2119].

51
52 NOTE: Vendors should carefully consider support of elements with cardinalities (0 or 1)
53 or (0 or more). Support of such an element means that the element is processed
54 appropriately for its defined function and not just recognized and ignored. A given *Party*
55 might use these elements in some *CPPs* or *CPAs* and not in others. Some of these elements
56 define parameters or operating modes and should be implemented by all vendors. It might
57 be appropriate to implement optional elements that represent major run-time functions,
58 such as various alternative communication protocols or security functions, by means of
59 plug-ins so that a given *Party* MAY acquire only the needed functions rather than having
60 to install all of them.

61

62 **4.3 Version of the Specification**

63 Whenever this specification is modified, it SHALL be given a new version number. The value
64 of the *version* attribute of the *Schema* element of the XML Schema document SHALL be equal
65 to the version of the specification.

66

67 **4.4 Definitions**

68 Technical terms in this specification are defined in the ebXML Glossary[ebGLOSS].

69

70 **4.5 Audience**

71 One target audience for this specification is implementers of ebXML services and other
72 designers and developers of middleware and application software that is to be used for
73 conducting electronic *Business*. Another target audience is the people in each enterprise who are
74 responsible for creating *CPPs* and *CPAs*.

75

76 **4.6 Assumptions**

77 It is expected that the reader has an understanding of [XML] and is familiar with the concepts of
78 electronic *Business* (eBusiness).

79

80 **4.7 Related Documents**

81 Related documents include ebXML Specifications on the following topics:

- 82 • ebXML Technical Architecture Specification[ebTA]
- 83 • ebXML *Message* Service Specification[ebMS]
- 84 • ebXML Business Process Specification Schema[ebBPSS]

- 85 • ebXML Glossary [ebGLOSS]
- 86 • ebXML Core Component and Business Document Overview[ccOVER]
- 87 • ebXML Registry Services Specification[ebRS]

88

89 See Section 9 for the complete list of references.

90

91 5 Design Objectives

92 The objective of this specification is to ensure interoperability between two *Parties* even though
93 they MAY procure application software and run-time support software from different vendors.
94 The *CPA* defines the way two *Parties* will interact in performing the chosen *Business*
95 *Collaboration*. Both *Parties* SHALL use identical copies of the *CPA* to configure their run-time
96 systems. This assures that they are compatibly configured to exchange *Messages* whether or not
97 they have obtained their run-time systems from the same vendor. The configuration process
98 MAY be automated by means of a suitable tool that reads the *CPA* and performs the
99 configuration process.

100
101 In addition to supporting direct interaction between two *Parties*, this specification MAY also be
102 used to support interaction between two *Parties* through an intermediary such as a portal or
103 broker. In this initial version of this specification, this MAY be accomplished by creating a *CPA*
104 between each *Party* and the intermediary in addition to the *CPA* between the two *Parties*. The
105 functionality needed for the interaction between a *Party* and the intermediary is described in the
106 *CPA* between the *Party* and the intermediary. The functionality needed for the interaction
107 between the two *Parties* is described in the *CPA* between the two *Parties*.

108
109 It is an objective of this specification that a *CPA* SHALL be capable of being composed by
110 intersecting the respective *CPPs* of the *Parties* involved. The resulting *CPA* SHALL contain
111 only those elements that are in common, or compatible, between the two *Parties*. Variable
112 quantities, such as number of retries of errors, are then negotiated between the two *Parties*. The
113 design of the *CPP* and *CPA* schemata facilitates this composition/negotiation process. However,
114 the composition and negotiation processes themselves are outside the scope of this specification.
115 Appendix F contains a non-normative discussion of this subject.

116
117 It is a further objective of this specification to facilitate migration of both traditional EDI-based
118 applications and other legacy applications to platforms based on the ebXML specifications. In
119 particular, the *CPP* and *CPA* are components of the migration of applications based on the X12
120 838 Trading-Partner Profile to more automated means of setting up *Business* relationships and
121 doing *Business* under them.

122 6 System Overview

123 **6.1 What This Specification Does**

124 The exchange of information between two *Parties* requires each *Party* to know the other *Party's*
 125 supported *Business Collaborations*, the other *Party's* role in the *Business Collaboration*, and the
 126 technology details about how the other *Party* sends and receives *Messages*. In some cases, it is
 127 necessary for the two *Parties* to reach agreement on some of the details.

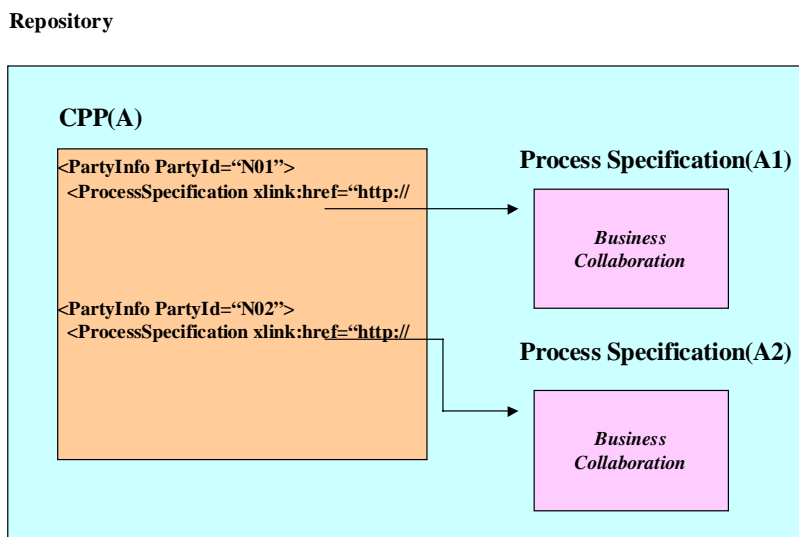
128
 129 The way each *Party* can exchange information, in the context of a *Business Collaboration*, can
 130 be described by a *Collaboration-Protocol Profile (CPP)*. The agreement between the *Parties* can
 131 be expressed as a *Collaboration-Protocol Agreement (CPA)*

132
 133 To enable *Parties* wishing to do *Business* to find other *Parties* that are suitable *Business*
 134 *Partners*, *CPPs* MAY be stored in a repository such as is provided by the ebXML
 135 Registry[ebRS]. Using a discovery process provided as part of the specifications of a repository,
 136 a *Party* MAY then use the facilities of the repository to find *Business Partners*.

137
 138 The document that defines the interactions between two *Parties* is an [XML] document called a
 139 *Process-Specification* document that conforms to the ebXML Business Process Specification
 140 Schema[ebBPSS]. The *CPP* and *CPA* include references to this *Process-Specification*
 141 document. The *Process-Specification* document MAY also be stored in a repository such as the
 142 ebXML Registry.

143
 144 Figure 1 illustrates the relationships between a *CPP* and two *Process-Specification* documents,

Figure 1: Structure of CPP & Business Process Specification in an ebXML Registry



145 A1 and A2, in an ebXML Registry. On the left is a *CPP*, A, which includes information about
146 two parts of an enterprise that are represented as different *Parties*. On the right are shown two
147 *Process-Specification* documents. Each of the *PartyInfo* elements in the *CPP* contains a
148 reference to one of the *Process-Specification* documents. This identifies the *Business*
149 *Collaboration* that the *Party* can perform.

150
151 This specification defines the markup language vocabulary for creating electronic *CPPs* and
152 *CPAs*. *CPPs* and *CPAs* are [XML] documents. In the appendices of this specification are a
153 sample *CPP*, a sample *CPA*, the DTD, and the corresponding XML Schema document.

154
155 The *CPP* describes the capabilities of an individual *Party*. A *CPA* describes the capabilities that
156 two *Parties* have agreed to use to perform a particular *Business Collaboration*. These *CPAs*
157 define the "information technology terms and conditions" that enable *Business* documents to be
158 electronically interchanged between *Parties*. The information content of a *CPA* is similar to the
159 information-technology specifications sometimes included in Electronic Data Interchange (EDI)
160 *Trading Partner Agreements (TPAs)*. However, these *CPAs* are not paper documents. Rather,
161 they are electronic documents that can be processed by computers at the *Parties'* sites in order to
162 set up and then execute the desired *Business* information exchanges. The "legal" terms and
163 conditions of a *Business* agreement are outside the scope of this specification and therefore are
164 not included in the *CPP* and *CPA*.

165
166 An enterprise MAY choose to represent itself as multiple *Parties*. For example, it might
167 represent a central office supply procurement organization and a manufacturing supplies
168 procurement organization as separate *Parties*. The enterprise MAY then construct a *CPP* that
169 includes all of its units that are represented as separate *Parties*. In the *CPP*, each of those units
170 would be represented by a separate *PartyInfo* element.

171
172 In general, the *Parties* to a *CPA* can have both client and server characteristics. A client requests
173 services and a server provides services to the *Party* requesting services. In some applications,
174 one *Party* only requests services and one *Party* only provides services. These applications have
175 some resemblance to traditional client-server applications. In other applications, each *Party*
176 MAY request services of the other. In that case, the relationship between the two *Parties* can be
177 described as a peer-peer relationship rather than a client-server relationship.

178

179 **6.2 Forming a CPA from Two CPPs**

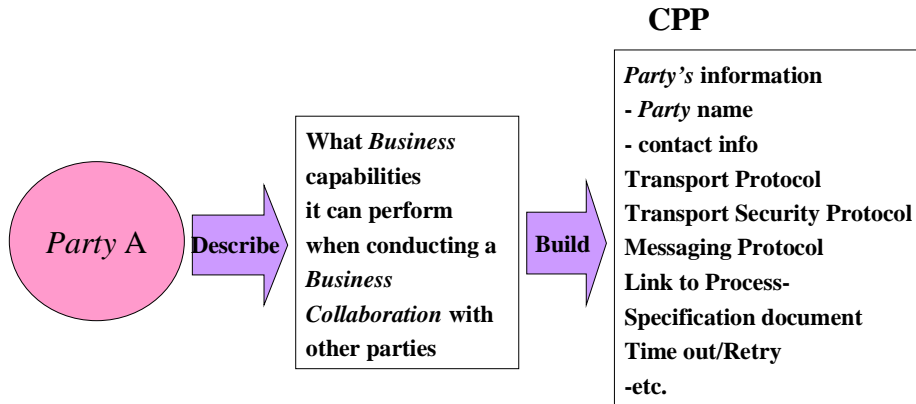
180 This section summarizes the process of discovering a *Party* to do *Business* with and forming a
181 *CPA* from the two *Parties'* *CPPs*. In general, this section is an overview of a possible procedure
182 and is not to be considered a normative specification. See Appendix F "Composing a CPA from
183 Two CPPs (Non-Normative)" for more information.

184

185 Figure 2 illustrates forming a *CPP*. *Party A* tabulates the information to be placed in a repository
186 for the discovery process, constructs a *CPP* that contains this information, and enters it into an
187 ebXML Registry or similar repository along with additional information about the *Party*. The
188 additional information might include a description of the *Businesses* that the *Party* engages in.
189 Once *Party A's* information is in the repository, other *Parties* can discover *Party A* by using the
190 repository's discovery services.

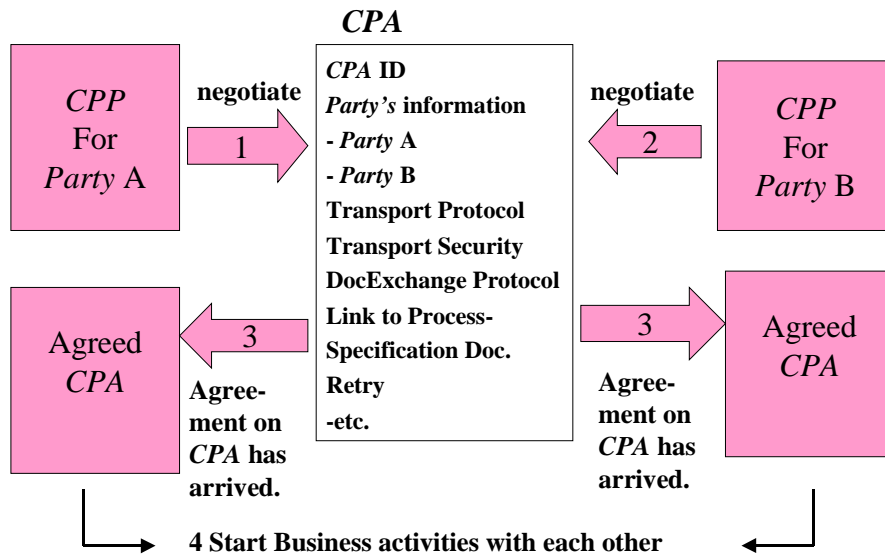
191

Figure 2: Overview of Collaboration-Protocol Profiles (CPP)



192 In figure 3, *Party A* and *Party B* use their *CPPs* to jointly construct a single copy of a *CPA* by
 193 calculating the intersection of the information in their *CPPs*. The resulting *CPA* defines how the
 194 two *Parties* will behave in performing their *Business Collaboration*.

Figure 3: Overview of Collaboration-Protocol Agreements (CPA)



195

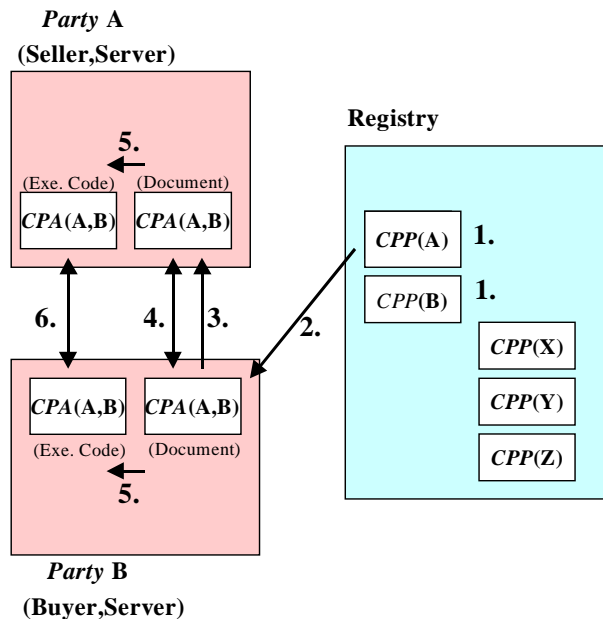
196

197
198
199
200
201

Figure 4 illustrates the entire process. The steps are listed at the left. The end of the process is that the two *Parties* configure their systems from identical copies of the agreed *CPA* and they are then ready to do *Business*.

Figure 4: Overview of Working Architecture of CPP/CPA with ebXML Registry

1. Any *Party* may register its *CPPs* to an ebXML Registry.
2. *Party B* discovers trading partner *A* (Seller) by searching in the Registry and downloads *CPP(A)* to *Party B's* server.
3. *Party B* creates *CPA(A,B)* and sends *CPA(A,B)* to *Party A*.
4. *Parties A* and *B* negotiate and store identical copies of the completed *CPA* as a document in both servers. This process is done manually or automatically.
5. *Parties A* and *B* configure their run-time systems with the information in the *CPA*.
6. *Parties A* and *B* do business under the new *CPA*.



202
203
204
205
206

NOTE: This specification makes the assumption that a *CPP* that has been registered in an ebXML or other Registry will be referenced by some Registry-assigned globally-unique identifier that MAY be used to distinguish among multiple *CPPs* belonging to the same *Party*. See section 7.1 for more information.

6.3 How the CPA Works

A *CPA* describes all the valid visible, and hence enforceable, interactions between the *Parties* and the way these interactions are carried out. It is independent of the internal processes executed at each *Party*. Each *Party* executes its own internal processes and interfaces them with the *Business Collaboration* described by the *CPA* and *Process-Specification* document. The *CPA* does not expose details of a *Party's* internal processes to the other *Party*. The intent of the *CPA* is to provide a high-level specification that can be easily comprehended by humans and yet is precise enough for enforcement by computers.

215
216
217
218
219

The information in the *CPA* is used to configure the *Parties'* systems to enable exchange of *Messages* in the course of performing the selected *Business Collaboration*. Typically, the software that performs the *Messages* exchanges and otherwise supports the interactions between the *Parties* is middleware that can support any selected *Business Collaboration*. One component

220 of this middleware is the ebXML *Message Service Handler*[ebMS]. In this specification, the
221 term "run-time system" or "run-time software" is used to denote such middleware.

222
223 The *CPA* and the *Process-Specification* document that it references define a conversation
224 between the two *Parties*. The conversation represents a single unit of *Business* as defined by the
225 *Binary-Collaboration* component of the *Process-Specification* document. The conversation
226 consists of one or more *Business Transactions*, each of which is a request *Message* from one
227 *Party* and a response *Message* from the other *Party*. The *Process-Specification* document
228 defines, among other things, the request and response *Messages* for each *Business Transaction*
229 and the order in which the *Business Transactions* are REQUIRED to occur. See [ebBPSS] for a
230 detailed explanation.

231
232 The *CPA* MAY actually reference more than one *Process-Specification* document. When a *CPA*
233 references more than one *Process-Specification* document, each *Process-Specification* document
234 defines a distinct type of conversation. Any one conversation involves only a single *Process-*
235 *Specification* document.

236
237 A new conversation is started each time a new unit of *Business* is started. The *Business*
238 *Collaboration* also determines when the conversation ends. From the viewpoint of a *CPA*
239 between *Party A* and *Party B*, the conversation starts at *Party A* when *Party A* sends the first
240 request *Message* to *Party B*. At *Party B*, the conversation starts when it receives the first request
241 of the unit of *Business* from *Party A*. A conversation ends when the *Parties* have completed the
242 unit of *Business*.

243
244 NOTE: The run-time system SHOULD provide an interface by which the *Business*
245 application can request initiation and ending of conversations.

246

247 **6.4 Where the CPA May Be Implemented**

248 Conceptually, a *Business-to-Business* (B2B) server at each *Party's* site implements the *CPA* and
249 *Process-Specification* document. The B2B server includes the run-time software, i.e. the
250 middleware that supports communication with the other *Party*, execution of the functions
251 specified in the *CPA*, interfacing to each *Party's* back-end processes, and logging the interactions
252 between the *Parties* for purposes such as audit and recovery. The middleware might support the
253 concept of a long-running conversation as the embodiment of a single unit of *Business* between
254 the *Parties*. To configure the two *Parties' systems* for *Business to Business* operations, the
255 information in the copy of the *CPA* and *Process-Specification* documents at each *Party's* site is
256 installed in the run-time system. The static information MAY be recorded in a local database and
257 other information in the *CPA* and *Process-Specification* document MAY be used in generating or
258 customizing the necessary code to support the *CPA*.

259
260 NOTE: It is possible to provide a graphic *CPP/CPA*-authoring tool that understands both
261 the semantics of the *CPP/CPA* and the XML syntax. Equally important, the definitions in
262 this specification make it feasible to automatically generate, at each *Party's* site, the code
263 needed to execute the *CPA*, enforce its rules, and interface with the *Party's* back-end
264 processes.

265

266 **6.5 Definition and Scope**

267
268 This specification defines and explains the contents of the *CPP* and *CPA XML* documents. Its
269 scope is limited to these definitions. It does not define how to compose a *CPA* from two *CPPs*
270 nor does it define anything related to run-time support for the *CPP* and *CPA*. It does include
271 some non-normative suggestions and recommendations regarding run-time support where these
272 notes serve to clarify the *CPP* and *CPA* definitions. See section 10 for a discussion of
273 conformance to this specification.

274
275 **NOTE:** This specification is limited to defining the contents of the *CPP* and *CPA*, and it is
276 possible to be conformant with it merely by producing a *CPP* or *CPA* document that
277 conforms to the DTD and XML Schema documents defined herein. It is, however, important
278 to understand that the value of this specification lies in its enabling a run-time system that
279 supports electronic commerce between two *Parties* under the guidance of the information in
280 the *CPA*.

281 7 CPP Definition

282 A *CPP* defines the capabilities of a *Party* to engage in electronic *Business* with other *Parties*.
283 These capabilities include both technology capabilities, such as supported communication and
284 messaging protocols, and *Business* capabilities in terms of what *Business Collaborations* it
285 supports.

286
287 This section defines and discusses the details in the *CPP* in terms of the individual XML
288 elements. The discussion is illustrated with some XML fragments. See Appendix C and
289 Appendix D for the DTD and XML Schema, respectively, and Appendix A for a sample *CPP*
290 document.

291
292 The ***ProcessSpecification***, ***DeliveryChannel***, ***DocExchange***, and ***Transport*** elements of the
293 *CPP* describe the processing of a unit of *Business* (conversation). These elements form a layered
294 structure somewhat analogous to a layered communication model. The remainder of this section
295 describes both the above-mentioned elements and the corresponding run-time processing.

296
297 ***Process-Specification layer*** - The *Process-Specification* layer defines the heart of the *Business*
298 agreement between the *Parties*: the services (*Business Transactions*) which *Parties* to the *CPA*
299 can request of each other and transition rules that determine the order of requests. This layer is
300 defined by the separate *Process-Specification* document that is referenced by the *CPP* and *CPA*.

301
302 ***Delivery Channels*** - A delivery channel describes a *Party's Message*-receiving characteristics. It
303 consists of one document-exchange definition and one transport definition. Several delivery
304 channels MAY be defined in one *CPP*.

305
306 ***Document-Exchange layer*** - The document-exchange layer accepts a *Business* document from
307 the *Process-Specification* layer at one *Party*, encrypts it if specified, adds a digital signature for
308 nonrepudiation if specified, and passes it to the transport layer for transmission to the other
309 *Party*. It performs the inverse steps for received *Messages*. The options selected for the
310 document-exchange layer are complementary to those selected for the transport layer. For
311 example, if *Message* security is desired and the selected transport protocol does not provide
312 *Message* encryption, then it must be specified at the document-exchange layer. The protocol for
313 exchanging *Messages* between two *Parties* is defined by the ebXML *Message Service*
314 Specification[ebMS] or other similar messaging service.

315
316 ***Transport layer*** - The transport layer is responsible for *Message* delivery using the selected
317 transport protocol. The selected protocol affects the choices selected for the document-exchange
318 layer. For example, some transport-layer protocols might provide encryption and authentication
319 while others have no such facility.

320
321 It should be understood that the functional layers encompassed by the *CPP* have no
322 understanding of the contents of the payload of the *Business* documents.

323

324 **7.1 Globally-Unique Identifier of CPP Instance Document**

325 When a *CPP* is placed in an ebXML or other Registry, the Registry assigns it a globally-unique
 326 identifier (GUID) that is part of its metadata. That GUID MAY be used to distinguish among
 327 *CPPs* belonging to the same *Party*.

328
 329 NOTE: A Registry cannot insert the GUID into the *CPP*. In general, a Registry does not
 330 alter the content of documents submitted to it. Furthermore, a *CPP* MAY be signed and
 331 alteration of a signed *CPP* would invalidate the signature.

333 **7.2 SchemaLocation Attribute**

334 The W3C XML Schema specification[XMLSCHEMA-1,XMLSCHEMA-2] that went to
 335 Candidate Recommendation status, effective October 24, 2000, has recently gone to Proposed
 336 Recommendation effective March 30, 2001. Many, if not most, tools providing support for
 337 schema validation and validating XML parsers available at the time that this specification was
 338 written have been designed to support the Candidate Recommendation draft of the XML Schema
 339 specification.

340
 341 In order to enable validating parsers and various schema-validating tools to correctly process and
 342 parse ebXML CPP and CPA documents, it has been necessary that the ebXML TP team produce
 343 a schema that conforms to the W3C Candidate Recommendation draft of the XML Schema
 344 specification. Implementations of CPP and CPA authoring tools are STRONGLY
 345 RECOMMENDED to include the XMLSchema-instance namespace-qualified schemaLocation
 346 attribute in the document's root element to indicate to validating parsers the location URI of the
 347 schema document that should be used to validate the document. Failure to include the
 348 schemaLocation attribute MAY result in interoperability issues with other tools that need to be
 349 able to validate these documents.

350
 351 At such time as the XML Schema specification is adopted as a W3C Recommendation, a revised
 352 CPP/CPA schema SHALL be produced that SHALL contain any updates as necessary to
 353 conform to that Recommendation.

354
 355 An example of the use of the schemaLocation attribute follows:

```

356
357     <CollaborationProtocolAgreement
358         xmlns="http://www.ebxml.org/namespaces/tradePartner"
359         xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
360         xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
361             http://ebxml.org/project_teams/trade_partner/cpp-cpa-10.xsd"
362         ...
363     >
364     ...
365 </CollaborationProtocolAgreement>
    
```

367

368 7.3 CPP Structure

369 Following is the overall structure of the *CPP*. Unless otherwise noted, *CPP* elements **MUST** be
370 in the order shown here. Subsequent sections describe each of the elements in greater detail.

```
371
372 <CollaborationProtocolProfile
373     xmlns="http://www.ebxml.org/namespaces/tradePartner"
374     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
375     xmlns:xlink="http://www.w3.org/1999/xlink"
376     version="1.1">
377     <PartyInfo> <!--one or more-->
378         ...
379     </PartyInfo>
380     <Packaging id="ID"> <!--one or more-->
381         ...
382     <Packaging>
383     <ds:Signature> <!--zero or one-->
384         ...
385     </ds:Signature>
386     <Comment>text</Comment> <!--zero or more-->
387 </CollaborationProtocolProfile>
388
```

389 7.4 CollaborationProtocolProfile element

390 The *CollaborationProtocolProfile* element is the root element of the *CPP* XML document.

391 The REQUIRED [XML] Namespace[XMLNS] declarations for the basic document are as
392 follows:

- 393 • The default namespace: xmlns="http://www.ebxml.org/namespaces/tradePartner",
- 394 • XML Digital Signature namespace:
395 xmlns:ds="http://www.w3.org/2000/09/xmldsig#",
- 396 • and the XLINK namespace: xmlns:xlink="http://www.w3.org/1999/xlink".

397
398 In addition, the *CollaborationProtocolProfile* element contains an IMPLIED *version* attribute
399 that indicates the version of the *CPP*. Its purpose is to provide versioning capabilities for
400 instances of an enterprise's *CPP*. The value of the version attribute SHOULD be a string
401 representation of a numeric value such as "1.0" or "2.3". The value of the version string
402 SHOULD be changed with each change made to the *CPP* document after it has been published.

403
404 NOTE: The method of assigning the version-identifier value is left to the implementation.
405

406 The *CollaborationProtocolProfile* element SHALL consist of the following child elements:

- 407 • One or more REQUIRED *PartyInfo* elements that identify the organization (or parts
408 of the organization) whose capabilities are described by the *CPP*.
- 409 • Zero or one *ds:Signature* elements that contain the digital signature that signs the
410 *CPP* document.
- 411 • Zero or more *Comment* elements.

412
413 A *CPP* document MAY be digitally signed so as to provide for a means of ensuring that the
414 document has not been altered (integrity) and to provide for a means of authenticating the author
415 of the document. A digitally signed *CPP* SHALL be signed using technology that conforms to
416 the joint W3C/IETF XML Digital Signature specification[XMLDSIG].

417

418 **7.5 PartyInfo Element**

419 The *PartyInfo* element identifies the organization whose capabilities are described in this *CPP*
 420 and includes all the details about this *Party*. More than one *PartyInfo* element MAY be
 421 provided in a *CPP* if the organization chooses to represent itself as subdivisions with different
 422 characteristics. Each of the subelements of *PartyInfo* is discussed later. The overall structure of
 423 the *PartyInfo* element is as follows:

424

```

425 <PartyInfo>
426     <PartyId type="..."> <!--one or more-->
427         ...
428     </PartyId>
429     <PartyRef xlink:type="...", xlink:href="..." />
430     <CollaborationRole> <!--one or more-->
431         ...
432     </CollaborationRole>
433     <Certificate> <!--one or more-->
434         ...
435     </Certificate>
436     <DeliveryChannel> <!--one or more-->
437         ...
438     </DeliveryChannel>
439     <Transport> <!--one or more-->
440         ...
441     </Transport>
442     <DocExchange> <!--one or more-->
443         ...
444     </DocExchange>
445 </PartyInfo>
    
```

446

447 The *PartyInfo* element consists of the following child elements:

- 448 • One or more REQUIRED *PartyId* elements that provide a logical identifier for the
 449 organization.
- 450 • A REQUIRED *PartyRef* element that provides a pointer to more information about
 451 the *Party*.
- 452 • One or more REQUIRED *CollaborationRole* elements that identify the roles that this
 453 *Party* can play in the context of a *Process Specification*.
- 454 • One or more REQUIRED *Certificate* elements that identify the certificates used by
 455 this *Party* in security functions.
- 456 • One or more REQUIRED *DeliveryChannel* elements that define the characteristics of
 457 each delivery channel that the *Party* can use to receive *Messages*. It includes both the
 458 transport level (e.g. HTTP) and the messaging protocol (e.g. ebXML *Message*
 459 *Service*).
- 460 • One or more REQUIRED *Transport* elements that define the characteristics of the
 461 transport protocol(s) that the *Party* can support to receive *Messages*.
- 462 • One or more REQUIRED *DocExchange* elements that define the *Message-exchange*
 463 characteristics, such as the *Message-exchange* protocol, that the *Party* can support.

464

465 **7.5.1 PartyId element**

466 The REQUIRED *PartyId* element provides a logical identifier that MAY be used to logically

467 identify the *Party*. Additional ***PartyId*** elements MAY be present under the same ***PartyInfo***
468 element so as to provide for alternative logical identifiers for the *Party*. If the *Party* has
469 preferences as to which logical identifier is used, the ***PartyId*** elements SHOULD be listed in
470 order of preference starting with the most-preferred identifier.

471
472 In a *CPP* that contains multiple ***PartyInfo*** elements, different ***PartyInfo*** elements MAY contain
473 ***PartyId*** elements that define different logical identifiers. This permits a large organization, for
474 example, to have different identifiers for different purposes.

475
476 The value of the ***PartyId*** element is any string that provides a unique identifier. The identifier
477 MAY be any identifier that is understood by both *Parties* to a *CPA*. Typically, the identifier
478 would be listed in a well-known directory such as DUNS or in any naming system specified by
479 [ISO6523].

480
481 The ***PartyId*** element has a single IMPLIED attribute: ***type*** that has a string value.

482
483 If the ***type*** attribute is present, then it provides a scope or namespace for the content of the
484 ***PartyId*** element.

485
486 If the ***type*** attribute is not present, the content of the ***PartyId*** element MUST be a URI that
487 conforms to [RFC2396]. It is RECOMMENDED that the value of the ***type*** attribute be a URN
488 that defines a namespace for the value of the ***PartyId*** element. Typically, the URN would be
489 registered as a well-known directory of organization identifiers.

490
491 The following example illustrates two URI references.

```
492     <PartyId type = "uriReference">urn:duns:123456789</PartyId>  
493     <PartyId type = "uriReference">urn:www.example.com</PartyId>
```

494
495
496 The first example is the URN for the *Party's* DUNS number, assuming that Dun and Bradstreet
497 has registered a URN for DUNS numbers with the Internet Assigned Numbers Authority
498 (IANA). The last field is the DUNS number of the organization.

499
500 The second example shows an arbitrary URN. This might be a URN that the *Party* has
501 registered with IANA to identify itself directly.

502

503 **7.5.2 PartyRef element**

504
505 The ***PartyRef*** element provides a link, in the form of a URI, to additional information about the
506 *Party*. Typically, this would be the URL from which the information can be obtained. The
507 information might be at the *Party's* web site or in a publicly accessible repository such as an
508 ebXML Registry, a UDDI repository, or an LDAP directory. Information available at that URI
509 MAY include contact names, addresses, and phone numbers, and perhaps more information
510 about the *Business Collaborations* that the *Party* supports. This information MAY be in the form
511 of an ebXML Core Component[ccOVER]. It is not within the scope of this specification to
512 define the content or format of the information at that URI.

513

514 The **PartyRef** element is an [XLINK] simple link. It has the following attributes:

- 515 • a REQUIRED *xlink:type* attribute,
- 516 • a REQUIRED *xlink:href* attribute.

517

518 7.5.2.1 *xlink:type* attribute

519 The REQUIRED *xlink:type* attribute SHALL have a FIXED value of "simple". This identifies
520 the element as being an [XLINK] simple link.

521

522 7.5.2.2 *xlink:href* attribute

523 The REQUIRED *xlink:href* attribute SHALL have a value that is a URI that conforms to
524 [RFC2396] and identifies the location of the external information about the *Party*.

525

526 An example of the **PartyRef** element is:

527

```
528     <PartyRef xlink:type="simple"
529             xlink:href="http://example2.com/ourInfo.html" />
```

530 7.5.3 CollaborationRole element

531 The **CollaborationRole** element associates a *Party* with a specific role in the *Business*
532 *Collaboration* that is defined in the *Process-Specification* document[ebBPSS]. Generally, the
533 *Process Specification* is defined in terms of roles such as "buyer" and "seller". The association
534 between a specific *Party* and the role(s) it is capable of fulfilling within the context of a *Process*
535 *Specification* is defined in both the *CPP* and *CPA* documents. In a *CPP*, the **CollaborationRole**
536 element identifies which role the *Party* is capable of playing in each *Process Specification*
537 documents referenced by the *CPP*. An example of the **CollaborationRole** element is:

538

```
539 <CollaborationRole id="N11" >
540   <ProcessSpecification name="BuySell" version="1.0">
541     ...
542   </ProcessSpecification>
543   <Role name="buyer" xlink:href="..." />
544   <CertificateRef certId = "N03"/>
545   <!-- primary binding with "preferred" DeliveryChannel -->
546   <ServiceBinding name="some process" channelId="N02" packageId="N06">
547     <!-- override "default" deliveryChannel for selected message(s)-->
548     <Override action="OrderAck" channelId="N05" packageId="N09"
549             xlink:type="simple"
550             xlink:href="..." />
551   </ServiceBinding>
552   <!-- the first alternate binding -->
553   <ServiceBinding channelId="N04" packageId="N06">
554     <Override action="OrderAck" channelId="N05" packageId="N09"
555             xlink:type="locator"
556             xlink:href="..." />
557   </ServiceBinding>
558 </CollaborationRole>
```

560

561 To indicate that the *Party* can play roles in more than one *Business Collaboration* or more than
562 one role in a given *Business Collaboration*, the **PartyInfo** element SHALL contain more than
563 one **CollaborationRole** element. Each **CollaborationRole** element SHALL contain the
564 appropriate combination of **ProcessSpecification** element and **Role** element.

565
566 The **CollaborationRole** element SHALL consist of the following child elements: a REQUIRED
567 **ProcessSpecification** element, a REQUIRED **Role** element, zero or one **CertificateRef** element,
568 and one or more **ServiceBinding** elements. The **ProcessSpecification** element identifies the
569 **Process-Specification** document that defines such role. The **Role** element identifies which role
570 the **Party** is capable of supporting. The **CertificateRef** element identifies the certificate to be
571 used. Each **ServiceBinding** element provides a binding of the role to a default **DeliveryChannel**.
572 The default **DeliveryChannel** describes the receive properties of all **Message** traffic that is to be
573 received by the **Party** within the context of the role in the identified **Process-Specification**
574 document. Alternative **DeliveryChannels** MAY be specified for specific purposes, using
575 **Override** elements as described below.

576
577 When there are more than one **ServiceBinding** child elements of a **CollaborationRole**, then the
578 order of the **ServiceBinding** elements SHALL be treated as signifying the **Party's** preference
579 starting with highest and working towards lowest. The default delivery channel for a given
580 **Process-Specification** document is the delivery channel identified by the highest-preference
581 **ServiceBinding** element that references the particular **Process-Specification** document.

582
583 NOTE: When a **CPA** is composed, the **ServiceBinding** preferences are applied in
584 choosing the highest-preference delivery channels that are compatible between the two
585 **Parties**.

586
587 When a **CPA** is composed, only **ServiceBinding** elements that are compatible between the two
588 **Parties** SHALL be retained. Each **Party** SHALL have a default delivery channel for each
589 **Process-Specification** document referenced in the **CPA**. For each **Process-Specification**
590 document, the default delivery channel for each **Party** is the delivery channel that is indicated by
591 the **channelId** attribute in the highest-preference **ServiceBinding** element that references that
592 **Process-Specification** document.

593
594 NOTE: An implementation MAY provide the capability of dynamically assigning
595 delivery channels on a per **Message** basis during performance of the **Business**
596 **Collaboration**. The delivery channel selected would be chosen, based on present
597 conditions, from those identified by **ServiceBinding** elements that refer to the **Business**
598 **Collaboration** that is sending the **Message**. If more than one delivery channel is
599 applicable, the one referred to by the highest-preference **ServiceBinding** element is used.

600
601 The **CollaborationRole** element has the following attribute:

- 602 • a REQUIRED **id** attribute.

603 604 7.5.3.1 id attribute

605 The REQUIRED **id** attribute is an [XML] ID attribute by which this **CollaborationRole** element
606 can be referenced from elsewhere in the **CPP** document.

607 608 7.5.3.2 CertificateRef element

609 The EMPTY **CertificateRef** element contains an IMPLIED IDREF attribute, **certId**, which
610 identifies the certificate to be used by referring to the **Certificate** element (under **PartyInfo**) that

611 has the matching ID attribute value.

612

613 7.5.3.3 certId attribute

614 The IMPLIED *certId* attribute is an [XML] IDREF that associates the *CollaborationRole* with a
615 *Certificate* with a matching ID attribute.

616

617 NOTE: This *certId* attribute relates to the authorizing role in the *Process Specification*
618 while the certificates identified in the delivery-channel description relate to *Message*
619 exchanges.

620

621 7.5.4 ProcessSpecification element

622 The *ProcessSpecification* element provides the link to the *Process-Specification* document that
623 defines the interactions between the two *Parties*. This document is prepared in accord with the
624 ebXML Business Process Specification Schema[ebBPSS]. The *Process-Specification* document
625 MAY be kept in an ebXML Registry.

626

627 The syntax of the *ProcessSpecification* element is:

628

```
629 <ProcessSpecification
630     name="BuySell"
631     version="1.0"
632     xlink:type="simple"
633     xlink:href="http://www.ebxml.org/services/purchasing.xml"
634     <ds:Reference ds:URI="http://www.ebxml.org/services/purchasing.xml">
635         <ds:Transforms>
636             <ds:Transform
637                 ds:Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
638             </ds:Transforms>
639             <ds:DigestMethod
640                 ds:Algorithm="http://www.w3.org/2000/09/xmlsig#dsa-sha1">
641                 String
642             </ds:DigestMethod>
643             <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
644         </ds:Reference>
645 </ProcessSpecification>
```

646

647

648 The *ProcessSpecification* element has a single REQUIRED child element, *ds:Reference*, and the
649 following attributes:

- 650 • a REQUIRED *name* attribute, with type ID,
- 651 • a REQUIRED *version* attribute,
- 652 • a FIXED *xlink:type* attribute,
- 653 • a REQUIRED *xlink:href* attribute.

654

655 The *ds:Reference* element relates to the *xlink:type* and *xlink:href* attributes as follows. Each
656 *ProcessSpecification* element SHALL contain one *xlink:href* attribute and one *xlink:type*
657 attribute with a value of "simple", and MAY contain one *ds:Reference* element formulated
658 according to the XML Digital Signature specification[XMLDSIG]. In case the document is
659 signed, it MUST use the *ds:Reference* element. When the *ds:Reference* element is present, it
660 MUST include a *ds:URI* attribute whose value is identical to that of the *xlink:href* attribute in

661 the enclosing *ProcessSpecification* element.

662

663 **7.5.4.1 name attribute**

664 The *ProcessSpecification* element MUST include a REQUIRED *name* attribute: an [XML] ID
665 that MAY be used to refer to this element from elsewhere within the *CPP* document.

666

667 **7.5.4.2 version attribute**

668 The *ProcessSpecification* element includes a REQUIRED *version* attribute to identify the
669 version of the *Process-Specification* document identified by the *xlink:href* attribute (and also
670 identified by the *ds:Reference* element, if any).

671

672 **7.5.4.3 xlink:type attribute**

673 The *xlink:type* attribute has a FIXED value of "simple". This identifies the element as being an
674 [XLINK] simple link.

675

676 **7.5.4.4 xlink:href attribute**

677 The REQUIRED *xlink:href* attribute SHALL have a value that identifies the *Process-*
678 *Specification* document and is a URI that conforms to [RFC2396].

679

680 **7.5.4.5 ds:Reference element**

681 The *ds:Reference* element identifies the same *Process-Specification* document as the enclosing
682 *ProcessSpecification* element's *xlink:href* attribute and additionally provides for verification that
683 the *Process-Specification* document has not changed since the *CPP* was created.

684

685 NOTE: *Parties* MAY test the validity of the *CPP* or *CPA* at any time. The following
686 validity tests MAY be of particular interest:

687

- 688 • test of the validity of a *CPP* and the referenced *Process-Specification* documents at
689 the time composition of a *CPA* begins in case they have changed since they were
690 created,
- 691 • test of the validity of a *CPA* and the referenced *Process-Specification* documents at
692 the time a *CPA* is installed into a *Party's* system,
- 693 • test of the validity of a *CPA* at intervals after the *CPA* has been installed into a *Party's*
694 system. The *CPA* and the referenced *Process-Specification* documents MAY be
695 processed by an installation tool into a form suited to the particular middleware.
696 Therefore, alterations to the *CPA* and the referenced *Process-Specification* documents
697 do not necessarily affect ongoing run-time operations. Such alterations might not be
698 detected until it becomes necessary to reinstall the *CPA* and the referenced *Process-*
699 *Specification* documents.

700

701 The syntax and semantics of the *ds:Reference* element and its child elements are defined in the
702 XML Digital Signature specification[XMLDSIG]. As an alternative to the string value of the
703 *ds:DigestMethod*, shown in the above example, the child element, *ds:HMACOutputLength*,
704 with a string value, MAY be used.

705

706 According to [XMLDSIG], a *ds:Reference* element can have a *ds:Transforms* child element,

707 which in turn has an ordered list of one or more *ds:Transform* child elements to specify a
708 sequence of transforms. However, this specification currently **REQUIRES** the Canonical
709 XML[XMLC14N] transform and forbids other transforms. Therefore, the following additional
710 requirements apply to a *ds:Reference* element within a *ProcessSpecification* element:

- 711
- 712 • The *ds:Reference* element **MUST** have a *ds:Transforms* child element.
- 713 • That *ds:Transforms* element **MUST** have exactly one *ds:Transform* child element.
- 714 • That *ds:Transform* element **MUST** specify the Canonical XML[XMLC14N]
715 transform via the following **REQUIRED** value for its **REQUIRED** *ds:Algorithm*
716 attribute: <http://www.w3.org/TR/2000/CR-xml-c14n-20001026>

717

718 Note that implementation of Canonical XML is **REQUIRED** by the XML Digital Signature
719 specification[XMLDSIG].

720

721 A *ds:Reference* element in a *ProcessSpecification* element has implications for *CPP* validity:

- 722
- 723 • A *CPP* **MUST** be considered invalid if any *ds:Reference* element within a
724 *ProcessSpecification* element fails reference validation as defined by the XML Digital
725 Signature specification[XMLDSIG].
- 726
- 727 • A *CPP* **MUST** be considered invalid if any *ds:Reference* within it cannot be
728 dereferenced.

729

730 Other validity implications of such *ds:Reference* elements are specified in the description of the
731 *ds:Signature* element.

732

733 **NOTE:** The XML Digital Signature specification[XMLDSIG] states "The signature
734 application **MAY** rely upon the identification (URI) and Transforms provided by the
735 signer in the Reference element, or it **MAY** obtain the content through other means such
736 as a local cache" (emphases on **MAY** added). However, it is **RECOMMENDED** that
737 ebXML *CPP/CPA* implementations not make use such cached results when signing or
738 validating.

739

740 **NOTE:** It is recognized that the XML Digital Signature specification[XMLDSIG]
741 provides for signing an XML document together with externally referenced documents.
742 In cases where a *CPP* or *CPA* document is in fact suitably signed, that facility could also
743 be used to ensure that the referenced *Process-Specification* documents are unchanged.
744 However, this specification does not currently mandate that a *CPP* or *CPA* be signed.

745

746 **NOTE:** If the *Parties* to a *CPA* wish to customize a previously existing *Process-*
747 *Specification* document, they **MAY** copy the existing document, modify it, and cause
748 their *CPA* to reference the modified copy. It is recognized that for reasons of clarity,
749 brevity, or historical record, the parties might prefer to reference a previously existing
750 *Process-Specification* document in its original form and accompany that reference with a
751 specification of the agreed modifications. Therefore, *CPP* usage of the *ds:Reference*
752 element's *ds:Transforms* subelement within a *ProcessSpecification* element might be
753 expanded in the future to allow other transforms as specified in the XML Digital

754 Signature specification[XMLDSIG]. For example, modifications to the original
755 document could then be expressed as XSLT transforms. After applying any transforms,
756 it would be necessary to validate the transformed document against the ebXML Business
757 Process Specification Schema[ebBPSS].
758

759 **7.5.5 Role element**

760 The REQUIRED *Role* element identifies which role in the *Process Specification* the *Party* is
761 capable of supporting via the *ServiceBinding* element(s) siblings within this *CollaborationRole*
762 element.
763

764 The *Role* element has the following attributes:

- 765 • a REQUIRED *name* attribute,
 - 766 • a FIXED *xlink:type* attribute,
 - 767 • a REQUIRED *xlink:href* attribute.
- 768

769 **7.5.5.1 name attribute**

770 The REQUIRED *name* attribute is a string that gives a name to the *Role*. Its value is taken from
771 one of the following sources in the *Process Specification*[ebBPSS] that is referenced by the
772 *ProcessSpecification* element depending upon which element is the "root" (highest order) of the
773 process referenced:

- 774 • *name* attribute of a *BinaryCollaboration/AuthorizedRole* element,
 - 775 • *fromAuthorizedRole* attribute of a *BusinessTransactionActivity* element,
 - 776 • *toAuthorizedRole* attribute of a *BusinessTransactionActivity* element,
 - 777 • *fromAuthorizedRole* attribute of a *CollaborationActivity* element,
 - 778 • *toAuthorizedRole* attribute of a *CollaborationActivity* element,
 - 779 • *name* attribute of the *business-partner-role* element.
- 780

781 **7.5.5.2 xlink:type attribute**

782 The *xlink:type* attribute has a FIXED value of "simple". This identifies the element as being an
783 [XLINK] simple link.
784

785 **7.5.5.3 xlink:href attribute**

786 The REQUIRED *xlink:href* attribute SHALL have a value that is a URI that conforms to
787 [RFC2396]. It identifies the location of the element or attribute within the *Process-Specification*
788 document that defines the role in the context of the *Business Collaboration*.
789

790 **7.5.6 ServiceBinding element**

791 The *ServiceBinding* element identifies a *DeliveryChannel* element for all of the *Message* traffic
792 that is to be sent to the *Party* within the context of the identified *Process-Specification* document.
793 An example of the *ServiceBinding* element is:

```
794  
795 <ServiceBinding name="SomeProcess" channelId="X03" packageId="N06">  
796     <Override action="OrderAck"  
797         channelId="X04"  
798         packageId="N09"  
799         xlink:type="simple"
```

```
800         xlink:href="..." /> <!--zero or more-->
801     </ServiceBinding>
```

802

803 The *ServiceBinding* element SHALL have zero or more *Override* child elements.

804

805 The *ServiceBinding* element has the following attributes:

806

- a REQUIRED *name* attribute,
- a REQUIRED *channelId* attribute,
- a REQUIRED *packageId* attribute.

807

808

809

810 7.5.6.1 name attribute

811 The value of the REQUIRED *name* attribute is a string value that labels the *ServiceBinding*
812 element. The value of the *name* attribute SHALL be used as the value of the *Service* element in
813 the ebXML *Message Header*[MSSSPEC].

814

815 7.5.6.2 channelId attribute

816 The REQUIRED *channelId* attribute is an [XML] IDREF that identifies the *DeliveryChannel*
817 that SHALL provide a default technical binding for all of the *Message* traffic that is received for
818 the *Process Specification* that is referenced by the *ProcessSpecification* element.

819

820 7.5.6.3 packageId attribute

821 The REQUIRED *packageId* attribute is an [XML] IDREF that identifies the *Packaging* element
822 that SHALL be used with the *ServiceBinding* element.

823

824 7.5.7 Override element

825 The *Override* element provides a *Party* with the ability to map, or bind, a different
826 *DeliveryChannel* to selected *Messages* that are to be received by the *Party* within the context of
827 the parent *ServiceBinding* element.

828

829 Each *Override* element SHALL specify a different *DeliveryChannel* for selected *Messages* that
830 are to be received by the *Party* in the context of the *Process Specification* that is associated with
831 the parent *ServiceBinding* element. The *Override* element has the following attributes:

832

- a REQUIRED *action* attribute,
- a REQUIRED *channelId* attribute,
- a REQUIRED *packageId* attribute,
- an IMPLIED *xlink:href* attribute,
- a FIXED *xlink:type* attribute.

833

834

835

836

837

838 Under a given *ServiceBinding* element, there SHALL be only one *Override* element whose
839 *action* attribute has a given value.

840

841 NOTE: It is possible that when a *CPA* is composed from two *CPPs*, a delivery channel in
842 one *CPP* might have an *Override* element that will not be compatible with the other *Party*.

843 This incompatibility MUST be resolved either by negotiation or by reverting to a compatible
844 default delivery channel.

845

846 **7.5.7.1 action attribute**

847 The REQUIRED *action* attribute is a string that identifies the *Message* that is to be associated
 848 with the *DeliveryChannel* that is identified by the *channelId* attribute. The value of the *action*
 849 attribute MUST match the corresponding *request* or *response* element/attribute in the *Process-*
 850 *Specification* document that is referenced by the *ProcessSpecification* element.

851
 852 **7.5.7.2 channelId attribute**

853 The REQUIRED *channelId* attribute is an [XML] IDREF that identifies the *DeliveryChannel*
 854 element that is to be associated with the *Message* that is identified by the *action* attribute.

855
 856 **7.5.7.3 packageId attribute**

857 The REQUIRED *packageId* attribute is an [XML] IDREF that identifies the *Packaging* element
 858 that is to be associated with the *Message* that is identified by the *action* attribute.

859
 860 **7.3.7.4 xlink:href attribute**

861 The IMPLIED *xlink:href* attribute MAY be present. If present, it SHALL provide an absolute
 862 [XPOINTER] URI expression that specifically identifies the *BusinessTransaction* element
 863 within the associated *Process-Specification* document[ebBPSS] that is identified by the
 864 *ProcessSpecification* element.

865
 866 **7.3.7.5 xlink:type attribute**

867 The IMPLIED *xlink:type* attribute has a FIXED value of "simple". This identifies the element as
 868 being an [XLINK] simple link.

869
 870 **7.5.8 Certificate element**

871 The *Certificate* element defines certificate information for use in this *CPP*. One or more
 872 *Certificate* elements MAY be provided for use in the various security functions in the *CPP*. An
 873 example of the *Certificate* element is:

```
874     <Certificate certId = "N03">
875         <ds:KeyInfo>. . .</ds:KeyInfo>
876     </Certificate>
```

877
 878
 879 The *Certificate* element has a single REQUIRED attribute: *certId*. The *Certificate* element has a
 880 single child element: *ds:KeyInfo*.

881
 882 **7.5.8.1 certId attribute**

883 The REQUIRED *certId* attribute is an ID attribute. Its is referred to in a *CertificateRef* element,
 884 using an IDREF attribute, where a certificate is specified elsewhere in the *CPP*. For example:

```
885     <CertificateRef certId = "N03"/>
```

886
 887
 888 **7.5.8.2 ds:KeyInfo element**

889 The *ds:KeyInfo* element defines the certificate information. The content of this element and any
 890 subelements are defined by the XML Digital Signature specification[XMLDSIG].

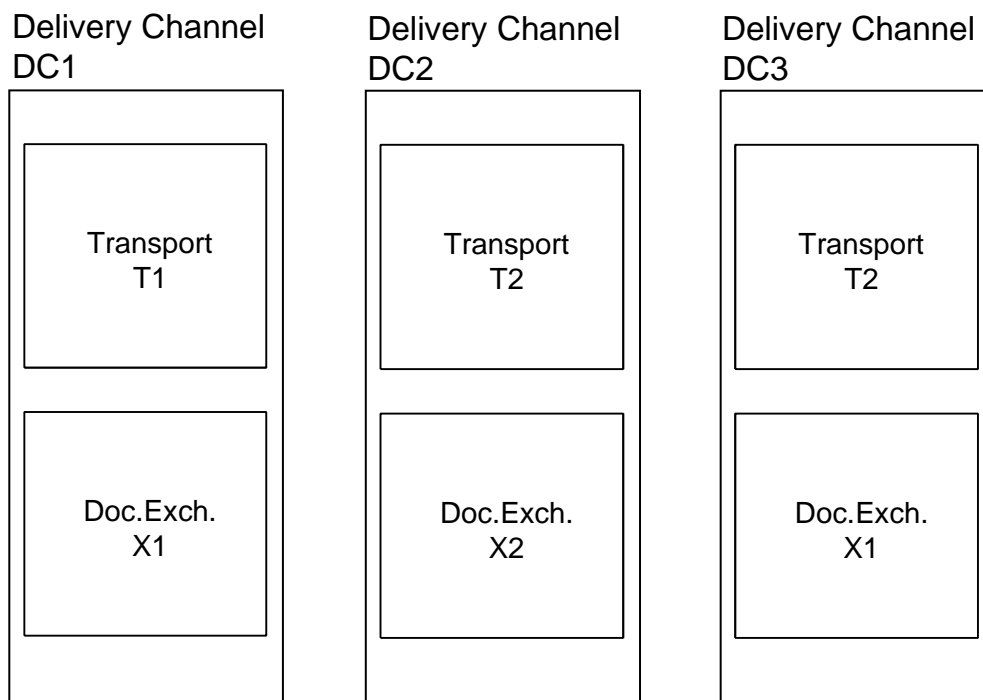
891
 892 NOTE: Software for creation of *CPPs* and *CPAs* MAY recognize the *ds:KeyInfo* element

893 and insert the subelement structure necessary to define the certificate.
 894

895 **7.5.9 DeliveryChannel element**

896 A delivery channel is a combination of a *Transport* element and a *DocExchange* element that
 897 describes the *Party's Message*-receiving characteristics. The *CPP* SHALL contain one or more
 898 *DeliveryChannel* elements, one or more *Transport* elements, and one or more *DocExchange*
 899 elements. Each delivery channel MAY refer to any combination of a *DocExchange* element and
 900 a *Transport* element. The same *DocExchange* element or the same *Transport* element MAY be
 901 referred to by more than one delivery channel. Two delivery channels MAY use the same
 902 transport protocol and the same document-exchange protocol and differ only in details such as
 903 communication addresses or security definitions. Figure 5 illustrates three delivery channels.

Figure 5: Three Delivery Channels



904 The delivery channels have ID attributes with values "DC1", "DC2", and "DC3". Each delivery
 905 channel contains one transport definition and one document-exchange definition. Each transport
 906 definition and each document-exchange definition also has a name as shown in the figure. Note
 907 that delivery-channel DC3 illustrates that a delivery channel MAY refer to the same transport
 908 definition and document-exchange definition used by other delivery channels but a different
 909 combination. In this case delivery-channel DC3 is a combination of transport definition T2 (also
 910 referred to by delivery-channel DC2) and document-exchange definition X1 (also referred to by
 911 delivery-channel DC1).
 912

913 A specific delivery channel SHALL be associated with each *ServiceBinding* element or
 914 *Override* element (*action* attribute). Following is the delivery-channel syntax.
 915

```
916
917     <DeliveryChannel channelId="N04" transportId="N05" docExchangeId="N06">
918         <Characteristics
919             syncReplyMode = "responseOnly"
920             nonrepudiationOfOrigin = "true"
921             nonrepudiationOfReceipt = "true"
922             secureTransport = "true"
923             confidentiality = "true"
924             authenticated = "true"
925             authorized = "true"/>
926     </DeliveryChannel>
```

927
928 Each *DeliveryChannel* element identifies one *Transport* element and one *DocExchange* element
929 that make up a single delivery channel definition.

930

931 The *DeliveryChannel* element has the following attributes:

- 932 • a REQUIRED *channelId* attribute,
- 933 • a REQUIRED *transportId* attribute,
- 934 • a REQUIRED *docExchangeId* attribute.

935

936 The *DeliveryChannel* element has one REQUIRED child element, *Characteristics*.

937

938 **7.5.9.1 channelId attribute**

939 The *channelId* attribute is an [XML] ID attribute that uniquely identifies the *DeliveryChannel*
940 element for reference, using IDREF attributes, from other parts of the *CPP* or *CPA*.

941

942 **7.5.9.2 transportId attribute**

943 The *transportId* attribute is an [XML] IDREF that identifies the *Transport* element that defines
944 the transport characteristics of the delivery channel. It MUST have a value that is equal to the
945 value of a *transportId* attribute of a *Transport* element elsewhere within the *CPP* document.

946

947 **7.5.9.3 docExchangeId attribute**

948 The *docExchangeId* attribute is an [XML] IDREF that identifies the *DocExchange* element that
949 defines the document-exchange characteristics of the delivery channel. It MUST have a value
950 that is equal to the value of a *docExchangeId* attribute of a *DocExchange* element elsewhere
951 within the *CPP* document.

952

953 **7.5.10 Characteristics element**

954 The *Characteristics* element describes the security characteristics and other attributes of the
955 delivery channel. The attributes of the *Characteristics* element, except *syncReplyMode*, MAY be
956 used to override the values of the corresponding attributes in the *Process-Specification*
957 document.

958

959 The *Characteristics* element has the following attributes:

- 960 • An IMPLIED *syncReplyMode* attribute,
- 961 • an IMPLIED *nonrepudiationOfOrigin* attribute,
- 962 • an IMPLIED *nonrepudiationOfReceipt* attribute,
- 963 • an IMPLIED *secureTransport* attribute,

- 964 • an IMPLIED *confidentiality* attribute,
- 965 • an IMPLIED *authenticated* attribute,
- 966 • an IMPLIED *authorized* attribute.

967

968 **7.5.10.1 syncReplyMode attribute**

969 The value of the *syncReplyMode* attribute is an enumeration of the following possible values:

- 970 • "signalsOnly"
- 971 • "responseOnly"
- 972 • "signalsAndResponse"
- 973 • "none"

974

975 This attribute, when present, indicates what the receiving application expects in a response when
976 bound to a synchronous communication protocol such as HTTP. The value of "signalsOnly"
977 indicates that the response returned (on the HTTP 200 response in the case of HTTP) will only
978 include one or more *Business* signals as defined in the *Process Specification* document[ebBPSS],
979 but not a *Business-response Message*. The value of "responseOnly" indicates that only the
980 *Business-response Message* will be returned. The value of "signalsAndResponse" indicates that
981 the application will return the *Business-response Message* in addition to one or more *Business*
982 signals. The value of "none", which is the implied default value in the absence of the
983 *syncReplyMode* attribute, indicates that neither the *Business-response Message* nor any *Business*
984 signals will be returned synchronously. In this case, the *Business-response Message* and any
985 *Business* signals will be returned as separate asynchronous responses.

986

987 The ebXML *Message Service's syncReply* attribute is set to a value of "true" whenever the
988 *syncReplyMode* attribute has a value other than "none".

989

990 If the delivery channel identifies a transport protocol that has no synchronous capabilities (such
991 as SMTP) and the *Characteristics* element has a *syncReplyMode* attribute with a value other
992 than "none", a response will contain the same content as if the transport protocol did support
993 synchronous responses.

994

995 **7.5.10.2 nonrepudiationOfOrigin attribute**

996 The *nonrepudiationOfOrigin* attribute is a Boolean with possible values of "true" and "false".
997 If the value is "true" then the delivery channel REQUIRES the *Message* to be digitally signed by
998 the certificate of the *Party* that sent the *Message*.

999

1000 **7.5.10.3 nonrepudiationOfReceipt attribute**

1001 The *nonrepudiationOfReceipt* attribute is a Boolean with possible values of "true" and "false".
1002 If the value is "true" then the delivery channel REQUIRES that the *Message* be acknowledged by
1003 a digitally signed *Message*, signed by the certificate of the *Party* that received the *Message*, that
1004 includes the digest of the *Message* being acknowledged.

1005

1006 **7.5.10.4 secureTransport attribute**

1007 The *secureTransport* attribute is a Boolean with possible values of "true" and "false". If the
1008 value is "true" then it indicates that the delivery channel uses a secure transport protocol such as
1009 [SSL] or [IPSEC].

1010

7.5.10.5 confidentiality attribute

The *confidentiality* attribute is a Boolean with possible values of "true" and "false". If the value is "true" then it indicates that the delivery channel **REQUIRES** that the *Message* be encrypted in a persistent manner. It **MUST** be encrypted above the level of the transport and delivered, encrypted, to the application.

1016

7.5.10.6 authenticated attribute

The *authenticated* attribute is a Boolean with possible values of "true" and "false". If the value is "true" then it indicates that the delivery channel **REQUIRES** that the sender of the *Message* be authenticated before delivery to the application.

1021

7.5.10.7 authorized attribute

The *authorized* attribute is a Boolean with possible values of "true" and "false". If the value is "true" then it indicates that the delivery channel **REQUIRES** that the sender of the *Message* be authorized before delivery to the application.

1026

7.5.11 Transport element

The *Transport* element of the *CPP* defines the *Party's* capabilities with regard to communication protocol, encoding, and transport security information.

1029

The overall structure of the *Transport* element is as follows:

1032

```

1033     <Transport transportId = "N05">
1034         <!--protocols are HTTP, SMTP, and FTP-->
1035         <SendingProtocol version = "1.1">HTTP</SendingProtocol>
1036         <!--one or more SendingProtocol elements-->
1037         <ReceivingProtocol version = "1.1">HTTP</ReceivingProtocol>
1038         <!--one or more endpoints-->
1039         <Endpoint uri="http://example.com/servlet/ebxmlhandler"
1040             type = "request"/>
1041         <TransportSecurity> <!--0 or 1 times-->
1042             <Protocol version = "3.0">SSL</Protocol>
1043             <CertificateRef certId = "N03"/>
1044         </TransportSecurity>
1045     </Transport>
1046 
```

7.5.11.1 transportId attribute

The *Transport* element has a single **REQUIRED** *transportId* attribute, of type [XML] ID, that provides a unique identifier for each *Transport* element, which **SHALL** be referred to by the *transportId* IDREF attribute in a *DeliveryChannel* element elsewhere within the *CPP* or *CPA* document.

1052

See section 7.5.10.1 for a discussion of synchronous replies.

1054

7.5.12 Transport protocol

Supported communication protocols are HTTP, SMTP, and FTP. The *CPP* **MAY** specify as many protocols as the *Party* is capable of supporting.

1058

1059 NOTE: It is the aim of this specification to enable support for any transport capable of
1060 carrying MIME content using the vocabulary defined herein.
1061

1062 **7.5.12.1 SendingProtocol element**

1063 The *SendingProtocol* element identifies the protocol that a *Party* can, or will, use to send
1064 *Business* data to its intended collaborator. The IMPLIED *version* attribute identifies the specific
1065 version of the protocol. For example, suppose that within a *CPP*, a *Transport* element,
1066 containing *SendingProtocol* elements whose values are SMTP and HTTP, is referenced within a
1067 *DeliveryChannel* element. Suppose, further, that this *DeliveryChannel* element is referenced for
1068 the role of Seller within a purchase-ordering process. Then the party is asserting that it can send
1069 purchase orders by either SMTP or HTTP. In a *CPP*, the *SendingProtocol* element MAY appear
1070 one or more times under each *Transport* element. In a *CPA*, the *SendingProtocol* element shall
1071 appear once.
1072

1073 **7.5.12.2 ReceivingProtocol element**

1074 The *ReceivingProtocol* element identifies the protocol by which a *Party* can receive its *Business*
1075 data from the other *Party*. The IMPLIED *version* attribute identifies the specific version of the
1076 protocol. For example, suppose that within a *CPP*, a *Transport* element is referenced within a
1077 *DeliveryChannel* element containing a *ReceivingProtocol* element whose value is HTTP.
1078 Suppose further that this *DeliveryChannel* element is referenced for the role of seller within a
1079 purchase ordering *Business Collaboration*. Then the party is asserting that it can receive *Business*
1080 responses to purchase orders over HTTP.
1081

1082 Within a *CPA*, the *SendingProtocol* and *ReceivingProtocol* elements serve to indicate the actual
1083 agreement upon what transports will be used for the complementary roles of the collaborators.
1084 For example, continuing the earlier examples, the seller in a purchase-order *Business*
1085 *Collaboration* could specify its receiving protocol to be SMTP and its sending protocol to be
1086 HTTP. These collaborator capabilities would match the buyer capabilities indicated in the *CPP*.
1087 These matches support an interoperable transport agreement where the buyer would send
1088 purchase orders by SMTP and where the responses to purchase orders (acknowledgements,
1089 cancellations, or change requests, for example) would be sent by the seller to the buyer using
1090 HTTP.
1091

1092 To fully describe receiving transport capabilities, the receiving-protocol information needs to be
1093 combined with URLs that provide the endpoints (see below).
1094

1095 NOTE: Though the URL scheme gives information about the protocol used, an explicit
1096 *ReceivingProtocol* element remains useful for future extensibility to protocols all of
1097 whose endpoints are identified by the same URL schemes, such as distinct transport
1098 protocols that all make use of HTTP endpoints. Likewise, both URL schemes of HTTP://
1099 and HTTPS:// can be regarded as the same receiving protocol since HTTPS is HTTP with
1100 [SSL] for the transport-security protocol. Therefore, the *ReceivingProtocol* element is
1101 separated from the endpoints, which are, themselves, needed to provide essential
1102 information needed for connections.
1103

1104 **7.5.13 Endpoint element**

1105 The REQUIRED *uri* attribute of the *Endpoint* element specifies the *Party's* communication
1106 addressing information associated with the *ReceiveProtocol* element. One or more *Endpoint*
1107 elements SHALL be provided for each *Transport* element in order to provide different addresses
1108 for different purposes. The value of the *uri* attribute is a URI that contains the electronic address
1109 of the *Party* in the form REQUIRED for the selected protocol. The value of the *uri* attribute
1110 SHALL conform to the syntax for expressing URIs as defined in [RFC2396].

1111
1112 The *type* attribute identifies the purpose of this endpoint. The value of *type* is an enumeration;
1113 permissible values are "login", "request", "response", "error", and "allPurpose". There can be, at
1114 most, one of each. The *type* attribute MAY be omitted. If it is omitted, its value defaults to
1115 "allPurpose". The "login" endpoint MAY be used for the address for the initial *Message* between
1116 the two *Parties*. The "request" and "response" endpoints are used for request and response
1117 *Messages*, respectively. The "error" endpoint MAY be used as the address for error *Messages*
1118 issued by the messaging service. If no "error" endpoint is defined, these error *Messages* SHALL
1119 be sent to the "response" address, if defined, or to the "allPurpose" endpoint. To enable error
1120 *Messages* to be received, each *Transport* element SHALL contain at least one endpoint of type
1121 "error", "response", or "allPurpose".

1122

1123 **7.5.14 Transport protocols**

1124 In the following sections, we discuss the specific details of each supported transport protocol.

1125

1126 **7.5.14.1 HTTP**

1127 HTTP is Hypertext Transfer Protocol[HTTP]. For HTTP, the address is a URI that SHALL
1128 conform to [RFC2396]. Depending on the application, there MAY be one or more endpoints,
1129 whose use is determined by the application.

1130

1131 Following is an example of an HTTP endpoint:

1132

```
1133     <Endpoint uri="http://example.com/servlet/ebxmlhandler"  
1134         type = "request"/>
```

1135

1136 The "request" and "response" endpoints MAY be dynamically overridden for a particular
1137 request or asynchronous response by application-specified URIs exchanged in *Business*
1138 documents exchanged under the *CPA*.

1139

1140 For a synchronous response, the "response" endpoint is ignored if present. A synchronous
1141 response is always returned on the existing connection, i.e. to the URI that is identified as the
1142 source of the connection.

1143

1144 **7.5.14.2 SMTP**

1145 SMTP is Simple Mail Transfer Protocol[SMTP]. For use with this standard, Multipurpose
1146 Internet Mail Extensions[MIME] MUST be supported. The MIME media type used by the
1147 SMTP transport layer is "Application" with a sub-type of "octet-stream".

1148

1149 For SMTP, the communication address is the fully qualified mail address of the destination *Party*

1150 as defined by [RFC822]. Following is an example of an SMTP endpoint:

```
1151  
1152     <Endpoint uri="mailto:ebxmlhandler@example.com"  
1153         type = "request"/>
```

1154
1155 SMTP with MIME automatically encodes or decodes the document as required, on each link in
1156 the path, and presents the decoded document to the destination document-exchange function.

1157
1158 NOTE: The SMTP mail transfer agent encodes binary data (i.e. data that are not 7-bit
1159 ASCII) unless it is aware that the upper level (mail user agent) has already encoded the
1160 data.

1161
1162 NOTE: SMTP by itself (without any authentication or encryption) is subject to denial of
1163 service and masquerading by unknown *Parties*. It is strongly suggested that those *Parties*
1164 who choose SMTP as their transport layer also choose a suitable means of encryption and
1165 authentication either in the document-exchange layer or in the transport layer such as
1166 [S/MIME].

1167
1168 NOTE: SMTP is an asynchronous protocol that does not guarantee a particular quality of
1169 service. A transport-layer acknowledgment (i.e. an SMTP acknowledgment) to the
1170 receipt of a mail *Message* constitutes an assertion on the part of the SMTP server that it
1171 knows how to deliver the mail *Message* and will attempt to do so at some point in the
1172 future. However, the *Message* is not hardened and might never be delivered to the
1173 recipient. Furthermore, the sender will see a transport-layer acknowledgment only from
1174 the nearest node. If the *Message* passes through intermediate nodes, SMTP does not
1175 provide an end-to-end acknowledgment. Therefore receipt of an SMTP
1176 acknowledgement does not guarantee that the *Message* will be delivered to the
1177 application and failure to receive an SMTP acknowledgment is not evidence that the
1178 *Message* was not delivered. It is recommended that the reliable-messaging protocol in
1179 the ebXML *Message* Service be used with SMTP.

1181 7.5.14.3 FTP

1182 FTP is File Transfer Protocol[RFC959].

1183
1184 Since a delivery channel specifies receive characteristics, each *Party* sends a *Message* using FTP
1185 PUT. The endpoint specifies the user id and input directory path (for PUTs to this *Party*). An
1186 example of an FTP endpoint is:

```
1187  
1188     <Endpoint uri="ftp://userid@server.foo.com"  
1189         type = "request"/>
```

1190
1191 Since FTP must be compatible across all implementations, the FTP for ebXML will use the
1192 minimum sets of commands and parameters available for FTP as specified in [RFC959], section
1193 5.1, and modified in [RFC1123], section 4.1.2.13. The mode SHALL be stream only and the
1194 type MUST be either ASCII Non-print (AN), Image (I) (binary), or Local 8 (L 8) (binary
1195 between 8-bit machines and machines with 36 bit words – for an 8-bit machine Local 8 is the
1196 same as Image).

1197

1198 Stream mode closes the data connection upon end of file. The server side FTP MUST set control
 1199 to "PASV" before each transfer command to obtain a unique port pair if there are multiple third
 1200 party sessions.

1201
 1202 NOTE: [RFC 959] states that User-FTP SHOULD send a PORT command to assign a
 1203 non-default data port before each transfer command is issued to allow multiple transfers
 1204 during a single FTP because of the long delay after a TCP connection is closed until its
 1205 socket pair can be reused.

1206
 1207 NOTE: The format of the 227 reply to a PASV command is not well-standardized and an
 1208 FTP client may assume that the parentheses indicated in [RFC959] will be present when
 1209 in some cases they are not. If the User-FTP program doesn't scan the reply for the first
 1210 digit of host and port numbers, the result will be that the User-FTP might point at the
 1211 wrong host. In the response, the h1, h2, h3, h4 is the IP address of the server host and the
 1212 p1, p2 is a non-default data transfer port that PASV has assigned.

1213
 1214 NOTE: As a recommendation for firewall transparency, [RFC1579] proposes that the
 1215 client sends a PASV command, allowing the server to do a passive TCP open on some
 1216 random port, and inform the client of the port number. The client can then do an active
 1217 open to establish the connection.

1218
 1219 NOTE: Since STREAM mode closes the data connection upon end of file, the receiving
 1220 FTP may assume abnormal disconnect if a 226 or 250 control code hasn't been received
 1221 from the sending machine.

1222
 1223 NOTE: [RFC1579] also makes the observation that it might be worthwhile to enhance the
 1224 FTP protocol to have the client send a new command APSV (all passive) at startup that
 1225 would allow a server that implements this option to always perform a passive open. A
 1226 new reply code 151 would be issued in response to all file transfer requests not preceded
 1227 by a PORT or PASV command; this *Message* would contain the port number to use for
 1228 that transfer. A PORT command could still be sent to a server that had previously
 1229 received APSV; that would override the default behavior for the next transfer operation,
 1230 thus permitting third-party transfers.

1231
 1232 **7.5.15 Transport security**

1233 The *TransportSecurity* element provides the *Party's* security specifications, associated with the
 1234 *ReceivingProtocol* element, for the transport layer of the *CPP*. It MAY be omitted if transport
 1235 security will not be used for any *CPAs* composed from this *CPP*. Unless otherwise specified
 1236 below, transport security applies to *Messages* in both directions.

1237
 1238 Following is the syntax:

```
1239 <TransportSecurity>
1240     <Protocol version = "3.0">SSL</Protocol>
1241     <CertificateRef certId = "N03"/> <!--zero or one-->
1242 </TransportSecurity>
```

1244
 1245 The *TransportSecurity* element contains two REQUIRED child elements, *Protocol* and
 1246 *CertificateRef*.

1247
 1248 **7.5.15.1 Protocol element**

1249 The value of the *Protocol* element can identify any transport security protocol that the *Party* is
 1250 prepared to support. The IMPLIED *version* attribute identifies the version of the specified
 1251 protocol.

1252
 1253 The specific security properties depend on the services provided by the identified protocol. For
 1254 example, SSL performs certificate-based encryption and certificate-based authentication.

1255
 1256 Whether authentication is bidirectional or just from *Message* sender to *Message* recipient
 1257 depends on the selected transport-security protocol.

1258
 1259 **7.5.15.2 CertificateRef element**

1260 The EMPTY *CertificateRef* element contains an IMPLIED IDREF attribute, *certId* that
 1261 identifies the certificate to be used by referring to the *Certificate* element (under *PartyInfo*) that
 1262 has the matching ID attribute value. The *CertificateRef* element MUST be present if the
 1263 transport-security protocol uses certificates. It MAY be omitted otherwise (e.g. if authentication
 1264 is by password).

1265
 1266 **7.5.15.3 Specifics for HTTP**

1267 For encryption with HTTP, the protocol is SSL[SSL] (Secure Socket Layer) Version 3.0, which
 1268 uses public-key encryption.

1269
 1270 **7.6 DocExchange Element**

1271 The *DocExchange* element provides information that the *Parties* must agree on regarding
 1272 exchange of documents between them. This information includes the messaging service
 1273 properties (e.g. ebXML *Message Service*[ebMS]).

1274
 1275 Following is the structure of the *DocExchange* element of the *CPP*. Subsequent sections
 1276 describe each child element in greater detail.

```

1277
1278     <DocExchange docExchangeId = "N06">
1279         <ebXMLBinding version = "0.92">
1280             <ReliableMessaging> <!--cardinality 0 or 1-->
1281                 ...
1282             </ReliableMessaging>
1283             <NonRepudiation> <!--cardinality 0 or 1-->
1284                 ...
1285             </NonRepudiation>
1286             <DigitalEnvelope> <!--cardinality 0 or 1-->
1287                 ...
1288             </DigitalEnvelope>
1289             <NamespaceSupported> <!-- 1 or more -->
1290                 ...
1291             </NamespaceSupported>
1292         </ebXMLBinding>
1293     </DocExchange>
    
```

1294
 1295 The *DocExchange* element of the *CPP* defines the properties of the messaging service to be
 1296 used with *CPAs* composed from the *CPP*.

1297
 1298 The *DocExchange* element is comprised of a single *ebXMLBinding* child element.

1299
 1300 NOTE: The document-exchange section can be extended to other messaging services by
 1301 adding additional *xxxBinding* elements and their child elements that describe the other
 1302 services, where *xxx* is replaced by the name of the additional binding. An example is
 1303 *XPBinding*, which might define support for the future XML Protocol specification.

1304
 1305 **7.6.1 docExchangeId attribute**

1306 The *DocExchange* element has a single IMPLIED *docExchangeId* attribute that is an [XML] ID
 1307 that provides a unique identifier that MAY be referenced from elsewhere within the *CPP*
 1308 document.

1309
 1310 **7.6.2 ebXMLBinding element**

1311 The *ebXMLBinding* element describes properties specific to the ebXML *Message*
 1312 Service[*ebMS*]. The *ebXMLBinding* element is comprised of the following child elements:

- 1313 • zero or one *ReliableMessaging* element which specifies the characteristics of reliable
 1314 messaging,
- 1315 • zero or one *NonRepudiation* element which specifies the requirements for signing the
 1316 *Message*,
- 1317 • zero or one *DigitalEnvelope* element which specifies the requirements for encryption
 1318 by the digital-envelope[*DIGENV*] method,
- 1319 • zero or more *NamespaceSupported* elements that identify any namespace extensions
 1320 supported by the messaging service implementation.

1321
 1322 **7.6.3 version attribute**

1323 The *ebXMLBinding* element has a single REQUIRED *version* attribute that identifies the
 1324 version of the ebXML *Message* Service specification being used.

1325
 1326 **7.6.4 ReliableMessaging element**

1327 The *ReliableMessaging* element specifies the properties of reliable ebXML *Message* exchange.
 1328 The default that applies if the *ReliableMessaging* element is omitted is "BestEffort". See
 1329 Section 7.6.4.1. The following is the element structure:

```

1330
1331 <ReliableMessaging deliverySemantics="OnceAndOnlyOnce"
1332     idempotency="false"
1333     messageOrderSemantics="Guaranteed">
1334     <!--The pair of elements Retries, RetryInterval
1335     has cardinality 0 or 1-->
1336     <Retries>5</Retries>
1337     <RetryInterval>60</RetryInterval> <!--time in seconds-->
1338     <PersistDuration>30S</PersistDuration>
    
```

1339 </ReliableMessaging>

1340

1341 The ***ReliableMessaging*** element is comprised of the following child elements. These elements
1342 have cardinality 0 or 1. They **MUST** all be either present or absent.

- 1343 • a ***Retries*** element,
- 1344 • a ***RetryInterval*** element,
- 1345 • a ***PersistDuration*** element.

1346

1347 The ***ReliableMessaging*** element has attributes as follows:

- 1348 • a REQUIRED ***deliverySemantics*** attribute,
- 1349 • a REQUIRED ***idempotency*** attribute,
- 1350 • an IMPLIED ***messageOrderSemantics*** attribute.

1351

1352 **7.6.4.1 deliverySemantics attribute**

1353 The ***deliverySemantics*** attribute of the ***ReliableMessaging*** element specifies the degree of
1354 reliability of *Message* delivery. This attribute is an enumeration of possible values that consist
1355 of:

- 1356 • "OnceAndOnlyOnce",
- 1357 • "BestEffort".

1358

1359 A value of "OnceAndOnlyOnce" specifies that a *Message* must be delivered exactly once.
1360 "BestEffort" specifies that reliable-messaging semantics are not to be used.

1361

1362 **7.6.4.2 idempotency attribute**

1363 The ***idempotency*** attribute of the ***ReliableMessaging*** element specifies whether the *Party*
1364 requires that all *Messages* exchanged be subject to an idempotency test (detection and discard of
1365 duplicate *Messages*) in the document-exchange layer. The attribute is a Boolean with possible
1366 values of "true" and "false". If the value of the attribute is "true", all *Messages* are subject to the
1367 test. If the value is "false", *Messages* are not subject to an idempotency test in the document-
1368 exchange layer. Testing for duplicates is based on the *Message* identifier; other information that
1369 is carried in the *Message Header* **MAY** also be tested, depending on the context.

1370

1371 NOTE: Additional testing for duplicates **MAY** take place in the *Business* application based
1372 on application information in the *Messages* (e.g. purchase order number).

1373

1374 The idempotency test checks whether a *Message* duplicates a prior *Message* between the same
1375 client and server. If the idempotency test is requested, the receiving messaging service passes a
1376 duplicate *Message* to the recipient *Business Collaboration* with a "duplicate" indication. The
1377 receiving messaging service also returns a "duplicate" indication to the sender of the duplicate.

1378

1379 NOTE: One of the main purposes of this test is to aid in retry following timeouts and in
1380 recovery following node failures. In these cases, the sending *Party* might have sent
1381 request *Messages* and not received responses. The sending *Party* **MAY** re-send such a
1382 *Message*. If the original *Message* had been received, the receiving server discards the
1383 duplicate *Message* and re-sends the original results to the requester.

1384

1385 If a communication protocol always checks for duplicate *Messages*, the check in the
1386 communication protocol overrides any idempotency specifications in the *CPA*.

1387

1388 **7.6.4.3 messageOrderSemantics attribute**

1389 The *messageOrderSemantics* attribute of the *ReliableMessaging* element controls the order in
1390 which *Messages* are received when reliable messaging is in effect (the value of the
1391 *deliverySemantics* attribute is "OnceAndOnlyOnce"). This attribute has possible values of:

- 1392 • "Guaranteed": For each conversation, the *Messages* are passed to the receiving
1393 application in the order that the sending application specified.
- 1394 • "NotGuaranteed": The *Messages* MAY be passed to the receiving application in different
1395 order from the order which sending application specified.

1396

1397 It should be understood that when the value of the *messageOrderSemantics* attribute is
1398 "Guaranteed", ordering of *Messages* applies separately to each conversation; the relative order of
1399 *Messages* in different conversations is not specified.

1400

1401 The default value of the *messageOrderSemantics* attribute is "NotGuaranteed". This attribute
1402 MUST NOT be present when the value of the *deliverySemantics* attribute is anything other than
1403 "OnceAndOnlyOnce".

1404

1405 The sending ebXML *Message Service*[ebMS] sets the value of the *messageOrderSemantics*
1406 attribute of the *QualityOfServiceInfo* element in the *Message* header to the value of the
1407 *messageOrderSemantics* attribute specified by the *To Party* in the *CPA*.

1408

1409 **7.6.4.4 Retries and RetryInterval elements**

1410 The *Retries* and *RetryInterval* elements specify the permitted number of retries and interval
1411 between retries (in seconds) of a request following a timeout. The purpose of the *RetryInterval*
1412 element is to improve the likelihood of success on retry by deferring the retry until any
1413 temporary conditions that caused the error might be corrected.

1414

1415 The *Retries* and *RetryInterval* elements MUST be included together or MAY be omitted
1416 together. If they are omitted, the values of the corresponding quantities (number of retries and
1417 retry interval) are a local matter at each *Party*.

1418

1419 **7.6.4.5 PersistDuration element**

1420 The value of the *PersistDuration* element is the minimum length of time, expressed as an XML
1421 Schema[XMLSCHEMA-2] *timeDuration*, that data from a *Message* that is sent reliably is kept in
1422 *Persistent Storage* by an ebXML *Message-Service* implementation that receives that *Message*.

1423

1424 **7.6.5 NonRepudiation element**

1425 Non-repudiation both proves who sent a *Message* and prevents later repudiation of the contents
1426 of the *Message*. Non-repudiation is based on signing the *Message* using XML Digital
1427 Signature[XMLDSIG]. The element structure is as follows:

1428

```
1429     <NonRepudiation>  
1430         <Protocol version = "1.0">XMLDSIG</Protocol>
```

```
1431         <HashFunction>sha1</HashFunction>
1432         <SignatureAlgorithm>rsa</SignatureAlgorithm>
1433         <CertificateRef certId = "N03"/>
1434     </NonRepudiation>
1435
```

1436 If the *NonRepudiation* element is omitted, the *Messages* are not digitally signed.

1437
1438 Security at the document-exchange level applies to all *Messages* in both directions for *Business*
1439 *Transactions* for which security is enabled.

1440
1441 The *NonRepudiation* element is comprised of the following child elements:

- 1442 • a REQUIRED *Protocol* element,
- 1443 • a REQUIRED *HashFunction* (e.g. SHA1, MD5) element,
- 1444 • a REQUIRED *SignatureAlgorithm* element,
- 1445 • a REQUIRED *Certificate* element.

1446

1447 7.6.5.1 Protocol element

1448 The REQUIRED *Protocol* element identifies the technology that will be used to digitally sign a
1449 *Message*. It has a single IMPLIED *version* attribute whose value is a string that identifies the
1450 version of the specified technology. An example of the *Protocol* element follows:

```
1451         <Protocol version="2000/10/31">http://www.w3.org/2000/09/xmlldsig#
1452         </Protocol>
1453
```

1454

1455 7.6.5.2 HashFunction element

1456 The REQUIRED *HashFunction* element identifies the algorithm that is used to compute the
1457 digest of the *Message* being signed.

1458

1459 7.6.5.3 SignatureAlgorithm element

1460 The REQUIRED *SignatureAlgorithm* element identifies the algorithm that is used to compute
1461 the value of the digital signature.

1462

1463 7.6.5.4 CertificateRef element

1464 The REQUIRED *CertificateRef* element refers to one of the *Certificate* elements elsewhere
1465 within the *CPP* document, using the IMPLIED *certId* IDREF attribute.

1466

1467 7.6.6 DigitalEnvelope element

1468 The *DigitalEnvelope* element[DIGENV] is an encryption procedure in which the *Message* is
1469 encrypted by symmetric encryption (shared secret key) and the secret key is sent to the *Message*
1470 recipient encrypted with the recipient's public key. The element structure is:

```
1471 <DigitalEnvelope>
1472     <Protocol version = "2.0">S/MIME</Protocol>
1473     <EncryptionAlgorithm>rsa</EncryptionAlgorithm>
1474     <CertificateRef certId = "N03"/>
1475 </DigitalEnvelope>
1476
```

1477

1478 Security at the document-exchange level applies to all *Messages* in both directions for *Business*
1479 *Transactions* for which security is enabled.

1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529

7.6.6.1 Protocol element

The REQUIRED *Protocol* element identifies the security protocol to be used. The FIXED *version* attribute identifies the version of the protocol.

7.6.6.2 EncryptionAlgorithm element

The REQUIRED *EncryptionAlgorithm* element identifies the encryption algorithm to be used.

7.6.6.3 CertificateRef element

The REQUIRED *CertificateRef* element identifies the certificate to be used by means of its *certId* attribute. The IMPLIED *certId* attribute is an attribute of type [XML] IDREF, which refers to a matching ID attribute in a *Certificate* element elsewhere in the *CPP* or *CPA*.

7.6.7 NamespaceSupported element

The *NamespaceSupported* element identifies any namespace extensions supported by the messaging service implementation. Examples are Security Services Markup Language[S2ML] and Transaction Authority Markup Language[XAML]. For example, support for the S2ML namespace would be defined as follows:

```
<NamespaceSupported location = "http://www.s2ml.org/s2ml.xsd"
version = "0.8">http://www.s2ml.org/s2ml</NamespaceSupported>
```

7.7 Packaging element

The subtree of the *Packaging* element provides specific information about how the *Message Header* and payload constituent(s) are packaged for transmittal over the transport, including the crucial information about what document-level security packaging is used and the way in which security features have been applied. Typically the subtree under the *Packaging* element indicates the specific way in which constituent parts of the *Message* are organized. MIME processing capabilities are typically the capabilities or agreements described in this subtree. The *Packaging* element provides information about MIME content types, XML namespaces, security parameters, and MIME structure of the data that is exchanged between *Parties*.

Following is an example of the *Packaging* element:

```
<Packaging id="id">
<!--The Packaging triple MAY appear one or more times-->
  <ProcessingCapabilities parse="..." generate="..." />
  <SimplePart
    id="id" mimetype="type" /> <!--one or more-->
    <NamespaceSupported location = "" version="">
      URI
    </NamespaceSupported> <!--zero or more-->
  <!--The child of CompositeList is an enumeration of either
  Composite or Encapsulation. The enumeration MAY appear one
  or more time, with the two elements intermixed-->
  <CompositeList>
    <Composite mimetype="type"
      id="name"
      mimeparameters="parameter">
      <Constituent idref="name" />
```

```
1530         </Composite>
1531         <Encapsulation mimetype="type" id="name">
1532             <Constituent idref="name"/>
1533         </Encapsulation>
1534     </CompositeList>
1535 </Packaging>
```

1536

1537 See "Matching Packaging" in Appendix F for a more specific example.

1538

1539 The **Packaging** element has one attribute; the REQUIRED *id* attribute, with type ID. It is
1540 referred to in the **ServiceBinding** element and in the **Override** element, by using the IDREF
1541 attribute, *packageId*.

1542

1543 The child elements of the **Packaging** element are **ProcessingCapabilities**, **SimplePart**, and
1544 **CompositeList**. This set of elements MAY appear one or more times as a child of each
1545 **Packaging** element in a **CPP** and SHALL appear once as a child of each **Packaging** element in a
1546 **CPA**.

1547

1548 7.7.1 ProcessingCapabilities element

1549 The **ProcessingCapabilities** element has two attributes with REQUIRED Boolean values of
1550 either "true" or "false". The attributes are *parse* and *generate*. Normally, these attributes will
1551 both have values of "true" to indicate that the packaging constructs specified in the other child
1552 elements can be both produced as well as processed at the software **Message** service layer.
1553 At least one of the *generate* or *parse* attributes MUST be true.

1554

1555 7.7.2 SimplePart element

1556 The **SimplePart** element provides a repeatable list of the constituent parts, primarily identified by
1557 the MIME content-type value. The **SimplePart** element has two REQUIRED attributes: *id* and
1558 *mimetype*. The *id* attribute, type ID, provides the value that will be used later to reference this
1559 **Message** part when specifying how the parts are packaged into composites, if composite
1560 packaging is present. The *mimetype* attribute provides the actual value of content-type for the
1561 simple **Message** part being specified.

1562

1563 7.7.3 SimplePart element

1564 The **SimplePart** element can have zero or more **NamespaceSupported** elements. Each of these
1565 identifies any namespace extensions supported for the XML packaged in the parent simple body
1566 part. Examples include Security Services Markup Language[S2ML] and Transaction Authority
1567 Markup Language[XAML]. For example, support for the S2ML namespace would be defined as
1568 follows:

1569

```
1570     <NamespaceSupported location = "http://www.s2ml.org/s2ml.xsd"
1571     version = "0.8">http://www.s2ml.org/s2ml</NamespaceSupported>
```

1572

1573 7.7.4 CompositeList element

1574 The final child element of **Packaging** is **CompositeList**, which is a container for the specific way

1575 in which the simple parts are combined into groups (MIME multipart) or encapsulated within
1576 security-related MIME content-types. The *CompositeList* element MAY be omitted from
1577 *Packaging* when no security encapsulations or composite multipart are used. When the
1578 *CompositeList* element is present, the content model for the *CompositeList* element is a
1579 repeatable sequence of choices of *Composite* or *Encapsulation* elements. The *Composite* and
1580 *Encapsulation* elements MAY appear intermixed as desired.

1581
1582 The sequence in which the choices are presented is important because, given the recursive
1583 character of MIME packaging, composites or encapsulations MAY include previously
1584 mentioned composites (or rarely, encapsulations) in addition to the *Message* parts characterized
1585 within the *SimplePart* subtree. Therefore, the "top-level" packaging will be described last in the
1586 sequence.

1587
1588 The *Composite* element has the following attributes:

- 1589 • a REQUIRED *mimetype* attribute,
- 1590 • a REQUIRED *id* attribute,
- 1591 • an IMPLIED *mimeparameters* attribute.

1592
1593 The *mimetype* attribute provides the value of the MIME content-type for this *Message* part, and
1594 this will be some MIME composite type, such as "multipart/related" or "multipart/signed". The
1595 *id* attribute, type ID, provides a way to refer to this composite if it needs to be mentioned as a
1596 constituent of some later element in the sequence. The *mimeparameters* attribute provides the
1597 values of any significant MIME parameter (such as "type=application/vnd.eb+xml") that is
1598 needed to understand the processing demands of the content-type.

1599
1600 The *Composite* element has one child element, *Constituent*.

1601
1602 The *Constituent* element has one REQUIRED attribute, *idref*, type IDREF, and has an EMPTY
1603 content model. The *idref* attribute has as its value the value of the *id* attribute of a previous
1604 *Composite*, *Encapsulation*, or *SimplePart* element. The purpose of this sequence of
1605 *Constituents* is to indicate both the contents and the order of what is packaged within the current
1606 *Composite* or *Encapsulation*.

1607
1608 The *Encapsulation* element is typically used to indicate the use of MIME security mechanisms,
1609 such as [S/MIME] or Open-PGP[RFC2015]. A security body part can encapsulate a MIME part
1610 that has been previously characterized. For convenience, all such security structures are under
1611 the *Encapsulation* element, even when technically speaking the data is not "inside" the body
1612 part. (In other words, the so-called clear-signed or detached signature structures possible with
1613 MIME multipart/signed are for simplicity found under the *Encapsulation* element.)

1614
1615 The *Encapsulation* element has the following attributes:

- 1616 • a REQUIRED *mimetype* attribute,
- 1617 • a REQUIRED *id* attribute,
- 1618 • an IMPLIED *mimeparameters* attribute.

1619
1620 The *mimetype* attribute provides the value of the MIME content-type for this *Message* part, such

1621 as "application/pkcs7-mime." The *id* attribute, type ID, provides a way to refer to this
1622 encapsulation if it needs to be mentioned as a constituent of some later element in the sequence.
1623 The *mimeparameters* attribute provides the values of any significant MIME parameter(s)
1624 needed to understand the processing demands of the content-type.

1625
1626 Both the *Encapsulation* attribute and the *Composite* element have child elements consisting of a
1627 *Constituent* element or of a repeatable sequence of *Constituent* elements, respectively.
1628

1629 7.8 ds:Signature element

1630 The *CPP* MAY be digitally signed using technology that conforms with the XML Digital
1631 Signature specification[XMLDSIG]. The *ds:Signature* element is the root of a subtree of
1632 elements that MAY be used for signing the *CPP*. The syntax is:

```
1633  
1634     <ds:Signature>...</ds:Signature>  
1635
```

1636 The content of this element and any subelements are defined by the XML Digital Signature
1637 specification. See Section 8.7 for a detailed discussion. The following additional constraints on
1638 *ds:Signature* are imposed:

- 1639
1640 • A *CPP* MUST be considered invalid if any *ds:Signature* element fails core validation as
1641 defined by the XML Digital Signature specification[XMLDSIG].
1642
- 1643 • Whenever a *CPP* is signed, each *ds:Reference* element within a *ProcessSpecification*
1644 element MUST pass reference validation and each *ds:Signature* element MUST pass
1645 core validation.

1646
1647 NOTE: In case a *CPP* is unsigned, software MAY nonetheless validate the *ds:Reference*
1648 elements within *ProcessSpecification* elements and report any exceptions.
1649

1650 NOTE: Software for creation of *CPPs* and *CPAs* MAY recognize *ds:Signature* and
1651 automatically insert the element structure necessary to define signing of the *CPP* and *CPA*.
1652 Signature creation itself is a cryptographic process that is outside the scope of this
1653 specification.
1654

1655 NOTE: See non-normative note in Section 7.5.4.5 for a discussion of times at which validity
1656 tests MAY be made.
1657

1658 7.9 Comment Element

1659 The *CollaborationProtocolProfile* element MAY contain zero or more *Comment* elements. The
1660 *Comment* element is a textual note that MAY be added to serve any purpose the author desires.
1661 The language of the *Comment* is identified by a REQUIRED *xml:lang* attribute. The *xml:lang*
1662 attribute MUST comply with the rules for identifying languages specified in [XML]. If multiple
1663 *Comment* elements are present, each SHOULD have a unique *xml:lang* attribute value. An
1664 example of a *Comment* element follows:
1665

1666 <Comment xml:lang="en-gb">yadda yadda, blah blah</Comment>

1667

1668 When a *CPA* is composed from two *CPPs*, all ***Comment*** elements from both *CPPs* SHALL be
1669 included in the *CPA* unless the two *Parties* agree otherwise.

1670 8 CPA Definition

1671 A *Collaboration-Protocol Agreement (CPA)* defines the capabilities that two *Parties* must agree
 1672 to enable them to engage in electronic *Business* for the purposes of the particular *CPA*. This
 1673 section defines and discusses the details of the *CPA*. The discussion is illustrated with some
 1674 XML fragments.

1675
 1676 Most of the XML elements in this section are described in detail in section 7, "CPP Definition".
 1677 In general, this section does not repeat that information. The discussions in this section are
 1678 limited to those elements that are not in the *CPP* or for which additional discussion is required in
 1679 the *CPA* context. See also Appendix C and Appendix D for the DTD and XML Schema,
 1680 respectively, and Appendix B for an example of a *CPA* document.

1681

1682 8.1 CPA Structure

1683 Following is the overall structure of the *CPA*:

1684

```

1685 <CollaborationProtocolAgreement
1686     xmlns="http://www.ebxml.org/namespaces/tradePartner"
1687     xmlns:bpm="http://www.ebxml.org/namespaces/businessProcess"
1688     xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
1689     xmlns:xlink = "http://www.w3.org/1999/xlink"
1690     cpaid="YoursAndMyCPA"
1691     version="1.2">
1692     <Status value = "proposed"/>
1693     <Start>1988-04-07T18:39:09</Start>
1694     <End>1990-04-07T18:40:00</End>
1695     <!--ConversationConstraints MAY appear 0 or 1 times-->
1696     <ConversationConstraints invocationLimit = "100"
1697         concurrentConversations = "4"/>
1698     <PartyInfo>
1699         ...
1700     </PartyInfo>
1701     <PartyInfo>
1702         ...
1703     </PartyInfo>
1704     <Packaging id="N20"> <!--one or more-->
1705         ...
1706     </Packaging>
1707     <!--ds:signature MAY appear 0 or more times-->
1708     <ds:Signature>any combination of text and elements
1709     </ds:Signature>
1710     <Comment xml:lang="en-gb">any text</Comment> <!--zero or more-->
1711 </CollaborationProtocolAgreement>
1712

```

1713 8.2 CollaborationProtocolAgreement Element

1714 The *CollaborationProtocolAgreement* element is the root element of a *CPA*. It has a
 1715 REQUIRED *cpaid* attribute of type [XML] CDATA that supplies a unique identifier for the

1716 document. The value of the *cpaid* attribute SHALL be assigned by one *Party* and used by both.
1717 It is RECOMMENDED that the value of the *cpaid* attribute be a URI. The value of the *cpaid*
1718 attribute MAY be used as the value of the *CPAId* element in the ebXML *Message*
1719 *Header*[ebMS].

1720

1721 NOTE: Each *Party* MAY associate a local identifier with the *cpaid* attribute.

1722

1723 In addition, the *CollaborationProtocolAgreement* element has an IMPLIED *version* attribute.
1724 This attribute indicates the version of the *CPA*. Its purpose is to provide versioning capabilities
1725 for an instance of a *CPA* as it undergoes negotiation between the two parties. The *version*
1726 attribute SHOULD also be used to provide versioning capability for a *CPA* that has been
1727 deployed and then modified. The value of the *version* attribute SHOULD be a string
1728 representation of a numeric value such as "1.0" or "2.3". The value of the version string
1729 SHOULD be changed with each change made to the *CPA* document both during negotiation and
1730 after it has been deployed.

1731

1732 NOTE: The method of assigning version identifiers is left to the implementation.

1733

1734 The *CollaborationProtocolAgreement* element has REQUIRED [XML] Namespace[XMLNS]
1735 declarations that are defined in Section 7, "CPP Definition".

1736

1737 The *CollaborationProtocolAgreement* element is comprised of the following child elements,
1738 each of which is described in greater detail in subsequent sections:

1739

- a REQUIRED *Status* element that identifies the state of the process that creates the *CPA*,
- a REQUIRED *Start* element that records the date and time that the *CPA* goes into effect,
- a REQUIRED *End* element that records the date and time after which the *CPA* must be renegotiated by the *Parties*,
- zero or one *ConversationConstraints* element that documents certain agreements about conversation processing,
- two REQUIRED *PartyInfo* elements, one for each *Party* to the *CPA*,
- one or more *ds:Signature* elements that provide signing of the *CPA* using the XML Digital Signature[XMLDSIG] standard.

1740

1741

1742

1743

1744

1745

1746

1747

1748

1749

1750

1751 8.3 Status Element

1752 The *Status* element records the state of the composition/negotiation process that creates the *CPA*.
1753 An example of the *Status* element follows:

1754

```
1755 <Status value = "proposed"/>
```

1756

1757 The *Status* element has a REQUIRED *value* attribute that records the current state of
1758 composition of the *CPA*. The value of this attribute is an enumeration of the following possible
1759 values:

1760

- "proposed", meaning that the *CPA* is still being negotiated by the *Parties*,

- 1761 • "agreed", meaning that the contents of the *CPA* have been agreed to by both *Parties*,
1762 • "signed", meaning that the *CPA* has been "signed" by the *Parties*. This "signing"
1763 MAY take the form of a digital signature that is described in section 8.7 below.
1764

1765 NOTE: The *Status* element MAY be used by a *CPA* composition and negotiation tool to
1766 assist it in the process of building a *CPA*.
1767

1768 **8.4 CPA Lifetime**

1769 The lifetime of the *CPA* is given by the *Start* and *End* elements. The syntax is:

1770 <Start>1988-04-07T18:39:09</Start>
1771 <End>1990-04-07T18:40:00</End>
1772
1773

1774 **8.4.1 Start element**

1775 The *Start* element specifies the starting date and time of the *CPA*. The *Start* element SHALL be
1776 a string value that conforms to the content model of a canonical timeInstant as defined in the
1777 XML Schema Datatypes Specification[XMLSCHEMA-2]. For example, to indicate 1:20 pm
1778 UTC (Coordinated Universal Time) on May 31, 1999, a *Start* element would have the following
1779 value:

1780 1999-05-31T13:20:00Z
1781
1782

1783 The *Start* element SHALL be represented as Coordinated Universal Time (UTC).
1784

1785 **8.4.2 End element**

1786 The *End* element specifies the ending date and time of the *CPA*. The *End* element SHALL be a
1787 string value that conforms to the content model of a canonical timeInstant as defined in the XML
1788 Schema Datatypes Specification[XMLSCHEMA-2]. For example, to indicate 1:20 pm UTC
1789 (Coordinated Universal Time) on May 31, 1999, an *End* element would have the following
1790 value:

1791 1999-05-31T13:20:00Z
1792
1793

1794 The *End* element SHALL be represented as Coordinated Universal Time (UTC).
1795

1796 When the end of the *CPA*'s lifetime is reached, any *Business Transactions* that are still in
1797 progress SHALL be allowed to complete and no new *Business Transactions* SHALL be started.
1798 When all in-progress *Business Transactions* on each conversation are completed, the
1799 *Conversation* shall be terminated whether or not it was completed.
1800

1801 NOTE: It should be understood that if a *Business* application defines a conversation as
1802 consisting of multiple *Business Transactions*, such a conversation MAY be terminated
1803 with no error indication when the end of the lifetime is reached. The run-time system
1804 could provide an error indication to the application.
1805

1806 NOTE: It should be understood that it MAY not be feasible to wait for outstanding
1807 conversations to terminate before ending the *CPA* since there is no limit on how long a
1808 conversation MAY last.

1809
1810 NOTE: The run-time system SHOULD return an error indication to both *Parties* when a
1811 new *Business Transaction* is started under this *CPA* after the date and time specified in
1812 the *End* element.

1813

1814 8.5 ConversationConstraints Element

1815

1816 The *ConversationConstraints* element places limits on the number of conversations under the
1817 *CPA*. An example of this element follows:

1818

```
1819 <ConversationConstraints invocationLimit = "100"  
1820 concurrentConversations = "4"/>
```

1821

1822 The *ConversationConstraints* element has the following attributes:

- 1823 • an IMPLIED *invocationLimit* attribute,
- 1824 • an IMPLIED *concurrentConversations* attribute.

1825

1826 8.5.1 invocationLimit attribute

1827 The *invocationLimit* attribute defines the maximum number of conversations that can be
1828 processed under the *CPA*. When this number has been reached, the *CPA* is terminated and must
1829 be renegotiated. If no value is specified, there is no upper limit on the number of conversations
1830 and the lifetime of the *CPA* is controlled solely by the *End* element.

1831

1832 NOTE: The *invocationLimit* attribute sets a limit on the number of units of *Business* that
1833 can be performed under the *CPA*. It is a *Business* parameter, not a performance parameter.

1834

1835 8.5.2 concurrentConversations attribute

1836 The *concurrentConversations* attribute defines the maximum number of conversations that can
1837 be in process under this *CPA* at the same time. If no value is specified, processing of concurrent
1838 conversations is strictly a local matter.

1839

1840 NOTE: The *concurrentConversations* attribute provides a parameter for the *Parties* to use
1841 when it is necessary to limit the number of conversations that can be concurrently processed
1842 under a particular *CPA*. For example, the back-end process might only support a limited
1843 number of concurrent conversations. If a request for a new conversation is received when
1844 the maximum number of conversations allowed under this *CPA* is already in process, an
1845 implementation MAY reject the new conversation or MAY enqueue the request until an
1846 existing conversation ends. If no value is given for *concurrentConversations*, how to handle
1847 a request for a new conversation for which there is no capacity is a local implementation
1848 matter.

1849

1850 8.6 PartyInfo Element

1851 The general characteristics of the *PartyInfo* element are discussed in sections 7.5 and **Error!**
1852 **Reference source not found.** .

1853
1854 The *CPA* SHALL have one *PartyInfo* element for each *Party* to the *CPA*. The *PartyInfo*
1855 element specifies the *Parties'* agreed terms for engaging in the *Business Collaborations* defined
1856 by the *Process-Specification* documents referenced by the *CPA*. If a *CPP* has more than one
1857 *PartyInfo* element, the appropriate *PartyInfo* element SHALL be selected from each *CPP* when
1858 composing a *CPA*.

1859
1860 In the *CPA*, there SHALL be one *PartyId* element under each *PartyInfo* element. The value of
1861 this element is the same as the value of the *PartyId* element in the ebXML *Message Service*
1862 specification[ebMS]. One *PartyId* element SHALL be used within a *To* or *From Header*
1863 element of an ebXML *Message*.

1864

1865 8.6.1 ProcessSpecification element

1866 The *ProcessSpecification* element identifies the *Business Collaboration* that the two *Parties*
1867 have agreed to perform. There MAY be one or more *ProcessSpecification* elements in a *CPA*.
1868 Each SHALL be a child element of a separate *CollaborationRole* element. See the discussion in
1869 Section 7.5.3.

1870

1871 8.7 ds:Signature Element

1872 A *CPA* document MAY be digitally signed by one or more of the *Parties* as a means of ensuring
1873 its integrity as well as a means of expressing the agreement just as a corporate officer's signature
1874 would do for a paper document. If signatures are being used to digitally sign an ebXML *CPA* or
1875 *CPP* document, then it is strongly RECOMMENDED that [XMLDSIG] be used to digitally sign
1876 the document. The *ds:Signature* element is the root of a subtree of elements that MAY be used
1877 for signing the *CPP*. The syntax is:

1878

```
1879 <ds:Signature>...</ds:Signature>
```

1880

1881 The content of this element and any subelements are defined by the XML Digital Signature
1882 specification[XMLDSIG]. The following additional constraints on *ds:Signature* are imposed:

1883

- 1884 • A *CPA* MUST be considered invalid if any *ds:Signature* fails core validation as defined
1885 by the XML Digital Signature specification.

1886

- 1887 • Whenever a *CPA* is signed, each *ds:Reference* within a *ProcessSpecification* MUST
1888 pass reference validation and each *ds:Signature* MUST pass core validation.

1889

1890 NOTE: In case a *CPA* is unsigned, software MAY nonetheless validate the *ds:Reference*
1891 elements within *ProcessSpecification* elements and report any exceptions.

1892

1893 NOTE: Software for creation of *CPPs* and *CPAs* MAY recognize *ds:Signature* and

1894 automatically insert the element structure necessary to define signing of the *CPP* and *CPA*.
1895 Signature creation itself is a cryptographic process that is outside the scope of this
1896 specification.

1897
1898 NOTE: See non-normative note in section 7.5.4.5 for a discussion of times at which a *CPA*
1899 MAY be validated.
1900

1901 **8.7.1 Persistent Digital Signature**

1902 If [XMLDSIG] is used to sign an ebXML *CPP* or *CPA*, the process defined in this section of the
1903 specification SHALL be used.
1904

1905 **8.7.1.1 Signature Generation**

1906 Following are the steps to create a digital signature:

- 1907 1. Create a *SignedInfo* element, a child element of *ds:Signature*. *SignedInfo* SHALL have
1908 child elements *SignatureMethod*, *CanonicalizationMethod*, and *Reference* as prescribed by
1909 [XMLDSIG].
- 1910 2. Canonicalize and then calculate the *SignatureValue* over *SignedInfo* based on algorithms
1911 specified in *SignedInfo* as specified in [XMLDSIG].
- 1912 3. Construct the *Signature* element that includes the *SignedInfo*, *KeyInfo*
1913 (RECOMMENDED), and *SignatureValue* elements as specified in [XMLDSIG].
- 1914 4. Include the namespace qualified *Signature* element in the document just signed, following
1915 the last *PartyInfo* element.

1916 1917 **8.7.1.2 ds:SignedInfo element**

1918 The *ds:SignedInfo* element SHALL be comprised of zero or one *ds:CanonicalizationMethod*
1919 element, the *ds:SignatureMethod* element, and one or more *ds:Reference* elements.
1920

1921 **8.7.1.3 ds:CanonicalizationMethod element**

1922 The *ds:CanonicalizationMethod* element is defined as OPTIONAL in [XMLDSIG], meaning
1923 that the element need not appear in an instance of a *ds:SignedInfo* element. The default
1924 canonicalization method that is applied to the data to be signed is [XMLC14N] in the absence of
1925 a *ds:CanonicalizationMethod* element that specifies otherwise. This default SHALL also serve
1926 as the default canonicalization method for the ebXML *CPP* and *CPA* documents.
1927

1928 **8.7.1.4 ds:SignatureMethod element**

1929 The *ds:SignatureMethod* element SHALL be present and SHALL have an *Algorithm* attribute.
1930 The RECOMMENDED value for the *Algorithm* attribute is:

1931 <http://www.w3.org/2000/09/xmlsig#dsa-sha1>
1932

1933 This RECOMMENDED value SHALL be supported by all compliant ebXML *CPP* or *CPA*
1934 software implementations.
1935

1936 1937 **8.7.1.5 ds:Reference element**

1938 The *ds:Reference* element for the *CPP* or *CPA* document SHALL have a REQUIRED URI

1939 attribute value of "" to provide for the signature to be applied to the document that contains the
 1940 **ds:Signature** element (the *CPA* or *CPP* document). The **ds:Reference** element for the *CPP* or
 1941 *CPA* document MAY include an IMPLIED *type* attribute that has a value of:

1942
 1943 "http://www.w3.org/2000/09/xmldsig#Object"
 1944

1945 in accordance with [XMLDSIG]. This attribute is purely informative. It MAY be omitted.
 1946 Implementations of software designed to author or process an ebXML *CPA* or *CPP* document
 1947 SHALL be prepared to handle either case. The **ds:Reference** element MAY include the *id*
 1948 attribute, type ID, by which this **ds:Reference** element MAY be referenced from a **ds:Signature**
 1949 element.

1950
 1951 **8.7.1.6 ds:Transform element**

1952 The **ds:Reference** element for the *CPA* or *CPP* document SHALL include a descendant
 1953 **ds:Transform** element that excludes the containing **ds:Signature** element and all its descendants.
 1954 This exclusion is achieved by means of specifying the **ds:Algorithm** attribute of the **Transform**
 1955 element as

1956 "http://www.w3.org/2000/09/xmldsig#enveloped-signature".
 1957

1958 For example:

```
1959 <ds:Reference ds:URI="">
1960   <ds:Transforms>
1961     <ds:Transform
1962       ds:Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
1963     </ds:Transforms>
1964     <ds:DigestMethod
1965       ds:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1966       <ds:DigestValue>...</ds:DigestValue>
1967   </ds:Reference>
```

1968
 1969 **8.7.1.7 ds:Xpath element**

1970 The **ds:Transform** element SHALL include a **ds:Algorithm** attribute that has a value of:

1971 http://www.w3.org/2000/09/xmldsig#enveloped-signature
 1972

1973 NOTE: When digitally signing a *CPA*, it is RECOMMENDED that each *Party* sign the
 1974 document in accordance with the process described above. The first *Party* that signs the
 1975 *CPA* will sign only the *CPA* contents, excluding their own signature. The second *Party*
 1976 signs over the contents of the *CPA* as well as the **ds:Signature** element that contains the
 1977 first *Party's* signature. It MAY be necessary that a notary sign over both signatures so as to
 1978 provide for cryptographic closure.
 1979

1980 **8.8 Comment element**

1981 The **CollaborationProtocolAgreement** element MAY contain zero or more **Comment** elements.
 1982 See section 7.9 for details of the syntax of the **Comment** element.
 1983

1984 **8.9 Composing a CPA from Two CPPs**

1985 This section discusses normative issues in composing a *CPA* from two *CPPs*. See also Appendix
 1986 F , "Composing a CPA from Two CPPs (Non-Normative)".

1987

1988 **8.9.1 ID Attribute Duplication**

1989 In composing a *CPA* from two *CPPs*, there is a hazard that ID attributes from the two *CPPs*
 1990 might have duplicate values. When a *CPA* is composed from two *CPPs*, duplicate ID attribute
 1991 values SHALL be tested for. If a duplicate ID attribute value is present, one of the duplicates
 1992 shall be given a new value and the corresponding IDREF attribute values from the corresponding
 1993 *CPP* SHALL be corrected.

1994

1995 **8.10 Modifying Parameters of the Process-Specification Document Based on**
 1996 **Information in the CPA**

1997 A *Process-Specification* document contains a number of parameters, expressed as XML
 1998 attributes. An example is the security attributes that are counterparts of the attributes of the *CPA*
 1999 *Characteristics* element. The values of these attributes can be considered to be default values or
 2000 recommendations. When a *CPA* is created, the *Parties* MAY decide to accept the
 2001 recommendations in the *Process-Specification* or they MAY agree on values of these parameters
 2002 that better reflect their needs.

2003

2004 When a *CPA* is used to configure a run-time system, choices specified in the *CPA* MUST always
 2005 assume precedence over choices specified in the referenced *Process-Specification* document. In
 2006 particular, all choices expressed in a *CPA*'s *Characteristics* and *Packaging* elements MUST be
 2007 implemented as agreed to by the *Parties*. These choices SHALL override the default values
 2008 expressed in the *Process-Specification* document. The process of installing the information from
 2009 the *CPA* and *Process-Specification* document MUST verify that all of the resulting choices are
 2010 mutually consistent and MUST signal an error if they are not.

2011

2012 NOTE: There are several ways of overriding the information in the *Process-*
 2013 *Specification* document by information from the *CPA*. For example:

2014

- 2015 • The CPA composition tool can create a separate copy of the Process-Specification
 2016 document. The tool can then directly modify the *Process-Specification* document
 2017 with information from the *CPA*. One advantage of this method is that the override
 2018 process is performed entirely by the *CPA* composition tool. A second advantage is
 2019 that with a separate copy of the *Process-Specification* document associated with the
 2020 particular *CPA*, there is no exposure to modifications of the *Process-Specification*
 2021 document between the time that the *CPA* is created and the time it is installed in the
 2022 *Parties'* systems.
- 2023 • A *CPA* installation tool can dynamically override parameters in the *Process-*
 2024 *Specification* document using information from the corresponding parameters in the
 2025 *CPA* at the time the *CPA* and *Process-Specification* document are installed in the
 2026 *Parties'* systems. This eliminates the need to create a separate copy of the *Process-*
 2027 *Specification* document.
- 2028 • Other possible methods might be based on XSLT transformations of the parameter
 2029 information in the *CPA* and/or the *Process-Specification* document.

2030

9 References

2031 Some references listed below specify functions for which specific XML definitions are provided
2032 in the *CPP* and *CPA*. Other specifications are referred to in this specification in the sense that
2033 they are represented by keywords for which the *Parties* to the *CPA* MAY obtain plug-ins or
2034 write custom support software but do not require specific XML element sets in the *CPP* and
2035 *CPA*.

2036
2037 In a few cases, the only available specification for a function is a proprietary specification.
2038 These are indicated by notes within the citations below.

2039 [ccOVER] ebXML Core Components and Business Process Document Overview,
2040 <http://www.ebxml.org>.

2041
2042
2043 [DIGENV] Digital Envelope, RSA Laboratories, <http://www.rsasecurity.com/rsalabs/>. NOTE:
2044 At this time, the only available specification for digital envelope appears to be the RSA
2045 Laboratories specification.

2046
2047 [ebBPSS] ebXML Business Process Specification Schema, <http://www.ebxml.org>.

2048
2049 [ebGLOSS] ebXML Glossary, <http://www.ebxml.org>.

2050
2051 [ebMS] ebXML Message Service Specification, <http://www.ebxml.org>.

2052
2053 [ebRS] ebXML Registry Services Specification, <http://www.ebxml.org>.

2054
2055 [ebTA] ebXML Technical Architecture Specification, <http://www.ebxml.org>.

2056
2057 [HTTP] Hypertext Transfer Protocol, Internet Engineering Task Force RFC2616.

2058
2059 [IPSEC] IP Security Document Roadmap, Internet Engineering Task Force RFC 2411.

2060
2061 [ISO6523] Structure for the Identification of Organizations and Organization Parts, International
2062 Standards Organization ISO-6523.

2063
2064 [MIME] MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying
2065 and Describing the Format of Internet *Message* Bodies. Internet Engineering Task Force RFC
2066 1521.

2067
2068 [RFC822] Standard for the Format of ARPA Internet Text Messages, Internet Engineering Task
2069 Force RFC 822.

2070
2071 [RFC959] File Transfer Protocol (FTP), Internet Engineering Task Force RFC 959.

2072
2073 [RFC1123] Requirements for Internet Hosts -- Application and Support, R. Braden, Internet
2074 Engineering Task Force, October 1989.

- 2075
2076 [RFC1579] Firewall-Friendly FTP, S. Bellovin, Internet Engineering Task Force, February 1994.
2077
2078 [RFC2015] MIME Security with Pretty Good Privacy, M. Elkins, Internet Engineering Task
2079 Force, RFC 2015.
2080
2081 [RFC2119] Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering
2082 Task Force RFC 2119.
2083
2084 [RFC2396] Uniform Resource Identifiers (URI): Generic Syntax; T. Berners-Lee, R. Fielding, L.
2085 Masinter - August 1998.
2086
2087 [S/MIME] S/MIME Version 3 Message Specification, Internet Engineering Task Force RFC
2088 2633.
2089
2090 [S2ML] Security Services Markup Language, <http://s2ml.org/>.
2091
2092 [SMTP] Simple Mail Transfer Protocol, Internet Engineering Task Force RFC 821.
2093
2094 [SSL] Secure Sockets Layer, Netscape Communications Corp. <http://developer.netscape.com>.
2095 NOTE: At this time, it appears that the Netscape specification is the only available specification
2096 of SSL. Work is in progress in IETF on "Transport Layer Security", which is intended as a
2097 replacement for SSL.
2098
2099 [XAML] Transaction Authority Markup Language, <http://xaml.org/>.
2100
2101 [XLINK] XML Linking Language, <http://www.w3.org/TR/xlink/>.
2102
2103 [XML] Extensible Markup Language (XML), World Wide Web Consortium,
2104 <http://www.w3.org>.
2105
2106 [XMLC14N] Canonical XML, Ver. 1.0, <http://www.w3.org/TR/XML-C14N/>.
2107
2108 [XMLDSIG] XML Signature Syntax and Processing, Worldwide Web Consortium,
2109 <http://www.w3.org/TR/xmlsig-core/>.
2110
2111 [XMLNS] Namespaces in XML, T. Bray, D. Hollander, and A. Layman, Jan. 1999,
2112 <http://www.w3.org/TR/REC-xml-names/>.
2113
2114 [XMLSCHEMA-1] XML Schema Part 1: Structures, <http://www.w3.org/TR/xmlschema-1/>.
2115
2116 [XMLSCHEMA-2] XML Schema Part 2: Datatypes,
2117 <http://www.w3.org/TR/xmlschema-2/>.
2118
2119 [XPOINTER] XML Pointer Language, ver. 1.0, <http://www.w3.org/TR/xptr>.

2120 10 Conformance

2121 In order to conform to this specification, an implementation:

- 2122 a) SHALL support all the functional and interface requirements defined in this specification,
2123 b) SHALL NOT specify any requirements that would contradict or cause non-conformance
2124 to this specification.

2125

2126 A conforming implementation SHALL satisfy the conformance requirements of the applicable
2127 parts of this specification,

2128

2129 An implementation of a tool or service that creates or maintains ebXML *CPP* or *CPA* instance
2130 documents SHALL be determined to be conformant by validation of the *CPP* or *CPA* instance
2131 documents, created or modified by said tool or service, against the XML
2132 Schema[XMLSCHEMA-1] definition of the *CPP* or *CPA* in Appendix D and available from

2133

2134 http://www.ebxml.org/schemas/cpp-cpa-v1_0.xsd

2135

2136 by using two or more validating XML Schema parsers that conform to the W3C XML Schema
2137 specifications[XMLSCHEMA-1,XMLSCHEMA-2].

2138

2139 The objective of conformance testing is to determine whether an implementation being tested
2140 conforms to the requirements stated in this specification. Conformance testing enables vendors to
2141 implement compatible and interoperable systems. Implementations and applications SHALL be
2142 tested using available test suites to verify their conformance to this specification.

2143

2144 Publicly available test suites from vendor neutral organizations such as OASIS and the U.S.A.
2145 National Institute of Science and Technology (NIST) SHOULD be used to verify the
2146 conformance of implementations, applications, and components claiming conformance to this
2147 specification. Open-source reference implementations MAY be available to allow vendors to test
2148 their products for interface compatibility, conformance, and interoperability.

2149

2150

2151

2152

2153 11 Disclaimer

2154 The views and specification expressed in this document are those of the authors and are not
2155 necessarily those of their employers. The authors and their employers specifically disclaim
2156 responsibility for any problems arising from correct or incorrect implementation or use of this
2157 design.

2158 12 Contact Information

2159 Martin W. Sachs (Team Leader)
2160 IBM T. J. Watson Research Center
2161 P.O.B. 704
2162 Yorktown Hts, NY 10598
2163 USA
2164 Phone: 914-784-7287
2165 email: mwsachs@us.ibm.com
2166

2167 Chris Ferris
2168 XML Technology Development
2169 Sun Microsystems, Inc
2170 One Network Drive
2171 Burlington, Ma 01824-0903
2172 USA
2173 Phone: 781-442-3063
2174 email: chris.ferris@east.sun.com
2175

2176 Dale W. Moberg
2177 Cyclone Commerce
2178 17767 North Perimeter Dr., Suite 103
2179 Scottsdale, AZ 85255
2180 USA
2181 Phone: 480-627-1800
2182 email: dmoberg@columbus.rr.com
2183

2184 Tony Weida
2185 Edifecs
2186 2310 130th Ave. NE, Suite 100
2187 Bellevue, WA 98005
2188 USA
2189 Phone: 212-678-5265
2190 email: TonyW@edifecs.com

2191 Copyright Statement

2192

2193 Copyright © ebXML 2001. All Rights Reserved.

2194

2195 This document and translations of it MAY be copied and furnished to others, and derivative
2196 works that comment on or otherwise explain it or assist in its implementation MAY be prepared,
2197 copied, published and distributed, in whole or in part, without restriction of any kind, provided
2198 that the above copyright notice and this paragraph are included on all such copies and derivative
2199 works. However, this document itself MAY not be modified in any way, such as by removing
2200 the copyright notice or references to ebXML, UN/CEFACT, or OASIS, except as required to
2201 translate it into languages other than English.

2202

2203 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
2204 successors or assigns.

2205

2206 This document and the information contained herein is provided on an "AS IS" basis and
2207 ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
2208 NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN
2209 WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
2210 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

2211

2212 **Appendix A Example of CPP Document (Non-Normative)**

2213 This example is available as an ASCII file at

2214 http://ebxml.org/project_teams/trade_partner/cpp-example.xml

```

2215
2216 <?xml version="1.0" encoding="UTF-8"?>
2217 <tp:CollaborationProtocolProfile
2218 xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"
2219 xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2220 xmlns:xlink="http://www.w3.org/1999/xlink"
2221 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2222 xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
2223 http://ebxml.org/project_teams/trade_partner/cpp-cpa-095.xsd" tp:version="1.1">
2224   <tp:PartyInfo>
2225     <tp:PartyId tp:type="DUNS">123456789</tp:PartyId>
2226     <tp:PartyRef tp:href="http://example.com/about.html"/>
2227     <tp:CollaborationRole tp:id="N00">
2228       <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
2229 xlink:type="locator" xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
2230       <tp:Role tp:name="buyer" xlink:type="simple"
2231 xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>
2232       <tp:CertificateRef tp:certId="N03"/>
2233       <tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402"
2234 tp:name="buyerService">
2235         <tp:Override tp:action="orderConfirm" tp:channelId="N08"
2236 tp:packageId="N0402" xlink:href="http://ebxml.org/processes/buySell.xml#orderConfirm"
2237 xlink:type="simple"/>
2238       </tp:ServiceBinding>
2239     </tp:CollaborationRole>
2240     <tp:Certificate tp:certId="N03">
2241       <ds:KeyInfo/>
2242     </tp:Certificate>
2243     <tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
2244 tp:docExchangeId="N06">
2245       <tp:Characteristics tp:syncReplyMode="none"
2246 tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
2247 tp:secureTransport="true" tp:confidentiality="true" tp:authenticated="true"
2248 tp:authorized="false"/>
2249     </tp:DeliveryChannel>
2250     <tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
2251 tp:docExchangeId="N06">
2252       <tp:Characteristics tp:syncReplyMode="none"
2253 tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
2254 tp:secureTransport="false" tp:confidentiality="true" tp:authenticated="true"
2255 tp:authorized="false"/>
2256     </tp:DeliveryChannel>
2257     <tp:Transport tp:transportId="N05">
2258       <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2259       <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
2260       <tp:Endpoint
2261 tp:uri="https://www.example.com/servlets/ebxmlhandler" tp:type="allPurpose"/>
2262       <tp:TransportSecurity>
2263         <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
2264         <tp:CertificateRef tp:certId="N03"/>
2265       </tp:TransportSecurity>
2266     </tp:Transport>
2267     <tp:Transport tp:transportId="N18">
2268       <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2269       <tp:ReceivingProtocol tp:version="1.1">SMTP</tp:ReceivingProtocol>
2270       <tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
2271 tp:type="allPurpose"/>
2272     </tp:Transport>
2273     <tp:DocExchange tp:docExchangeId="N06">
2274       <tp:ebXMLBinding tp:version="0.98b">
2275         <tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
2276 tp:idempotency="true" tp:messageOrderSemantics="Guaranteed">
2277           <tp:Retries>5</tp:Retries>
2278           <tp:RetryInterval>30</tp:RetryInterval>

```

```

2279         <tp:PersistDuration>P1D</tp:PersistDuration>
2280     </tp:ReliableMessaging>
2281     <tp:NonRepudiation>
2282
2283     <tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
2284
2285     <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
2286
2287     <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
2288     sha1</tp:SignatureAlgorithm>
2289         <tp:CertificateRef tp:certId="N03"/>
2290     </tp:NonRepudiation>
2291     <tp:DigitalEnvelope>
2292         <tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
2293         <tp:EncryptionAlgorithm>DES-
2294     CBC</tp:EncryptionAlgorithm>
2295         <tp:CertificateRef tp:certId="N03"/>
2296     </tp:DigitalEnvelope>
2297 </tp:ebXMLBinding>
2298 </tp:DocExchange>
2299 </tp:PartyInfo>
2300 <tp:Packaging tp:id="N0402">
2301     <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>
2302     <tp:SimplePart tp:id="N40" tp:mimetype="text/xml">
2303         <tp:NamespaceSupported
2304     tp:location="http://ebxml.org/project_teams/transport/messageService.xsd"
2305     tp:version="0.98b">http://www.ebxml.org/namespaces/messageService</tp:NamespaceSupport
2306     ed>
2307         <tp:NamespaceSupported
2308     tp:location="http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"
2309     tp:version="1.0">http://www.w3.org/2000/09/xmldsig</tp:NamespaceSupported>
2310     </tp:SimplePart>
2311     <tp:CompositeList>
2312         <tp:Composite tp:id="N42" tp:mimetype="multipart/related"
2313     tp:mimeparameters="type=text/xml;">
2314             <tp:Constituent tp:idref="N40"/>
2315             <tp:Constituent tp:idref="N41"/>
2316         </tp:Composite>
2317         <tp:Encapsulation tp:id="N41" tp:mimetype="multipart/signed1"
2318     tp:mimeparameters="charset=UTF-8;">
2319             <tp:Constituent tp:idref="N40"/>
2320         </tp:Encapsulation>
2321     </tp:CompositeList>
2322 </tp:Packaging>
2323     <tp:Comment tp:xml_lang="en-us">buy/sell agreement between example.com and
2324     contrived-example.com</tp:Comment>
2325 </tp:CollaborationProtocolProfile>

```

2326 **Appendix B Example of CPA Document (Non-Normative)**

2327 The example in this appendix is to be parsed with an XML Schema parser. It is available as an
 2328 ASCII file at

2329 http://ebxml.org/project_teams/trade_partner/cpa-example.xml

2330

2331 An example that can be parsed with the DTD is available at:

2332 http://ebxml.org/project_teams/trade_partner/cpa-example-dtd.xml

2333

2334 NOTE: Two separate examples of the CPA are needed because at least some existing tools
 2335 require the DTD to have a `<!DOCTYPE . . . >` to assign the DTD and not to have a
 2336 namespace qualifier.

2337

```

2338 <?xml version="1.0"?>
2339 <!-- edited with XML Spy v3.5 (http://www.xmlspy.com) by christopher ferris (sun
2340 microsystems, inc) -->
2341 <tp:CollaborationProtocolAgreement
2342 xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"
2343 xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2344 xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
2345 http://ebxml.org/project_teams/trade_partner/cpp-cpa-095.xsd"
2346 xmlns:xlink="http://www.w3.org/1999/xlink"
2347 xmlns:ds="http://www.w3.org/2000/09/xmldsig#" tp:cpaid="uri:yoursandmycpa"
2348 tp:version="1.2">
2349     <tp:Status tp:value="proposed"/>
2350     <tp:Start>2001-05-20T07:21:00Z</tp:Start>
2351     <tp:End>2002-05-20T07:21:00Z</tp:End>
2352     <tp:ConversationConstraints tp:invocationLimit="100"
2353 tp:concurrentConversations="100"/>
2354     <tp:PartyInfo>
2355         <tp:PartyId tp:type="DUNS">123456789</tp:PartyId>
2356         <tp:PartyRef xlink:href="http://example.com/about.html"/>
2357         <tp:CollaborationRole tp:id="N00">
2358             <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
2359 xlink:type="simple" xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
2360             <tp:Role tp:name="buyer" xlink:type="simple"
2361 xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>
2362             <tp:CertificateRef tp:certId="N03"/>
2363             <tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402"
2364 tp:name="buyerService">
2365                 <tp:Override tp:action="orderConfirm" tp:channelId="N08"
2366 tp:packageId="N0402" xlink:href="http://ebxml.org/processes/buySell.xml#orderConfirm"
2367 xlink:type="simple"/>
2368             </tp:ServiceBinding>
2369             </tp:CollaborationRole>
2370             <tp:Certificate tp:certId="N03">
2371                 <ds:KeyInfo/>
2372             </tp:Certificate>
2373             <tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
2374 tp:docExchangeId="N06">
2375                 <tp:Characteristics tp:syncReplyMode="none"
2376 tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
2377 tp:secureTransport="true" tp:confidentiality="true" tp:authenticated="true"
2378 tp:authorized="false"/>
2379             </tp:DeliveryChannel>
2380             <tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
2381 tp:docExchangeId="N06">
2382                 <tp:Characteristics tp:syncReplyMode="none"
2383 tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
2384 tp:secureTransport="false" tp:confidentiality="true" tp:authenticated="true"
2385 tp:authorized="false"/>
2386             </tp:DeliveryChannel>
2387             <tp:Transport tp:transportId="N05">
2388                 <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
    
```

```

2389         <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
2390         <tp:Endpoint
2391 tp:uri="https://www.example.com/servlets/ebxmlhandler" tp:type="allPurpose"/>
2392         <tp:TransportSecurity>
2393             <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
2394             <tp:CertificateRef tp:certId="N03"/>
2395         </tp:TransportSecurity>
2396     </tp:Transport>
2397     <tp:Transport tp:transportId="N18">
2398         <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2399         <tp:ReceivingProtocol tp:version="1.1">SMTP</tp:ReceivingProtocol>
2400         <tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
2401 tp:type="allPurpose"/>
2402     </tp:Transport>
2403     <tp:DocExchange tp:docExchangeId="N06">
2404         <tp:ebXMLBinding tp:version="0.98b">
2405             <tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
2406 tp:idempotency="true" tp:messageOrderSemantics="Guaranteed">
2407                 <tp:Retries>5</tp:Retries>
2408                 <tp:RetryInterval>30</tp:RetryInterval>
2409                 <tp:PersistDuration>P1D</tp:PersistDuration>
2410             </tp:ReliableMessaging>
2411             <tp:NonRepudiation>
2412
2413             <tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
2414
2415             <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
2416
2417             <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
2418 sha1</tp:SignatureAlgorithm>
2419                 <tp:CertificateRef tp:certId="N03"/>
2420                 </tp:NonRepudiation>
2421                 <tp:DigitalEnvelope>
2422                     <tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
2423                     <tp:EncryptionAlgorithm>DES-
2424 CBC</tp:EncryptionAlgorithm>
2425                         <tp:CertificateRef tp:certId="N03"/>
2426                     </tp:DigitalEnvelope>
2427                 </tp:ebXMLBinding>
2428             </tp:DocExchange>
2429     </tp:PartyInfo>
2430     <tp:PartyInfo>
2431         <tp:PartyId tp:type="DUNS">987654321</tp:PartyId>
2432         <tp:PartyRef xlink:type="simple" xlink:href="http://contrived-
2433 example.com/about.html"/>
2434         <tp:CollaborationRole tp:id="N30">
2435             <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
2436 xlink:type="simple" xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
2437             <tp:Role tp:name="seller" xlink:type="simple"
2438 xlink:href="http://ebxml.org/processes/buySell.xml#seller"/>
2439             <tp:CertificateRef tp:certId="N33"/>
2440             <tp:ServiceBinding tp:channelId="N34" tp:packageId="N0402"
2441 tp:name="sellerService"/>
2442         </tp:CollaborationRole>
2443         <tp:Certificate tp:certId="N33">
2444             <ds:KeyInfo/>
2445         </tp:Certificate>
2446         <tp:DeliveryChannel tp:channelId="N34" tp:transportId="N35"
2447 tp:docExchangeId="N06">
2448             <tp:Characteristics tp:nonrepudiationOfOrigin="true"
2449 tp:nonrepudiationOfReceipt="false" tp:secureTransport="true" tp:confidentiality="true"
2450 tp:authenticated="true" tp:authorized="false"/>
2451         </tp:DeliveryChannel>
2452         <tp:Transport tp:transportId="N35">
2453             <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2454             <tp:SendingProtocol>SMTP</tp:SendingProtocol>
2455             <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
2456             <tp:Endpoint tp:uri="https://www.contrived-
2457 example.com/servlets/ebxmlhandler" tp:type="allPurpose"/>
2458             <tp:TransportSecurity>
2459                 <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
2460                 <tp:CertificateRef tp:certId="N33"/>

```

```

2461         </tp:TransportSecurity>
2462     </tp:Transport>
2463     <tp:DocExchange tp:docExchangeId="N36">
2464         <tp:ebXMLBinding tp:version="0.98b">
2465             <tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
2466 tp:idempotency="true" tp:messageOrderSemantics="Guaranteed">
2467                 <tp:Retries>5</tp:Retries>
2468                 <tp:RetryInterval>30</tp:RetryInterval>
2469                 <tp:PersistDuration>P1D</tp:PersistDuration>
2470             </tp:ReliableMessaging>
2471             <tp:NonRepudiation>
2472
2473         <tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
2474
2475         <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
2476
2477         <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
2478 sha1</tp:SignatureAlgorithm>
2479             <tp:CertificateRef tp:certId="N33"/>
2480             </tp:NonRepudiation>
2481             <tp:DigitalEnvelope>
2482                 <tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
2483                 <tp:EncryptionAlgorithm>DES-
2484 CBC</tp:EncryptionAlgorithm>
2485                 <tp:CertificateRef tp:certId="N33"/>
2486                 </tp:DigitalEnvelope>
2487             </tp:ebXMLBinding>
2488         </tp:DocExchange>
2489     </tp:PartyInfo>
2490     <tp:Packaging tp:id="N0402">
2491         <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>
2492         <tp:SimplePart tp:id="N40" tp:mimetype="text/xml">
2493             <tp:NamespaceSupported
2494 tp:location="http://ebxml.org/project_teams/transport/messageService.xsd"
2495 tp:version="0.98b">http://www.ebxml.org/namespaces/messageService</tp:NamespaceSupport
2496 ed>
2497             <tp:NamespaceSupported
2498 tp:location="http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"
2499 tp:version="1.0">http://www.w3.org/2000/09/xmldsig</tp:NamespaceSupported>
2500             </tp:SimplePart>
2501             <tp:CompositeList>
2502                 <tp:Composite tp:id="N033" tp:mimetype="multipart/related"
2503 tp:mimeparameters="type=text/xml;">
2504                     <tp:Constituent tp:idref="N40"/>
2505                     <tp:Constituent tp:idref="N41"/>
2506                 </tp:Composite>
2507                 <tp:Encapsulation tp:id="N41" tp:mimetype="text/xml"
2508 tp:mimeparameters="charset=UTF-8;">
2509                     <tp:Constituent tp:idref="N40"/>
2510                 </tp:Encapsulation>
2511             </tp:CompositeList>
2512         </tp:Packaging>
2513         <tp:Comment xml:lang="en-us">buy/sell agreement between example.com and
2514 contrived-example.com</tp:Comment>
2515     </tp:CollaborationProtocolAgreement>

```


2516 **Appendix C DTD Corresponding to Complete CPP/CPA**
 2517 **Definition (Normative)**

2518 This DTD is available as an ASCII file at:

2519 http://ebxml.org/project_teams/trade_partner/cpp-cpa-095.dtd

```

2520
2521 <?xml version='1.0' encoding='UTF-8' ?>
2522
2523 <!--Generated by XML Authority-->
2524
2525 <!ELEMENT CollaborationProtocolAgreement (Status , Start , End ,
2526 ConversationConstraints? , PartyInfo+ , Packaging , ds:Signature* , Comment*)>
2527
2528 <!ATTLIST CollaborationProtocolAgreement cpaid CDATA #IMPLIED
2529 version CDATA #IMPLIED >
2530 <!ELEMENT CollaborationProtocolProfile (PartyInfo+ , Packaging , ds:Signature? ,
2531 Comment*)>
2532
2533 <!ATTLIST CollaborationProtocolProfile version CDATA #IMPLIED >
2534 <!ELEMENT ProcessSpecification (ds:Reference?)>
2535
2536 <!ATTLIST ProcessSpecification version CDATA #REQUIRED
2537 name CDATA #REQUIRED
2538 xlink:type CDATA #FIXED 'simple'
2539 xlink:href CDATA #IMPLIED >
2540 <!ELEMENT Protocol (#PCDATA)>
2541
2542 <!ATTLIST Protocol version CDATA #IMPLIED >
2543 <!ELEMENT SendingProtocol (#PCDATA)>
2544
2545 <!ATTLIST SendingProtocol version CDATA #IMPLIED >
2546 <!ELEMENT ReceivingProtocol (#PCDATA)>
2547
2548 <!ATTLIST ReceivingProtocol version CDATA #IMPLIED >
2549 <!ELEMENT CollaborationRole (ProcessSpecification , Role , CertificateRef? ,
2550 ServiceBinding+)>
2551
2552 <!ATTLIST CollaborationRole id ID #IMPLIED >
2553 <!ELEMENT PartyInfo (PartyId+ , PartyRef , CollaborationRole+ , Certificate+ ,
2554 DeliveryChannel+ , Transport+ , DocExchange+)>
2555
2556 <!ELEMENT PartyId (#PCDATA)>
2557
2558 <!ATTLIST PartyId type CDATA #IMPLIED >
2559 <!ELEMENT PartyRef EMPTY>
2560
2561 <!ATTLIST PartyRef xlink:type (simple ) #IMPLIED
2562 xlink:href CDATA #IMPLIED >
2563 <!ELEMENT DeliveryChannel (Characteristics)>
2564
2565 <!ATTLIST DeliveryChannel channelId ID #REQUIRED
2566 transportId IDREF #REQUIRED
2567 docExchangeId IDREF #REQUIRED >
2568 <!ELEMENT Transport (SendingProtocol+ , ReceivingProtocol , Endpoint+ ,
2569 TransportSecurity?)>
2570
2571 <!ATTLIST Transport transportId ID #REQUIRED >
2572 <!ELEMENT Endpoint EMPTY>
2573
2574 <!ATTLIST Endpoint uri CDATA #REQUIRED
2575 type (login | request | response | error | allPurpose )
2576 'allPurpose' >
2577 <!ELEMENT Retries (#PCDATA)>
2578
2579 <!ELEMENT RetryInterval (#PCDATA)>
2580
2581 <!ELEMENT TransportSecurity (Protocol , CertificateRef?)>
    
```

```

2582
2583 <!ELEMENT Certificate (ds:KeyInfo)>
2584
2585 <!ATTLIST Certificate certId ID #REQUIRED >
2586 <!ELEMENT DocExchange (ebXMLBinding)>
2587
2588 <!ATTLIST DocExchange docExchangeId ID #REQUIRED >
2589 <!ELEMENT PersistDuration (#PCDATA)>
2590
2591 <!ATTLIST PersistDuration e-dtype NMOKEN #FIXED 'timeDuration' >
2592 <!ELEMENT ReliableMessaging (Retries , RetryInterval , PersistDuration)?>
2593
2594 <!ATTLIST ReliableMessaging
2595 deliverySemantics (OnceAndOnlyOnce | BestEffort ) #REQUIRED
2596 messageOrderSemantics (Guaranteed | NotGuaranteed ) "NotGuaranteed"
2597 idempotency CDATA #REQUIRED >
2598 <!ELEMENT NonRepudiation (Protocol , HashFunction , SignatureAlgorithm ,
2599 CertificateRef)>
2600
2601 <!ELEMENT HashFunction (#PCDATA)>
2602
2603 <!ELEMENT EncryptionAlgorithm (#PCDATA)>
2604
2605 <!ELEMENT SignatureAlgorithm (#PCDATA)>
2606
2607 <!ELEMENT DigitalEnvelope (Protocol , EncryptionAlgorithm , CertificateRef)>
2608
2609 <!ELEMENT CertificateRef EMPTY>
2610
2611 <!ATTLIST CertificateRef certId IDREF #REQUIRED >
2612 <!ELEMENT ebXMLBinding (ReliableMessaging? , NonRepudiation? , DigitalEnvelope? ,
2613 NamespaceSupported*)>
2614
2615 <!ATTLIST ebXMLBinding version CDATA #REQUIRED >
2616 <!ELEMENT NamespaceSupported (#PCDATA)>
2617
2618 <!ATTLIST NamespaceSupported location CDATA #REQUIRED
2619 version CDATA #IMPLIED >
2620 <!ELEMENT Characteristics EMPTY>
2621
2622 <!ATTLIST Characteristics syncReplyMode (responseOnly |
2623 signalsAndResponse |
2624 signalsOnly |
2625 none ) #IMPLIED
2626 nonrepudiationOfOrigin CDATA #IMPLIED
2627 nonrepudiationOfReceipt CDATA #IMPLIED
2628 secureTransport CDATA #IMPLIED
2629 confidentiality CDATA #IMPLIED
2630 authenticated CDATA #IMPLIED
2631 authorized CDATA #IMPLIED >
2632 <!ELEMENT ServiceBinding (Override*)>
2633
2634 <!ATTLIST ServiceBinding channelId IDREF #REQUIRED
2635 packageId IDREF #REQUIRED
2636 name CDATA #REQUIRED >
2637 <!ELEMENT Status EMPTY>
2638
2639 <!ATTLIST Status value (agreed | signed | proposed ) #REQUIRED >
2640 <!ELEMENT Start (#PCDATA)>
2641
2642 <!ELEMENT End (#PCDATA)>
2643
2644 <!ELEMENT Type (#PCDATA)>
2645
2646 <!ELEMENT ConversationConstraints EMPTY>
2647
2648 <!ATTLIST ConversationConstraints invocationLimit CDATA #IMPLIED
2649 concurrentConversations CDATA #IMPLIED >
2650 <!ELEMENT Override EMPTY>
2651
2652 <!ATTLIST Override action CDATA #REQUIRED
2653 channelId ID #REQUIRED

```

```

2654         packageId IDREF #REQUIRED
2655         xlink:href CDATA #IMPLIED
2656         xlink:type CDATA #FIXED 'simple' >
2657 <!ELEMENT Role EMPTY>
2658
2659 <!ATTLIST Role name CDATA #REQUIRED
2660         xlink:type CDATA #FIXED 'simple'
2661         xlink:href CDATA #IMPLIED >
2662 <!ELEMENT Constituent EMPTY>
2663
2664 <!ATTLIST Constituent idref CDATA #REQUIRED >
2665 <!ELEMENT ProcessingCapabilities EMPTY>
2666
2667 <!ATTLIST ProcessingCapabilities parse CDATA #REQUIRED
2668         generate CDATA #REQUIRED >
2669 <!ELEMENT SimplePart (NamespaceSupported*)>
2670
2671 <!ATTLIST SimplePart id ID #IMPLIED
2672         mimetype CDATA #REQUIRED >
2673 <!ELEMENT Encapsulation (Constituent)>
2674
2675 <!ATTLIST Encapsulation id ID #IMPLIED
2676         mimetype CDATA #REQUIRED
2677         mimeparameters CDATA #IMPLIED >
2678 <!ELEMENT Composite (Constituent+)>
2679
2680 <!ATTLIST Composite id ID #IMPLIED
2681         mimetype CDATA #REQUIRED
2682         mimeparameters CDATA #IMPLIED >
2683 <!ELEMENT CompositeList (Encapsulation | Composite)+>
2684
2685 <!ELEMENT Packaging (ProcessingCapabilities , SimplePart+ , CompositeList?)>
2686
2687 <!ATTLIST Packaging id ID #REQUIRED >
2688 <!ELEMENT Comment (#PCDATA)>
2689
2690 <!ATTLIST Comment xml:lang CDATA #REQUIRED >
2691 <!ELEMENT ds:Signature ANY>
2692
2693 <!ELEMENT ds:Reference ANY>
2694
2695 <!ELEMENT ds:KeyInfo ANY>
2696

```

2697 **Appendix D XML Schema Document Corresponding to**
 2698 **Complete CPP and CPA Definition (Normative)**

2699 This XML Schema document is available as an ASCII file at:

2700 http://ebxml.org/project_teams/trade_partner/cpp-cpa-095.xsd

```

2701
2702 <?xml version="1.0" encoding="UTF-8"?>
2703 <schema targetNamespace="http://www.ebxml.org/namespaces/tradePartner"
2704   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2705   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2706   xmlns:xlink="http://www.w3.org/1999/xlink"
2707   xmlns:tns="http://www.ebxml.org/namespaces/tradePartner"
2708   xmlns="http://www.w3.org/2000/10/XMLSchema"
2709   elementFormDefault="qualified"
2710   attributeFormDefault="unqualified"
2711   version="1.0">
2712   <import namespace="http://www.w3.org/1999/xlink"
2713     schemaLocation="http://ebxml.org/project_teams/transport/xlink.xsd"/>
2714   <import namespace="http://www.w3.org/2000/09/xmldsig#"
2715     schemaLocation="http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"/>
2716   <import namespace="http://www.w3.org/XML/1998/namespace"
2717     schemaLocation="http://ebxml.org/project_teams/transport/xml_lang.xsd"/>
2718   <attributeGroup name="pkg.grp">
2719     <attribute ref="tns:id"/>
2720     <attribute name="mimetype" type="tns:non-empty-string" use="required"/>
2721     <attribute name="mimeparameters" type="tns:non-empty-string"/>
2722   </attributeGroup>
2723   <attributeGroup name="xlink.grp">
2724     <attribute ref="xlink:type"/>
2725     <attribute ref="xlink:href"/>
2726   </attributeGroup>
2727   <element name="CollaborationProtocolAgreement">
2728     <complexType>
2729       <sequence>
2730         <element ref="tns:Status"/>
2731         <element ref="tns:Start"/>
2732         <element ref="tns:End"/>
2733         <element ref="tns:ConversationConstraints" minOccurs="0"/>
2734         <element ref="tns:PartyInfo" maxOccurs="unbounded"/>
2735         <element ref="tns:Packaging"/>
2736         <element ref="ds:Signature" minOccurs="0"
2737         maxOccurs="unbounded"/>
2738         <element ref="tns:Comment" minOccurs="0"
2739         maxOccurs="unbounded"/>
2740       </sequence>
2741       <attribute name="cpaid" type="tns:non-empty-string"/>
2742       <attribute ref="tns:version"/>
2743       <anyAttribute namespace="##targetNamespace
2744         http://www.w3.org/2000/10/XMLSchema-instance" processContents="lax"/>
2745     </complexType>
2746   </element>
2747   <element name="CollaborationProtocolProfile">
2748     <complexType>
2749       <sequence>
2750         <element ref="tns:PartyInfo" maxOccurs="unbounded"/>
2751         <element ref="tns:Packaging"/>
2752         <element ref="ds:Signature" minOccurs="0"/>
2753         <element ref="tns:Comment" minOccurs="0"
2754         maxOccurs="unbounded"/>
2755       </sequence>
2756       <attribute ref="tns:version"/>
2757       <anyAttribute namespace="##targetNamespace
2758         http://www.w3.org/2000/10/XMLSchema-instance" processContents="lax"/>
2759     </complexType>
2760   </element>
2761   <element name="ProcessSpecification">
2762     <complexType>

```

```

2763         <sequence>
2764             <element ref="ds:Reference" minOccurs="0"/>
2765         </sequence>
2766         <attribute ref="tns:version"/>
2767         <attribute name="name" type="tns:non-empty-string"
2768 use="required"/>
2769         <attributeGroup ref="tns:xlink.grp"/>
2770     </complexType>
2771 </element>
2772 <element name="Protocol" type="tns:protocol.type"/>
2773 <element name="SendingProtocol" type="tns:protocol.type"/>
2774 <element name="ReceivingProtocol" type="tns:protocol.type"/>
2775 <element name="CollaborationRole">
2776     <complexType>
2777         <sequence>
2778             <element ref="tns:ProcessSpecification"/>
2779             <element ref="tns:Role"/>
2780             <element ref="tns:CertificateRef" minOccurs="0"/>
2781             <element ref="tns:ServiceBinding" maxOccurs="unbounded"/>
2782         </sequence>
2783         <attribute ref="tns:id"/>
2784     </complexType>
2785 </element>
2786 <element name="PartyInfo">
2787     <complexType>
2788         <sequence>
2789             <element ref="tns:PartyId" maxOccurs="unbounded"/>
2790             <element ref="tns:PartyRef"/>
2791             <element ref="tns:CollaborationRole" maxOccurs="unbounded"/>
2792             <element ref="tns:Certificate" maxOccurs="unbounded"/>
2793             <element ref="tns:DeliveryChannel" maxOccurs="unbounded"/>
2794             <element ref="tns:Transport" maxOccurs="unbounded"/>
2795             <element ref="tns:DocExchange" maxOccurs="unbounded"/>
2796         </sequence>
2797     </complexType>
2798 </element>
2799 <element name="PartyId">
2800     <complexType>
2801         <simpleContent>
2802             <extension base="tns:non-empty-string">
2803                 <attribute name="type" type="tns:non-empty-string"/>
2804             </extension>
2805         </simpleContent>
2806     </complexType>
2807 </element>
2808 <element name="PartyRef">
2809     <complexType>
2810         <attributeGroup ref="tns:xlink.grp"/>
2811     </complexType>
2812 </element>
2813 <element name="DeliveryChannel">
2814     <complexType>
2815         <sequence>
2816             <element ref="tns:Characteristics"/>
2817         </sequence>
2818         <attribute name="channelId" type="ID" use="required"/>
2819         <attribute name="transportId" type="IDREF" use="required"/>
2820         <attribute name="docExchangeId" type="IDREF" use="required"/>
2821     </complexType>
2822 </element>
2823 <element name="Transport">
2824     <complexType>
2825         <sequence>
2826             <element ref="tns:SendingProtocol" maxOccurs="unbounded"/>
2827             <element ref="tns:ReceivingProtocol"/>
2828             <element ref="tns:Endpoint" maxOccurs="unbounded"/>
2829             <element ref="tns:TransportSecurity" minOccurs="0"/>
2830         </sequence>
2831         <attribute name="transportId" type="ID" use="required"/>
2832     </complexType>
2833 </element>
2834 <element name="Endpoint">

```

```

2835         <complexType>
2836             <attribute name="uri" type="uriReference" use="required"/>
2837             <attribute name="type" type="tns:endpointType.type" use="default"
value="allPurpose"/>
2838         </complexType>
2839     </element>
2840     <element name="Retries" type="string"/>
2841     <element name="RetryInterval" type="string"/>
2842     <element name="TransportSecurity">
2843         <complexType>
2844             <sequence>
2845                 <element ref="tns:Protocol"/>
2846                 <element ref="tns:CertificateRef" minOccurs="0"/>
2847             </sequence>
2848         </complexType>
2849     </element>
2850     <element name="Certificate">
2851         <complexType>
2852             <sequence>
2853                 <element ref="ds:KeyInfo"/>
2854             </sequence>
2855             <attribute name="certId" type="ID" use="required"/>
2856         </complexType>
2857     </element>
2858     <element name="DocExchange">
2859         <complexType>
2860             <sequence>
2861                 <element ref="tns:ebXMLBinding"/>
2862             </sequence>
2863             <attribute name="docExchangeId" type="ID" use="required"/>
2864         </complexType>
2865     </element>
2866     <element name="ReliableMessaging">
2867         <complexType>
2868             <sequence minOccurs="0">
2869                 <element ref="tns:Retries"/>
2870                 <element ref="tns:RetryInterval"/>
2871                 <element name="PersistDuration" type="timeDuration"/>
2872             </sequence>
2873             <attribute name="deliverySemantics" type="tns:ds.type"
use="required"/>
2874             <attribute name="idempotency" type="boolean" use="required"/>
2875             <attribute name="messageOrderSemantics" type="tns:mos.type"
use="optional" value="NotGuaranteed"/>
2876         </complexType>
2877         <!-- <element name="PersistDuration" type="duration"/> -->
2878     </element>
2879     <element name="NonRepudiation">
2880         <complexType>
2881             <sequence>
2882                 <element ref="tns:Protocol"/>
2883                 <element ref="tns:HashFunction"/>
2884                 <element ref="tns:SignatureAlgorithm"/>
2885                 <element ref="tns:CertificateRef"/>
2886             </sequence>
2887         </complexType>
2888     </element>
2889     <element name="HashFunction" type="string"/>
2890     <element name="EncryptionAlgorithm" type="string"/>
2891     <element name="SignatureAlgorithm" type="string"/>
2892     <element name="DigitalEnvelope">
2893         <complexType>
2894             <sequence>
2895                 <element ref="tns:Protocol"/>
2896                 <element ref="tns:EncryptionAlgorithm"/>
2897                 <element ref="tns:CertificateRef"/>
2898             </sequence>
2899         </complexType>
2900     </element>
2901     <element name="CertificateRef">
2902         <complexType>
2903             <attribute name="certId" type="IDREF" use="required"/>

```

```

2907         </complexType>
2908     </element>
2909     <element name="ebXMLBinding">
2910         <complexType>
2911             <sequence>
2912                 <element ref="tns:ReliableMessaging" minOccurs="0"/>
2913                 <element ref="tns:NonRepudiation" minOccurs="0"/>
2914                 <element ref="tns:DigitalEnvelope" minOccurs="0"/>
2915                 <element ref="tns:NamespaceSupported" minOccurs="0"
2916 maxOccurs="unbounded"/>
2917             </sequence>
2918             <attribute ref="tns:version"/>
2919         </complexType>
2920     </element>
2921     <element name="NamespaceSupported">
2922         <complexType>
2923             <simpleContent>
2924                 <extension base="uriReference">
2925                     <attribute name="location" type="uriReference"
2926 use="required"/>
2927                     <attribute ref="tns:version"/>
2928                 </extension>
2929             </simpleContent>
2930         </complexType>
2931     </element>
2932     <element name="Characteristics">
2933         <complexType>
2934             <attribute ref="tns:syncReplyMode"/>
2935             <attribute name="nonrepudiationOfOrigin" type="boolean"/>
2936             <attribute name="nonrepudiationOfReceipt" type="boolean"/>
2937             <attribute name="secureTransport" type="boolean"/>
2938             <attribute name="confidentiality" type="boolean"/>
2939             <attribute name="authenticated" type="boolean"/>
2940             <attribute name="authorized" type="boolean"/>
2941         </complexType>
2942     </element>
2943     <element name="ServiceBinding">
2944         <complexType>
2945             <sequence>
2946                 <element ref="tns:Override" minOccurs="0"
2947 maxOccurs="unbounded"/>
2948             </sequence>
2949             <attribute name="channelId" type="IDREF" use="required"/>
2950             <attribute name="packageId" type="IDREF" use="required"/>
2951             <attribute name="name" type="tns:non-empty-string"
2952 use="required"/>
2953         </complexType>
2954         <unique name="action.const">
2955             <selector xpath="./Override"/>
2956             <field xpath="@action"/>
2957         </unique>
2958     </element>
2959     <element name="Status">
2960         <complexType>
2961             <attribute name="value" type="tns:statusValue.type"
2962 use="required"/>
2963         </complexType>
2964     </element>
2965     <element name="Start" type="timeInstant"/>
2966     <element name="End" type="timeInstant"/>
2967     <!--
2968     <element name="Start" type="dateTime"/>
2969     <element name="End" type="dateTime"/>
2970     -->
2971     <element name="Type" type="string"/>
2972     <element name="ConversationConstraints">
2973         <complexType>
2974             <attribute name="invocationLimit" type="int"/>
2975             <attribute name="concurrentConversations" type="int"/>
2976         </complexType>
2977     </element>
2978     <element name="Override">

```

```

2979         <complexType>
2980             <attribute name="action" type="tns:non-empty-string"
2981 use="required" />
2982             <attribute name="channelId" type="ID" use="required" />
2983             <attribute name="packageId" type="IDREF" use="required" />
2984             <attributeGroup ref="tns:xlink.grp" />
2985         </complexType>
2986     </element>
2987     <element name="Role">
2988         <complexType>
2989             <attribute name="name" type="tns:non-empty-string"
2990 use="required" />
2991             <attributeGroup ref="tns:xlink.grp" />
2992         </complexType>
2993     </element>
2994     <element name="Constituent">
2995         <complexType>
2996             <attribute ref="tns:idref" />
2997         </complexType>
2998     </element>
2999     <element name="Packaging">
3000         <complexType>
3001             <sequence>
3002                 <element name="ProcessingCapabilities">
3003                     <complexType>
3004                         <attribute name="parse" type="boolean"
3005 use="required" />
3006                         <attribute name="generate" type="boolean"
3007 use="required" />
3008                     </complexType>
3009                 </element>
3010                 <element name="SimplePart" maxOccurs="unbounded">
3011                     <complexType>
3012                         <sequence>
3013                             <element ref="tns:NamespaceSupported"
3014 minOccurs="0" maxOccurs="unbounded" />
3015                         </sequence>
3016                         <attributeGroup ref="tns:pkg.grp" />
3017                     </complexType>
3018                 </element>
3019                 <element name="CompositeList" minOccurs="0">
3020                     <complexType>
3021                         <choice maxOccurs="unbounded">
3022                             <element name="Encapsulation">
3023                                 <complexType>
3024                                     <sequence>
3025                                         <element
3026 ref="tns:Constituent" />
3027                                     </sequence>
3028                                 </complexType>
3029                             </element>
3030                         </choice>
3031                     </complexType>
3032                 </element>
3033                 <element name="Composite">
3034                     <complexType>
3035                         <sequence>
3036                             <element
3037 ref="tns:Constituent" maxOccurs="unbounded" />
3038                         </sequence>
3039                     </complexType>
3040                 </element>
3041             </sequence>
3042             <attributeGroup ref="tns:pkg.grp" />
3043         </complexType>
3044     </element>
3045     <sequence>
3046         <attribute ref="tns:id" />
3047     </sequence>
3048 </element>
3049 <element name="Comment">
3050 <complexType>

```



```
3051         <simpleContent>
3052             <extension base="tns:non-empty-string">
3053                 <attribute ref="xml:lang" />
3054             </extension>
3055         </simpleContent>
3056     </complexType>
3057 </element>
3058 <!-- COMMON -->
3059 <simpleType name="ds.type">
3060     <restriction base="NMTOKEN">
3061         <enumeration value="OnceAndOnlyOnce" />
3062         <enumeration value="BestEffort" />
3063     </restriction>
3064 </simpleType>
3065 <simpleType name="mos.type">
3066     <restriction base="NMTOKEN">
3067         <enumeration value="Guaranteed" />
3068         <enumeration value="NotGuaranteed" />
3069     </restriction>
3070 </simpleType>
3071 <simpleType name="statusValue.type">
3072     <restriction base="NMTOKEN">
3073         <enumeration value="agreed" />
3074         <enumeration value="signed" />
3075         <enumeration value="proposed" />
3076     </restriction>
3077 </simpleType>
3078 <simpleType name="endpointType.type">
3079     <restriction base="NMTOKEN">
3080         <enumeration value="login" />
3081         <enumeration value="request" />
3082         <enumeration value="response" />
3083         <enumeration value="error" />
3084         <enumeration value="allPurpose" />
3085     </restriction>
3086 </simpleType>
3087 <simpleType name="non-empty-string">
3088     <restriction base="string">
3089         <minLength value="1" />
3090     </restriction>
3091 </simpleType>
3092 <simpleType name="syncReplyMode.type">
3093     <restriction base="NMTOKEN">
3094         <enumeration value="responseOnly" />
3095         <enumeration value="signalsAndResponse" />
3096         <enumeration value="signalsOnly" />
3097         <enumeration value="none" />
3098     </restriction>
3099 </simpleType>
3100 <complexType name="protocol.type">
3101     <simpleContent>
3102         <extension base="tns:non-empty-string">
3103             <attribute ref="tns:version" />
3104         </extension>
3105     </simpleContent>
3106 </complexType>
3107 <attribute name="idref" type="IDREF" form="unqualified" />
3108 <attribute name="id" type="ID" form="unqualified" />
3109 <attribute name="version" type="tns:non-empty-string" />
3110 <attribute name="syncReplyMode" type="tns:syncReplyMode.type" />
3111 </schema>
```

3112 **Appendix E** Formats of Information in the CPP and CPA
3113 (Normative)

3114 This section defines format information that is not defined by the [XML] specification and is not
3115 defined in the descriptions of specific elements.

3116
3117 Formats of Character Strings

3118
3119 **Protocol and Version Elements**

3120
3121 Values of *Protocol*, *Version*, and similar elements are flexible. In general, any protocol and
3122 version for which the support software is available to both *Parties* to a *CPA* MAY be selected as
3123 long as the choice does not require changes to the DTD or schema and therefore a change to this
3124 specification.

3125
3126 NOTE: A possible implementation MAY be based on the use of plug-ins or exits to
3127 support the values of these elements.

3128
3129 **Alphanumeric Strings**

3130
3131 Alphanumeric strings not further defined in this section follow these rules unless otherwise
3132 stated in the description of an individual element:

- 3133
- 3134 • Values of elements are case insensitive unless otherwise stated.
 - 3135 • Strings which represent file or directory names are case sensitive to ensure that they are
3136 acceptable to both UNIX and Windows systems.

3137
3138 **Numeric Strings**

3139
3140 A numeric string is a signed or unsigned decimal integer in the range imposed by a 32-bit binary
3141 number, i.e. -2,147,483,648 to +2,417,483,647. Negative numbers MAY or MAY not be
3142 permitted in particular elements.

3143 **Appendix F** Composing a CPA from Two CPPs (Non- 3144 Normative)

3145 3146 Overview and Limitations

3147
3148 In this appendix, we discuss the tasks involved in *CPA* formation from *CPPs*. The detailed
3149 procedures for *CPA* formation are currently left for implementers. Therefore, no normative
3150 specification is provided for algorithms for *CPA* formation. In this initial section, we provide
3151 some background on *CPA* formation tasks.

3152
3153 There are three basic reasons why we prefer to provide information about the component tasks
3154 involved in *CPA* formation rather than attempt to provide an algorithm for *CPA* formation:

- 3155 1. The precise informational inputs to the *CPA* formation procedure vary.
- 3156 2. There exist at least two distinct approaches to *CPA* formation. One useful approach for
3157 certain situations involves basing *CPA* formation from a *CPA* template; the other approach
3158 involves composition from *CPPs*.
- 3159 3. The conditions for output of a given *CPA* given two *CPPs* can involve different levels and
3160 extents of interoperability. In other words, when an optimal solution that satisfies every level
3161 of requirement and every other additional constraint does not exist, a *Party* MAY propose a
3162 *CPA* that satisfies enough of the requirements for “a good enough” implementation. User
3163 input MAY be solicited to determine what is a good enough implementation, and so MAY
3164 be as varied as there are user configuration options to express preferences. In practice,
3165 compromises MAY be made on security, reliable messaging, levels of signals and
3166 acknowledgements, and other matters in order to find some acceptable means of doing
3167 *Business*.

3168
3169 Each of these reasons is elaborated in greater detail in the following sections.

3170 3171 3172 Variability in Inputs

3173
3174 User preferences provide one source of variability in the inputs to the *CPA* formation process.
3175 Let us suppose in this section that each of the *Parties* has made its *CPP* available to potential
3176 collaborators. Normally one *Party* will have a desired *Business Collaboration* (defined in a
3177 *Process-Specification* document) to implement with its intended collaborator. So the information
3178 inputs will normally involve a user preference about intended *Business Collaboration* in addition
3179 to just the *CPPs*.

3180
3181 A *CPA* formation tool MAY have access to local user information not advertised in the *CPP* that
3182 MAY contribute to the *CPA* that is formed. A user MAY have chosen to only advertise those
3183 system capabilities that reflect nondeprecated capabilities. For example, a user MAY only
3184 advertise HTTP and omit FTP, even when capable of using FTP. The reason for omitting FTP
3185 might be concerns about the scalability of managing user accounts, directories, and passwords
3186

3187 for FTP sessions. Despite not advertising an FTP capability, configuration software MAY use
3188 tacit knowledge about its own FTP capability to form a *CPA* with an intended collaborator who
3189 happens to have only an FTP capability for implementing a desired *Business Collaboration*. In
3190 other words, *Business* interests MAY, in this case, override the deprecation policy. Both tacit
3191 knowledge and detailed preference information account for variability in inputs into the *CPA*
3192 formation process.

3193
3194

3195 Different Approaches

3196
3197 When a *CPA* is formed from a *CPA* template, it is typically because the capabilities of one of the
3198 *Parties* are limited, and already tacitly known. For example, if a *CPA* template were implicitly
3199 presented to a Web browser for use in an implementation using browser based forms capabilities,
3200 then the template maker can assume that the other *Party* has suitable web capabilities (or is about
3201 to download them). Therefore, all that really needs to be done is to supply ***PartyRef***, ***Certificate***,
3202 and similar items for substitution into a *CPA* template. The *CPA* template will already have all
3203 the capabilities of both *Parties* specified at the various levels, and will have placeholders for
3204 values to be supplied by one of the *Partners*. A simple form might be adequate to gather the
3205 needed information and produce a *CPA*.

3206
3207

3208 Variable Output "Satisficing" Policies

3209
3210 A *CPA* can support a fully interoperable configuration in which agreement has been reached on
3211 all technical levels needed for *Business Collaboration*. In such a case, matches in capabilities
3212 will have been found in all relevant technical levels.

3213
3214 However, there can be interoperable configurations agreed to in a *CPA* in which not all aspects
3215 of a *Business Collaboration* match. Gaps MAY exist in packaging, security, signaling, reliable
3216 messaging and other areas and yet the systems can still transport the *Business* data, and special
3217 means can be employed to handle the exceptions. In such situations, a *CPA* MAY reflect
3218 configured policies or expressly solicited user permission to ignore some shortcomings in
3219 configurations. A system might not be capable of responding in a *Business Collaboration* so as
3220 to support a recommended ability to supply nonrepudiation of receipt, but might still be
3221 acceptable for *Business* reasons. A system might not be able to handle all the processing required
3222 to support "multipart/related" processing with a type value of "application/vnd.eb+xml," and yet
3223 still be able to treat the multipart according to "multipart/mixed" handling and allow *Business*
3224 *Collaboration* to take place. In fact, short of a failure to be able to transport data and a failure to
3225 be able to provide data relevant to the *Business Collaboration*, there are few features that might
3226 not be temporarily or indefinitely compromised about, given overriding *Business* interests. This
3227 situation of "partial interoperability" is to be expected to persist for some time, and so interferes
3228 with formulating a "clean" algorithm for deciding on what is sufficient for interoperability.

3229
3230 In summary, the previous considerations indicate that at the present it is at best premature to seek
3231 a simple algorithm for *CPA* formation from *CPPs*. It is to be expected that as capability
3232 characterization and exchange becomes a more refined subject, that advances will be made in
3233 characterizing *CPA* formation and negotiation.

3234
3235 Despite it being too soon to propose a simple algorithm for *CPA* formation that covers all the
3236 above variations, it is currently possible to enumerate the basic tasks involved in matching
3237 capabilities within *CPPs*. This information might assist the software implementer in designing a
3238 partially automated and partially interactive software system useful for configuring *Business*
3239 *Collaboration* so as to arrive at satisfactorily complete levels of interoperability. To understand
3240 the context for characterizing the constituent tasks, the general perspective on *CPPs* and *CPAs*
3241 needs to be briefly recalled.

3242
3243

3244 CPA Formation Component Tasks

3245

3246 Technically viewed, a *CPA* provides "bindings" between *Business-Collaboration* specifications
3247 (as defined in the *Process-Specification* document) and those services and protocols that are used
3248 to implement these specifications. The implementation takes place at several levels and involves
3249 varied services at these levels. A *CPA* that arrives at a fully interoperable binding of a *Business*
3250 *Collaboration* to its implementing services and protocols can be thought of as arriving at
3251 interoperable, application-to-application integration. *CPAs* MAY fall short of this goal and still
3252 be useful and acceptable to the collaborating *Parties*. Certainly, if no matching data-transport
3253 capabilities can be discovered, a *CPA* would not provide much in the way of interoperable
3254 *Business-to-Business* integration. Likewise, partial *CPAs* will leave significant system work to be
3255 done before a completely satisfactory application-to-application integration is realized. Even so,
3256 partial integration MAY be sufficient to allow collaboration, and to enjoy payoffs from increased
3257 levels of automation.

3258

3259 In practice, the *CPA* formation process MAY produce a complete *CPA*, a failure result, a gap list
3260 that drives a dialog with the user, or perhaps even a *CPA* that implements partial interoperability
3261 "good enough" for the *Business* collaborators. Because both matching capabilities and
3262 interoperability can be matters of degree, the constituent tasks are finding the matches in
3263 capabilities at different levels and for different services. We next proceed to characterize many
3264 of these constituent tasks.

3265

3266

3267 CPA Formation from *CPPs*: Enumeration of Tasks

3268

3269 To simplify discussion, assume in the following that we are viewing the tasks faced by a
3270 software agent when:

- 3271 1. an intended collaborator is known and the collaborator's *CPP* has been retrieved,
- 3272 2. the *Business Collaboration* between us and our intended collaborator has been selected,
- 3273 3. the specific role that our software agent is to play in the *Business Collaboration* is
3274 known, and
- 3275 4. the capabilities that are to be advertised in our *CPP* are known.

3276

3277 For vividness, we will suppose that our example agent wishes to play the role of supplier and
3278 seeks to find one of its current customers to begin a Purchase Order *Business Collaboration* in
3279 which the intended player plays a complementary role. For simplicity, we assume that the

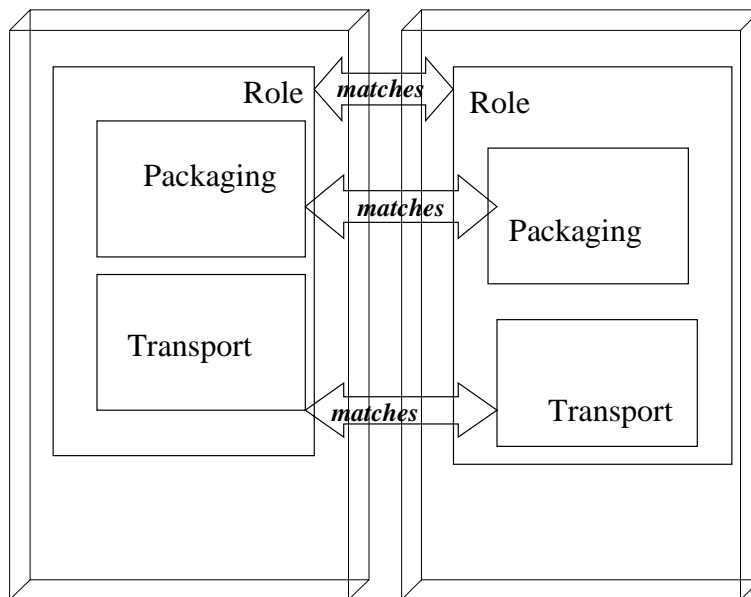
3280 information about capabilities is restricted to what is available in our agent's *CPP* and in the
 3281 *CPP* of its intended collaborator.

3282
 3283 In general, the constituent tasks consist of finding "matches" between our capabilities and our
 3284 intended collaborator's at the various levels of the protocol stacks and with respect to the
 3285 services supplied at these various levels.

3286
 3287 Figure 6 illustrates the basic tasks informing a *CPA* from two *CPPs*: matching roles, matching
 3288 packaging, and matching transport.

3289

Figure 6: Basic Tasks in Forming a CPA



3290
 3291 The first task to be considered is certainly the most basic: finding that our intended collaborator
 3292 and ourselves have complementary role capabilities.

3293
 3294

3295 **Matching Roles**

3296
 3297 Our agent has its role already selected in the *Business Collaboration*. So it now begins to check
 3298 the *Role* elements in its collaborator's *CPP*. The first element to examine is the *PartyInfo*
 3299 element that contains a subtree of elements called *CollaborationRole*. This set is searched to
 3300 discover a role that complements the role of our agent within the *Business Collaboration* that we
 3301 have chosen. For simple binary collaboration cases, it is typically sufficient to find that our
 3302 intended collaborator's *CollaborationRole* set contains *ProcessSpecification* elements that we

3303 intend to implement and where the role is not identical to our role. For more general
 3304 collaborations, we would need to know the list of roles available within the process, and keep
 3305 track that for each of the collaborators, the roles chosen instantiate those that have been specified
 3306 within the *Process-Specification* document. Collaborations involving more than two roles are not
 3307 discussed further.

3308
 3309

3310 Matching Transport

3311
 3312 We now have available a list of candidate *CollaborationRole* elements with the desired
 3313 *ProcessSpecification* element (Purchase Ordering) and where our intended collaborator plays the
 3314 buyer role. For simplicity, we shall suppose just one *CollaborationRole* element meets these
 3315 conditions within each of the relevant *CPPs* and not discuss iterating over lists. (Within these
 3316 remarks, where repetition is possible, we will frame the discussion by assuming that just one
 3317 element is present.)

3318
 3319 Matching transport first means matching the *SendingProtocol* capabilities of our intended
 3320 collaborator with the *ReceivingProtocol* capabilities found on our side. Perusal of the *CPP* DTD
 3321 or Schema will reveal that the *ServiceBinding* element provides the doorway to the relevant
 3322 information from each side's *CollaborationRole* element with the *channelId* attribute. This
 3323 *channelId* attribute's value allows us to find *DeliveryChannels* within each *CPP*. The
 3324 *DeliveryChannel* has a *transportId* attribute that allows us to find the relevant *Transport*
 3325 subtrees.

3326

3327 For example, suppose that our intended buyer has a *Transport* entry:

3328
 3329 <Transport transportId = "buyerid001">
 3330 <SendingProtocol>HTTP</SendingProtocol>
 3331 <ReceivingProtocol>
 3332 HTTP
 3333 </ReceivingProtocol>
 3334 <Endpoint uri = "https://www.buyername.com/po-response"
 3335 type = "allPurpose"/>
 3336 <TransportSecurity>
 3337 <Protocol version = "1.0">TLS</Protocol>
 3338 <CertificateRef certId = certid001">BuyerName</CertificateRef>
 3339 </TransportSecurity>
 3340 </Transport>

3341

3342 and our seller has a *Transport* entry:

3343
 3344 <Transport transportId = "sellid001">
 3345 <SendingProtocol>HTTP</SendingProtocol>
 3346 <ReceivingProtocol>
 3347 HTTP
 3348 </ReceivingProtocol>
 3349 <Endpoint uri = "https://www.sellername.com/pos_here"
 3350 type = "allPurpose"/>
 3351 <TransportSecurity>
 3352 <Protocol version = "1.0">TLS</Protocol>
 3353 <CertificateRef certId = "certid002">Sellername</CertificateRef>

3354 </TransportSecurity>
3355 </Transport>

3356
3357 A transport match for requests involves finding the initiator role or buyer has a *SendingProtocol*
3358 that matches one of our *ReceivingProtocols*. So here, "HTTP" provides a match. A transport
3359 match for responses involves finding the responder role or seller has a *SendingProtocol* that
3360 matches one of the buyer's *ReceivingProtocols*. So in the above example, "HTTP" again
3361 provides a match. When such matches exist, we then have discovered an interoperable solution at
3362 the transport level. If not, no *CPA* will be available, and a high-priority gap has been identified
3363 that will need to be remedied by whatever exception handling procedures are in place.

3364
3365

3366 Matching Transport Security

3367
3368 Matches in transport security, such as in the above, will reflect agreement in versions and values
3369 of protocols. Software can supply some knowledge here so that if one side has SSL-3 and the
3370 other TLS-1, it can guess that security is available by means of a fallback of TLS to SSL.

3371
3372

3373 Matching Document Packaging

3374
3375 Probably one of the most complex matching problems arises when it comes to finding whether
3376 there are matches in document-packaging capabilities. Here both security and other MIME
3377 handling capabilities can combine to create complexity for appraising whether full
3378 interoperability can be attained.

3379

3380 Access to the information needed for undertaking this task is found under the *ServiceBinding*
3381 elements, and again we suppose that each side has just one *ServiceBinding* element. However,
3382 we will initially suppose that two *Packaging* elements are available to consider under each role.
3383 Several quite different ways of thinking about the matching task are available, and several
3384 methods for the tasks MAY be performed when assessing whether a good enough match exists.

3385

3386 To continue our previous purchase-ordering example, we recall that the packaging is the
3387 particular combination of body parts, XML instances (*Headers* and payloads), and security
3388 encapsulations used in assembling the *Message* from its data sources. Both requests and
3389 responses will have packaging. The most complete specification of packaging, which MAY not
3390 always be needed, would consist of:

3391

- 3392 1. The buyer asserting what packaging it can generate for its purchase order, and what
3393 packaging it can parse for its purchase order response *Messages*.
3394 2. The seller asserting what packaging it can generate for its purchase order responses and
3395 what packaging it can parse for received purchase orders.

3396

3397 Matching by structural comparison would then involve comparing the packaging details of the
3398 purchase orders generated by the seller with the purchase orders parsable by the buyer. The
3399 comparison would seek to establish that the MIME types of the *SimplePart* elements of
3400 corresponding subtrees match and would then proceed to check that the *CompositeList* matched

3401 in MIME types and in sequence of composition.

3402

3403 For example, if each *CPP* contained the packaging subtrees below, and under the appropriate
3404 **ServiceBindings**, then there would be a straightforward match by structural comparison:

3405

```

3406 <Packaging id="I1001">
3407     <ProcessingCapabilities parse = "true" generate = "true"/>
3408     <SimplePart id = "P1" mimetype = "text/xml"/>
3409         <NamespaceSupported location
3410             = "http://schemas.xmlsoap.org/soap/envelope/" version = "1.1">
3411             http://schemas.xmlsoap.org/soap/envelope
3412         </NamespaceSupported>
3413     <NamespaceSupported location =
3414         "http://www.ebxml.org/namespaces/messageHeader"
3415         version = "1.0">
3416         http://www.ebxml.org/namespaces/messageHeader
3417     </NamespaceSupported>     <NamespaceSupported location =
3418         "http://www.w3.org/2000/09/xmldsig#"
3419         version = "1.0">
3420         http://www.w3.org/2000/09/xmldsig#
3421     </NamespaceSupported>
3422     <SimplePart id = "P2" mimetype = "application/xml"/>
3423     <CompositeList>
3424         <Composite mimetype = "multipart/related" id = "P3"
3425             mimeparameters = "type=text/xml">
3426             <Constituent idref = "P1"/>
3427             <Constituent idref = "P2"/>
3428         </Composite>
3429     </CompositeList>
3430 </Packaging>
3431 <Packaging id="I2001">
3432     <ProcessingCapabilities parse = "true" generate = "true"/>
3433     <SimplePart id = "P11" mimetype = "text/xml"/>
3434     <SimplePart id = "P12" mimetype = "application/xml"/>
3435     <CompositeList>
3436         <Composite mimetype = "multipart/related" id = "P13"
3437             mimeparameters = "type=text/xml">
3438             <Constituent idref = "P11"/>
3439             <Constituent idref = "P12"/>
3440         </Composite>
3441     </CompositeList>
3442 </Packaging>

```

3443

3444 However, it is to be expected that over time it will become possible only to assert what
3445 packaging is *generated* within each **ServiceBinding** for the requester and responder roles. This
3446 simplification assumes that each side has knowledge of what MIME types it handles correctly,
3447 what encapsulations it handles correctly, and what composition modes it handles correctly. By
3448 scanning the packaging specifications against its lists of internal capabilities, it can then look up
3449 whether other side's generated packaging scheme is one it can process and accept it under those
3450 conditions. Knowing what generated packaging style was produced by the other side could
3451 enable the software agent to propose a packaging scheme using only the MIME types and
3452 packaging styles used in the incoming *Message*. Such a packaging scheme would be likely to be
3453 acceptable to the other side when included within a proposed *CPA*. Over time, and as proposal
3454 and negotiation conventions get established, it is to be expected that the methods used for
3455 determining a match in packaging capabilities will move away from structural comparison to
3456 simpler methods, using more economical representations. For example, parsing capabilities may
3457 eventually be captured by using a compact description of the accepting grammar for the

3458 packaging and content labelling schemes that can be parsed and for which semantic handlers are
3459 available.

3460

3461 Matching Document-Level Security

3462

3463 Although the matching task for document-level security is a subtask of the Packaging-matching
3464 task, it is useful to discuss some specifics tied to the three major document-level security
3465 approaches found in [S/MIME], OpenPGP[RFC2015], and XMLDsig[XMLDSIG].

3466

3467 XMLDsig matching capability can be inferred from document-matching capabilities when the
3468 use of ebXML *Message Service*[ebMS] packaging is present. However, there are other sources
3469 that should be checked to confirm this match. A *SimplePart* element can have a
3470 *NameSpaceSupported* element. XMLDsig capability should be found there. Likewise, a detailed
3471 check on this match should examine the information under the *NonRepudiation* element and
3472 similar elements under the ebXMLBinding element to check for compatibility in hash functions
3473 and algorithms.

3474

3475 The existence of several radically different approaches to document-level security, together with
3476 the fact that it is unusual at present for a given *Party* to commit to more than one form of such
3477 security, means that there can be basic failures to match security frameworks. Therefore, there
3478 might be no match in capabilities that supports full interoperability at all levels. For the moment,
3479 we assume that document-level security matches will require both sides able to handle the same
3480 security composites (multipart/signed using S/MIME, for example.)

3481

3482 However, suppose that there are matches at the transport and transport layer security levels, but
3483 that the two sides have failures at the document-security layer because one side makes use of
3484 PGP signatures while the other uses S/MIME. Does this mean that no *CPA* can be proposed?
3485 That is not necessarily the case.

3486

3487 Both S/MIME and OpenPGP permit signatures to be packaged within "multipart/signed"
3488 composites. In such a case, it MAY be possible to extract the data and arrive at a partial
3489 implementation that falls short with respect to nonrepudiation. While neither side could check
3490 the other's signatures, it might still be possible to have confidential document transmission and
3491 transport-level authentication for the *Business* data. Eventually *CPA*-formation software MAY
3492 be created that is able to identify these exceptional situations and "salvage" a proposed *CPA* with
3493 downgraded security features. Whether the other side would accept such a proposed *CPA* would,
3494 naturally, involve what their preferences are with respect to initiating a *Business Collaboration*
3495 and sacrificing some security features. *CPA*-formation software MAY eventually be capable of
3496 these adaptations, but it is to be expected that human assistance will be required for such
3497 situations in the near term.

3498

3499 Of course, an implementation MAY simply decide to terminate looking for a *CPA* when a match
3500 fails in any crucial factor for an interoperable implementation. At the very least, the users should
3501 be warned that the only *CPAs* that can be proposed will be missing security or other normally
3502 desirable features or features recommended by the *Business Collaboration*.

3503

3504

3505 Other Considerations

3506 Though preferences among multiple capabilities are indicated by the document order in which
3507 they are listed, it is possible that ties may occur. At present, these ties are left to be resolved by a
3508 negotiation process not discussed here.

3509