



Creating A Single Global Electronic Market

Collaboration-Protocol Profile and Agreement Specification Version 0.93

ebXML Trading-Partners Team

03/19/01 5:38 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/project_teams/trade_partner/private/cpa-cpp-spec-0.93.doc

Latest version:

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

Previous version:

http://www.ebxml.org/project_teams/trade_partner/private/cpa-cpp-spec-0.911.doc

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, Sterling 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 Definitions.....	6
	4.4 Audience	6
	4.5 Assumptions.....	6
	4.6 Related Documents	6
5	Design Objectives	7
6	System Overview	8
	6.1 What This Specification Does.....	8
	6.2 Forming a <i>CPA</i> from Two <i>CPPs</i>	9
	How the <i>CPA</i> Works.....	11
	6.4 Where the <i>CPA</i> May Be Implemented.....	12
	6.5 Definition and Scope.....	12
7	<i>CPP</i> Definition.....	14
	7.1 <i>CPP</i> Structure	15
	7.2 CollaborationProtocolProfile element.....	15
	7.3 PartyInfo Element	15
	7.3.1 PartyId element	16
	7.3.2 PartyRef element.....	17
	7.3.3 CollaborationRole element.....	18
	7.3.4 ProcessSpecification Element	20
	7.3.5 Role element	22
	7.3.6 ServiceBinding element	23
	7.3.7 Override element.....	26
	7.3.8 Certificate element	27
	7.3.9 DeliveryChannel element.....	27
	7.3.10 Characteristics element	29
	7.3.11 Transport element.....	30
	7.3.12 Transport Protocol.....	30
	7.3.13 Endpoint Element.....	31
	7.3.14 Transport Protocols	32
	7.3.15 Transport Security.....	34
	7.4 DocExchange element.....	35
	7.4.1 docExchangeId attribute.....	36
	7.4.2 ebXMLBinding element.....	36
	7.4.3 version attribute.....	36
	7.4.4 MessageEncoding element.....	36
	7.4.5 ReliableMessaging element.....	37
	7.4.6 NonRepudiation element.....	38
	7.4.7 DigitalEnvelope element	39
	7.4.8 NamespaceSupported element	40
	7.5 <i>ds:Signature</i> element.....	40
	7.6 Comment element	40
8	<i>CPA</i> Definition.....	42
	8.1 <i>CPA</i> Structure	42
	8.2 CollaborationProtocolAgreement element.....	42
	8.3 CPAType element	43
	8.4 Status element	44

8.5	CPA Lifetime	44
8.5.1	Start element	44
8.5.2	End element.....	44
8.6	ConversationConstraints element.....	45
8.6.1	invocationLimit attribute.....	45
8.6.2	concurrentConversations attribute.....	45
8.7	PartyInfo element	46
8.7.1	ProcessSpecification element.....	46
8.8	ds:Signature element	46
8.8.1	Persistent Digital Signature.....	47
8.9	Comment element	48
8.10	Composing a CPA from Two CPPs.....	49
8.10.1	ID Attribute Duplication	49
8.11	Modifying Parameters of the Process-Specification Document Based on Information in the CPA.....	49
9	References.....	51
10	Conformance.....	54
11	Disclaimer	55
	Contact Information	56
	Copyright Statement	57
Appendix A	Example of CPP Document (Non-Normative)	58
Appendix B	Example of CPA Document (Non-normative)	60
Appendix C	DTD Corresponding to Complete CPP/CPA Definition (Normative)	64
Appendix D	XML Schema Document Corresponding to Complete CPA Definition (Normative)	67
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 specification[BPMSPEC], 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 Process*.

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[TECHARCH].

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[EBXMLGLOSS]. 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 Definitions

63 Technical terms in this specification are defined in the ebXML Glossary[EBXMLGLOSS].

64

65 4.4 Audience

66 One target audience for this specification is implementers of ebXML services and other
67 designers and developers of middleware and application software that is to be used for
68 conducting electronic business. Another target audience is the people in each enterprise who are
69 responsible for creating *CPPs* and *CPAs*.

70

71 4.5 Assumptions

72 It is expected that the reader has an understanding of [XML] and is familiar with the concepts of
73 electronic business (e-business).

74

75 4.6 Related Documents

76 Related documents include ebXML Specifications on the following topics:

- 77 • ebXML Technical Architecture Specification[TECHARCH]
- 78 • ebXML *Message* Service Specification[MSSPEC]
- 79 • ebXML Business Process Specification SchemaBPMSPEC]
- 80 • ebXML Glossary [EBXMLGLOSS]
- 81 • ebXML Core Components Specification[EBXMLCC]
- 82 • ebXML Registry and Repository Specification[REGREP]

83

84 See Section 9 for the complete list of references.

85

86 5 Design Objectives

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

95
96 In addition to supporting direct interaction between two *Parties*, this specification *MAY* also be
97 used to support interaction between two *Parties* through an intermediary such as a portal or
98 broker. In this initial version of this specification, this *MAY* be accomplished by creating a *CPA*
99 between each *Party* and the intermediary in addition to the *CPA* between the two *Parties*. The
100 functionality needed for the interaction between a *Party* and the intermediary is described in the
101 *CPA* between the *Party* and the intermediary. The functionality needed for the interaction
102 between the two *Parties* is described in the *CPA* between the two *Parties*.

103
104 It is an objective of this specification that a *CPA* *SHALL* be capable of being composed by
105 intersecting the respective *CPPs* of the *Parties* involved. The resulting *CPA* *SHALL* contain
106 only those elements that are in common, or compatible, between the two *parties*. Variable
107 quantities, such as number of retries of errors, are then negotiated between the two *Parties*. The
108 design of the *CPP* and *CPA* schemata facilitates this composition/negotiation process. However,
109 the composition and negotiation processes themselves are outside the scope of this
110 specification. Appendix F contains a non-normative discussion of this subject.

111
112 It is a further objective of this specification to facilitate migration of both traditional EDI-based
113 applications and other legacy applications to platforms based on the ebXML specifications. In
114 particular, the *CPP* and *CPA* are components of the migration of applications based on the X12
115 838 Trading-Partner Profile to more automated means of setting up business relationships and
116 doing business under them.

117 6 System Overview

118 6.1 What This Specification Does

119 The exchange of information between two *Parties* requires each *Party* to know the other *Party's*
 120 supported *Collaborative Processes*, the other *Party's* role in the *Collaborative Process*, and the
 121 technology details about how the other *Party* sends and receives *Messages*. In some cases, it is
 122 necessary for the two *Parties* to reach agreement on some of the details.

123

124 The way each *Party* can exchange information, in the context of a *Collaborative Process*, can be
 125 described by a *Collaboration-Protocol Profile (CPP)*. The agreement between the *Parties* can be
 126 expressed as a *Collaboration-Protocol Agreement (CPA)*

127

128 To enable *Parties* wishing to do business to find other *Parties* that are suitable *Business*
 129 *Partners*, *CPPs* MAY be stored in a repository such as is provided by the ebXML
 130 Registry[REGREP]. Using a discovery process provided as part of the specifications of a
 131 repository, a *Party* MAY then use the facilities of the repository to find *Business Partners*.

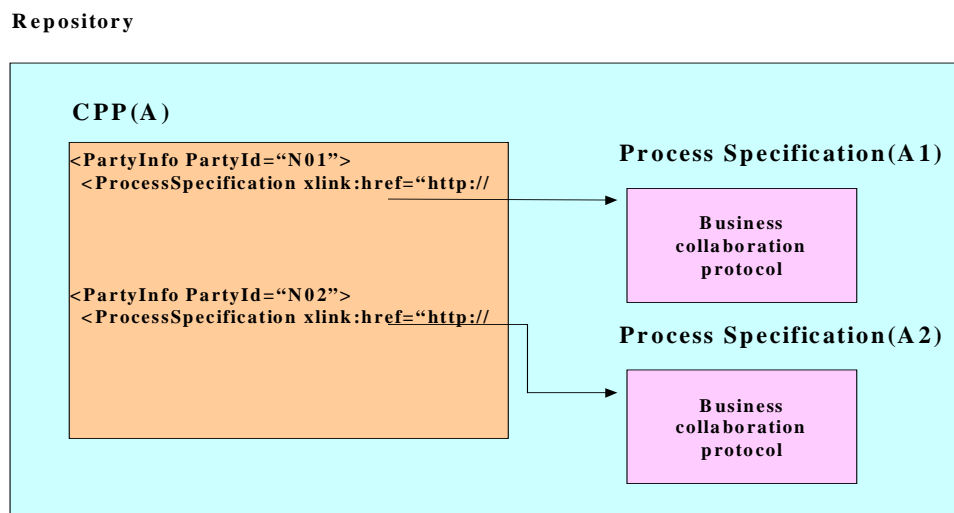
132

133 The document that defines the interactions between two *Parties* is an [XML] document called a
 134 *Process-Specification* document that conforms to the ebXML Business Process Specification
 135 Schema specification[BPMSPEC]. The *CPP* and *CPA* include references to this *Process-*
 136 *Specification* document. The *Process-Specification* document MAY also be stored in a repository
 137 such as the ebXML Registry.

138

139 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



140 A1 and A2, in an ebXML Registry. On the left is a *CPP*, A, that includes information about two

141 parts of an enterprise that are represented as different *Parties*. On the right are shown two
142 *Process-Specification* documents. Each of the ***PartyInfo*** elements in the *CPP* contains a
143 reference to one of the *Process-Specification* documents. This identifies the *Business Process*
144 that the *Party* can perform.

145
146 This specification defines the markup language vocabulary for creating electronic *CPPs* and
147 *CPAs*. *CPPs* and *CPAs* are [XML] documents. In the appendices of this specification are a
148 sample *CPP*, a sample *CPA*, the DTD, and the corresponding XML Schema document.

149
150 The *CPP* describes the capabilities of an individual *Party*. A *CPA* describes the capabilities that
151 two *Parties* have agreed to use to perform a particular *Business Process*. These *CPAs* define the
152 "information technology terms and conditions" that enable *Business* documents to be
153 electronically interchanged between *Parties*. The information content of a *CPA* is similar to the
154 information-technology specifications sometimes included in Electronic Data Interchange (EDI)
155 trading-partner agreements (TPA). However, these *CPAs* are not paper documents. Rather, they
156 are electronic documents that can be processed by computers at the *Parties'* sites in order to set
157 up and then execute the desired business information exchanges. The "legal" terms and
158 conditions of a business agreement are outside the scope of this specification and therefore are
159 not included in the *CPP* and *CPA*.

160
161 An enterprise MAY choose to represent itself as multiple *Parties*. For example, it might
162 represent a central office supply procurement organization and a manufacturing supplies
163 procurement organization as separate *Parties*. The enterprise MAY then construct a *CPP* that
164 includes all of its units that are represented as separate *Parties*. In the *CPP*, each of those units
165 would be represented by a separate ***PartyInfo*** element.

166
167 In general, the *Parties* to a *CPA* can have both client and server characteristics. A client requests
168 services and a server provides services to the *Party* requesting services. In some applications,
169 one *Party* only requests services and one *Party* only provides services. These applications have
170 some resemblance to traditional client-server applications. In other applications, each *Party*
171 MAY request services of the other. In that case, the relationship between the two *Parties* can be
172 described as a peer-peer relationship rather than a client-server relationship.

173

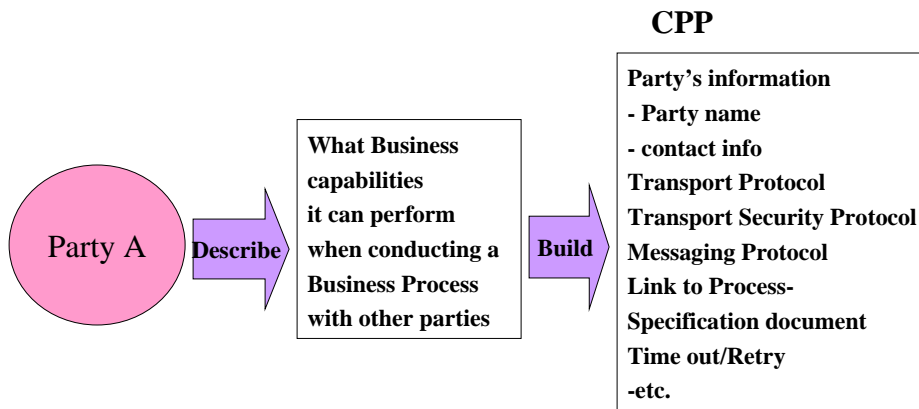
174 **6.2 Forming a CPA from Two CPPs**

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

179

180 Figure 2 illustrates forming a *CPP*. *Party A* tabulates the information to be placed in a
181 repository for the discovery process, constructs a *CPP* that contains this information, and enters
182 it into an ebXML Registry or similar repository.

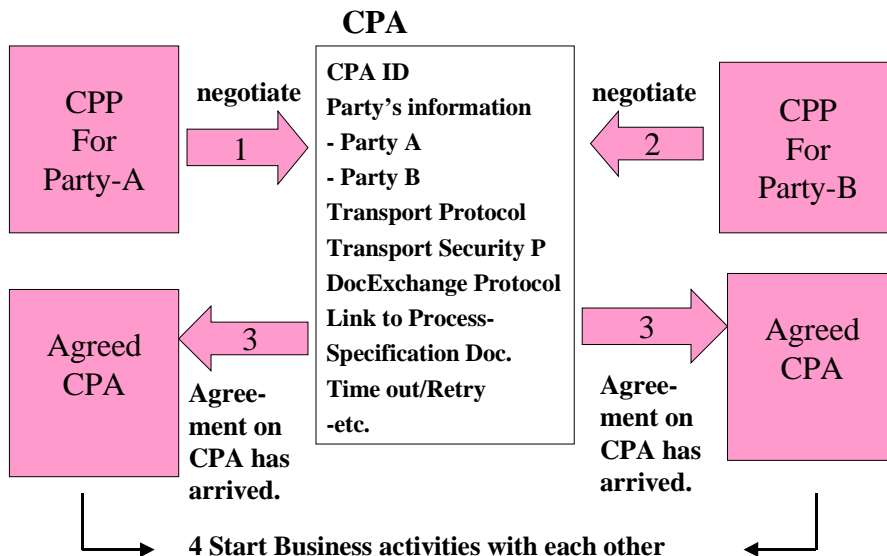
Figure 2: Overview of Collaboration-Protocol Profiles (CPP)



183

184 In figure 3, *Party A* and *Party B* use their *CPPs* to jointly construct a single copy of a *CPA* by
 185 calculating the intersection of the information in their *CPPs*. The resulting *CPA* defines how the
 186 two *parties* will behave in performing their *Collaborative Process*.

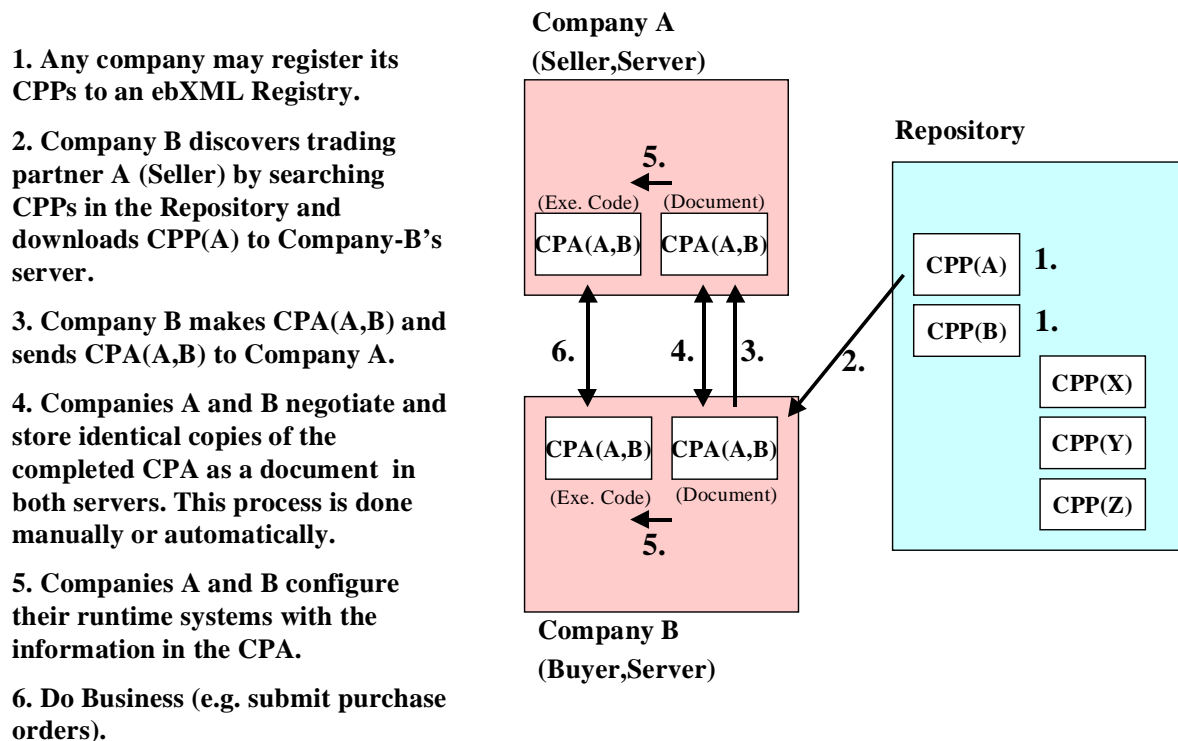
Figure 3: Overview of Collaboration-Protocol Agreements (CPA)



187

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

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



191 6.3 How the CPA Works

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

199
 200 The information in the *CPA* is used to configure the *Parties'* systems to enable exchange of
 201 *Messages* in the course of performing the selected *Business Process*. Typically, the software that
 202 performs the *Messages* exchanges and otherwise supports the interactions between the *Parties* is
 203 middleware that can support any selected *Business Process*. One component of this middleware
 204 is the ebXML *Message Service Handler*[MSSPEC]. In this specification, the term "runtime
 205 system" or "runtime software" is used to denote such middleware.

206

207 The *CPA* and the *Process-Specification* document that it references define a conversation
208 between the two *Parties*. The conversation represents a single unit of business as defined by the
209 *Binary-Collaboration* component of the *Process-Specification* document. The conversation
210 consists of one or more *Business Transactions*, each of which is a request *Message* from one
211 *Party* and a response *Message* from the other *Party*. The *Process-Specification* document
212 defines, among other things, the request and response *Messages* for each *Business Transaction*
213 and the order in which the *Business Transactions* are REQUIRED to occur. See [BPMSPEC] for
214 a detailed explanation.

215
216 The *CPA* MAY actually reference more than one *Process-Specification* document. When a *CPA*
217 references more than one *Process-Specification* document, each *Process-Specification* document
218 defines a distinct type of conversation. Any one conversation involves only a single *Process-*
219 *Specification* document.

220
221 A new conversation is started each time a new unit of business is started. The *Business Process*
222 also determines when the conversation ends. From the viewpoint of a *CPA* between *Party A* and
223 *Party B*, the conversation starts at *Party A* when *Party A* sends the first request *Message* to
224 *Party B*. At *Party B*, the conversation starts when it receives the first request of the unit of
225 business from *Party A*. A conversation ends when the *Parties* have completed the unit of
226 business.

227
228 NOTE: The run-time system SHOULD provide an interface by which the business
229 application can request initiation and ending of conversations.

230

231 **6.4 Where the CPA May Be Implemented**

232 Conceptually, the *CPA* and *Process-Specification* document are implemented by a business-to-
233 business (B2B) server at each *Party's* site. The B2B server includes the runtime software, i.e. the
234 middleware that supports communication with the other *Party*, execution of the functions
235 specified in the *CPA*, interfacing to each *Party's* back-end processes, and logging the interactions
236 between the *Parties* for purposes such as audit and recovery. The middleware might support the
237 concept of a long-running conversation as the embodiment of a single unit of business between
238 the *Parties*. To configure the two *Parties'* systems for business to business operations, the
239 information in the copy of the *CPA* and *Process-Specification* documents at each *Party's* site is
240 installed in the run-time system. The static information MAY be recorded in a local database and
241 other information in the *CPA* and *Process-Specification* document MAY be used in generating or
242 customizing the necessary code to support the *CPA*.

243
244 NOTE: It is possible to provide a graphic *CPP/CPA*-authoring tool that understands both
245 the semantics of the *CPP/CPA* and the XML syntax. Equally important, the definitions in
246 this specification make it feasible to automatically generate, at each *Party's* site, the code
247 needed to execute the *CPA*, enforce its rules, and interface with the *Party's* back-end
248 processes.

249

250 **6.5 Definition and Scope**

251

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

258
259 NOTE: This specification is limited to defining the contents of the *CPP* and *CPA*, and it is
260 possible to be conformant with it merely by producing a *CPP* or *CPA* document that
261 conforms to the DTD and XML Schema documents defined herein. It is, however, important
262 to understand that the value of this specification lies in its enabling a runtime system that
263 supports electronic commerce between two *Parties* under the guidance of the information in
264 the *CPA*.

265 7 CPP Definition

266 A *CPP* defines the capabilities of a *Party* to engage in electronic business with other *Parties*.
267 These capabilities include both technology capabilities such as supported communication and
268 messaging protocols, and business capabilities in terms of what *Business Processes* it supports.

269
270 This section defines and discusses the details in the *CPP* in terms of the individual XML
271 elements. The discussion is illustrated with some XML fragments. See Appendix C and
272 Appendix D for the DTD and XML Schema, respectively, and Appendix A for a sample *CPP*
273 document.

274
275 The *ProcessSpecification*, *DeliveryChannel*, *DocExchange*, and *Transport* elements of the
276 *CPP* describe the processing of a unit of business (conversation). These elements form a layered
277 structure somewhat analogous to a layered communication model. The remainder of this section
278 describes both the above-mentioned elements and the corresponding run-time processing.

279
280 **Process-Specification layer** - The Process-Specification layer defines the heart of the business
281 agreement between the *Parties*: the services (*Business Transactions*) which *Parties* to the *CPA*
282 can request of each other and transition rules that determine the order of requests. This layer is
283 defined by the separate *Process-Specification* document that is referenced by the *CPP* and *CPA*.

284
285 **Delivery Channels** - A delivery channel describes a *Party's Message*-receiving characteristics. It
286 consists of one document-exchange definition and one transport definition. Several delivery
287 channels MAY be defined in one *CPP*.

288
289
290 **Document-Exchange layer** - The document-exchange layer accepts a business
291 from the *Process-Specification* layer at one *Party*, encrypts it if specified, adds a digital signature
292 for nonrepudiation if specified, and passes it to the transport layer for transmission to the other
293 *Party*. It performs the inverse steps for received *Messages*. The options selected for the
294 document-exchange layer are complementary to those selected for the transport layer. For
295 example, if *Message* security is desired and the selected transport protocol does not provide
296 *Message* encryption, then it must be specified at the document-exchange layer. The protocol for
297 exchanging *Messages* between two *Parties* is defined by the ebXML *Message Service*
298 Specification [MSSPEC] or other similar messaging service.

299
300 **Transport layer** - The transport layer is responsible for *Message* delivery using the selected
301 transport protocol. The selected protocol affects the choices selected for the document-exchange
302 layer. For example, some transport-layer protocols might provide encryption and authentication
303 while others have no such facility.

304
305 It should be understood that the functional layers encompassed by the *CPP* have no
306 understanding of the contents of the payload of the business documents.

307

308 7.1 CPP Structure

309 This section describes the overall structure of the *CPP*. Unless otherwise noted, *CPP* elements
 310 MUST be in the order shown here. Subsequent sections describe each of the elements in greater
 311 detail.
 312

```

313
314 <CollaborationProtocolProfile
315     xmlns="http://www.ebxml.org/namespaces/tradePartner"
316     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
317     xmlns:xlink="http://www.w3.org/1999/xlink">
318     <PartyInfo>  <!--one or more-->
319         ...
320     </PartyInfo>
321     <ds:Signature>  <!--zero or one-->
322         ...
323     </ds:Signature>
324     <Comment>text</Comment>  <!--zero or more-->
325 </CollaborationProtocolProfile>
326
```

327 7.2 CollaborationProtocolProfile element

328 The *CollaborationProtocolProfile* element is the root element of the *CPP* XML document. The
 329 REQUIRED [XML] Namespace[XMLNS] declarations for the basic document are as follows:

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

334
 335 The *CollaborationProtocolProfile* element SHALL consist of the following child elements:

- 336 • One or more REQUIRED *PartyInfo* elements that identify the organization (or parts
 337 of the organization) whose capabilities are described by the *CPP*.
- 338 • Zero or one *ds:Signature* elements that contain the digital signature that signs the
 339 *CPP* document.
- 340 • Zero or more *Comment* elements.

341
 342 A *CPP* document MAY be digitally signed so as to provide for a means of ensuring that the
 343 document has not been altered (integrity) and to provide for a means of authenticating the author
 344 of the document. A digitally signed *CPP* SHALL be signed using technology that conforms to
 345 the joint W3C/IETF XML Digital Signature specification[XMLDSIG].
 346

347 7.3 PartyInfo Element

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

```

353
354 <PartyInfo>
355     <PartyId type="...">  <!--one or more-->
```

```

356     ...
357     </PartyId>
358     <PartyRef xlink:type="...", xlink:href="..."/>
359     <CollaborationRole> <!--one or more-->
360     ...
361     </CollaborationRole>
362     <Certificate> <!--one or more-->
363     ...
364     </Certificate>
365     <DeliveryChannel> <!--one or more-->
366     ...
367     </DeliveryChannel>
368     <Transport> <!--one or more-->
369     ...
370     </Transport>
371     <DocExchange> <!--one or more-->
372     ...
373     </DocExchange>
374 </PartyInfo>
375

```

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

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

7.3.1 PartyId element

The REQUIRED **PartyId** element provides a logical identifier that MAY be used to logically identify the *Party*. Additional **PartyId** elements MAY be present so as to provide for alternative logical identifiers for the *Party*. This permits a large organization, for example, to have different identifiers for different purposes.

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

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

407

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

410

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

415

416 The following example illustrates two URI references.

417

```
418 <PartyId type = "uriReference">urn:duns.com:duns:1234567890123</PartyId>  
419 <PartyId type = "uriReference">urn:www.example.com</PartyId>
```

420

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

424

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

427

428 7.3.2 PartyRef element

429

430 The *PartyRef* element provides a link, in the form of a URI, to additional information about the
431 *Party*. Typically, this would be the URL from which the information can be obtained. The
432 information might be at the *Party's* web site or in a publicly accessible repository such as an
433 ebXML Registry, a UDDI repository, or an LDAP directory. Information available at that URI
434 MAY include contact names, addresses, and phone numbers, and perhaps more information
435 about the *Business Processes* that the *Party* supports. This information MAY be in the form of
436 an ebXML Core Component[EBXMLCC]. It is not within the scope of this specification to
437 define the content or format of the information at that URI.

438

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

- 440 • a REQUIRED *xlink:type* attribute,
- 441 • a REQUIRED *xlink:href* attribute.

442

443 7.3.2.1 xlink:type attribute

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

446

447 7.3.2.2 xlink:href attribute

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

450

451 An example of the *PartyRef* element is:

452

```

453     <PartyRef xlink:type="simple"
454           xlink:href="http://example2.com/ourInfo.html"/>

```

455 7.3.3 CollaborationRole element

```

456 <CollaborationRole id="N11" >
457   <ProcessSpecification name="BuySell" version="1.0">
458     ...
459   </ProcessSpecification>
460   <Role name="buyer" xlink:href="..."/>
461   <CertificateRef certId = "N03"/>    <!-- primary binding with "preferred"
462     DeliveryChannel -->
463   <ServiceBinding name="some process" channelId="N02">
464     <!-- override "default" deliveryChannel for selected message(s)-->
465     <Override action="OrderAck" channelId="N05"
466       xlink:type="locator"
467       xlink:href="..."/>
468     <Packaging>
469       ...
470     </Packaging> <!--one or more-->
471   </ServiceBinding>
472   <!-- the first alternate binding -->
473   <ServiceBinding channelId="N04">
474     <Override action="OrderAck" channelId="N05"
475     xlink:type="locator"
476     xlink:href="..."/>
477     <Packaging>
478       ...
479     </Packaging> <!--one or more-->
480   </ServiceBinding>
481 </CollaborationRole>

```

482 The **CollaborationRole** element associates a *Party* with a specific role in the *Business Process* that is defined in the *Process-Specification* document[BPMSPEC]. Generally, the *Process Specification* is defined in terms of roles such as "buyer" and "seller". The association between a specific *Party* and the role(s) it is capable of fulfilling within the context of a *Process Specification* is defined in both the *CPP* and *CPA* documents. In a *CPP*, the **CollaborationRole** element identifies which role the *Party* is capable of playing in each *Process Specification* documents referenced by the *CPP*.

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

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

506

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

512

513 NOTE: When a *CPA* is composed, the *ServiceBinding* preferences are applied in
514 choosing the highest-preference delivery channels that are compatible between the two
515 *Parties*.

516

517 When a *CPA* is composed, only *ServiceBinding* elements that are compatible between the two
518 *Parties* SHALL be retained. Each *Party* SHALL have a default delivery channel for each
519 *Process-Specification* document referenced in the *CPA*. For each *Process-Specification*
520 document, the default delivery channel for each *Party* is the delivery channel that is indicated by
521 the *channelId* attribute in the highest-preference *ServiceBinding* element that references that
522 *Process-Specification* document.

523

524 NOTE: An implementation MAY provide the capability of dynamically assigning
525 delivery channels on a per *Message* basis during performance of the *Business Process*.
526 The delivery channel selected would be chosen, based on present conditions, from those
527 identified by *ServiceBinding* elements that refer to the *Business Process* that is sending
528 the *Message*. If more than one delivery channel is applicable, the one referred to by the
529 highest-preference *ServiceBinding* element is used.

530

531 The *CollaborationRole* element has the following attribute:

- 532 • a REQUIRED *id* attribute.

533

534 7.3.3.1 *id* attribute

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

537

538 7.3.3.2 *CertificateRef* element

539 The EMPTY *CertificateRef* element contains an IMPLIED attribute, *certId* that identifies the
540 certificate to be used by referring to the *Certificate* element (under *PartyInfo*) that has the
541 matching ID attribute value.

542

543 7.3.3.3 *certId* attribute

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

546

547 NOTE: This *certID* attribute relates to the authorizing role in the *Process Specification*
548 while the certificates identified in the delivery-channel description relate to *Message*
549 exchanges.

550

551

552 7.3.4 ProcessSpecification Element

553 The *ProcessSpecification* element provides the link to the *Process-Specification* document that
 554 defines the interactions between the two *Parties*. This document is prepared in accord with the
 555 ebXML Business Process Specification Schema specification[BPMSPEC]. The *Process-*
 556 *Specification* document MAY be kept in an ebXML Registry.

557
 558 The syntax of the *ProcessSpecification* element is:

```
559 <ProcessSpecification
560     name="BuySell"
561     version="1.0"
562     xlink:type="locator"
563     xlink:href="http://www.ebxml.org/services/purchasing.xml"
564     <ds:Reference ds:URI="http://www.ebxml.org/services/purchasing.xml">
565         <ds:Transforms>
566             <ds:Transform
567                 ds:Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
568             </ds:Transforms>
569             <ds:DigestMethod
570                 ds:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
571                 String
572             </ds:DigestMethod>
573             <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
574         </ds:Reference>
575     </ProcessSpecification>
```

576
 577 As an alternative to the string value of the *ds:DigestMethod*, the child element,
 578 *ds:HMACOutputLength*, with a string value, MAY be used. See the XML Digital Signature
 579 Specification[XMLDSIG] for more information.

580
 581 The *ProcessSpecification* element has a single REQUIRED child element, *ds:Reference*, and the
 582 following attributes:

- 583 • a REQUIRED *name* attribute, with type ID,
- 584 • a REQUIRED *version* attribute,
- 585 • a FIXED *xlink:type* attribute,
- 586 • a REQUIRED *xlink:href* attribute.

587
 588 The *ds:Reference* element relates to the *xlink:type* and *xlink:href* attributes as follows. Each
 589 *ProcessSpecification* element SHALL contain one *xlink:href* attribute and one *xlink:type*
 590 attribute with a value of "locator", and MAY contain one *ds:Reference* element formulated
 591 according to the XML Digital Signature specification[XMLDSIG]. In case the document is
 592 signed, it MUST use the *ds:Reference* element. When the *ds:Reference* element is present, it
 593 MUST include a *ds:URI* attribute whose value is identical to that of the *xlink:href* attribute in
 594 the enclosing *ProcessSpecification* element.

595 596 597 7.3.4.1 name attribute

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

600 601 7.3.4.2 version attribute

602 The *ProcessSpecification* element includes a REQUIRED *version* attribute to identify the

603 version of the *Process-Specification* document identified by the ***xlink:href*** attribute (and also
604 identified by the ***ds:Reference*** element, if any).

605

606 **7.3.4.3 xlink:type attribute**

607 The ***xlink:type*** attribute has a FIXED value of "locator". This identifies the element as being an
608 [XLINK] locator.

609

610 **7.3.4.4 xlink:href attribute**

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

613

614 **7.3.4.5 ds:Reference Element**

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

618

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

621

- 622 • test of the validity of a *CPP* and the referenced *Process-Specification* documents at
623 the time composition of a *CPA* begins in case they have changed since they were
624 created,
- 625 • test of the validity of a *CPA* and the referenced *Process-Specification* documents at
626 the time a *CPA* is installed into a *Party's* system,
- 627 • test of the validity of a *CPA* at intervals after the *CPA* has been installed into a *Party's*
628 system. The *CPA* and the referenced *Process-Specification* documents MAY be
629 processed by an installation tool into a form suited to the particular middleware.
630 Therefore, alterations to the *CPA* and the referenced *Process-Specification* documents
631 do not necessarily affect ongoing run-time operations. Such alterations might not be
632 detected until it becomes necessary to reinstall the *CPA* and the referenced *Process-*
633 *Specification* documents.

634

635 The syntax and semantics of the ***ds:Reference*** element and its child elements are defined in the
636 XML Digital Signature specification[XMLDSIG], with the following additional requirements:

637

- 638 • Each ***ds:Reference*** element within a ***ProcessSpecification*** element MUST specify a
639 ***ds:Transform*** to canonicalize the reference, and that transform MUST be Canonical
640 XML[XMLC14N]. Note that implementation of Canonical XML is REQUIRED by the
641 XML Digital Signature specification[XMLDSIG].
- 642 • A ***ds:Reference*** element within a ***ProcessSpecification*** element SHALL NOT specify a
643 ***ds:Transform*** that would alter the canonical form of the reference as defined by
644 Canonical XML[XMLC14N].

645

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

648

- 649 • A *CPP* MUST be considered invalid if any *ds:Reference* element within a
650 *ProcessSpecification* element fails reference validation as defined by the XML Digital
651 Signature specification[XMLDSIG].
652
- 653 • A *CPP* MUST be considered invalid if any *ds:Reference* within it cannot be
654 dereferenced.
655

656 Other validity implications of such *ds:Reference* elements are specified in the description of the
657 *ds:Signature* element.
658

659 NOTE: The XML Digital Signature specification[XMLDSIG] states "The signature
660 application MAY rely upon the identification (URI) and Transforms provided by the
661 signer in the Reference element, or it MAY obtain the content through other means such
662 as a local cache" (emphases on MAY added). However, it is RECOMMENDED that
663 ebXML *CPP/CPA* implementations not make use such cached results when signing or
664 validating.
665

666 NOTE: It is recognized that the XML Digital Signature specification[XMLDSIG]
667 provides for signing an XML document together with externally referenced documents.
668 In cases where a *CPP* or *CPA* document is in fact suitably signed, that facility could also
669 be used to ensure that the referenced *Process-Specification* documents are unchanged.
670 However, this specification does not currently mandate that a *CPP* or *CPA* be signed.
671

672 NOTE: If the *Parties* to a *CPA* wish to customize a previously existing *Process-*
673 *Specification* document, they MAY copy the existing document, modify it, and cause
674 their *CPA* to reference the modified copy. It is recognized that for reasons of clarity,
675 brevity, or historical record, the parties might prefer to reference a previously existing
676 *Process-Specification* document in its original form and accompany that reference with a
677 specification of the agreed modifications. Therefore, *CPP* usage of the *ds:Reference*
678 element's *ds:Transforms* subelement within a *ProcessSpecification* element might be
679 expanded in the future to allow other transforms as specified in the XML Digital
680 Signature specification[XMLDSIG]. For example, modifications to the original
681 document could then be expressed as XSLT transforms. After applying any transforms,
682 it would be necessary to validate the transformed document against the ebXML Business
683 Process Specification Schema specification[BPMSPEC].
684

685 7.3.5 Role element

686 The REQUIRED *Role* element identifies which role in the *Process Specification* the *Party* is
687 capable of supporting via the *ServiceBinding* element(s) siblings within this *CollaborationRole*
688 element.
689

690 The *Role* element has the following attributes:

- 691 • a REQUIRED *name* attribute,
- 692 • a FIXED *xlink:type* attribute,
- 693 • a REQUIRED *xlink:href* attribute.

694

695 7.3.5.1 name attribute

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

- 700 • *initiator* attribute of the *binary-collaboration* element,
- 701 • *responder* attribute of the *binary-collaboration* element,
- 702 • *from* attribute of the *business-transaction-activity* element,
- 703 • *to* attribute of the *business-transaction-activity* element,
- 704 • *from* attribute of the *collaboration-activity* element,
- 705 • *to* attribute of the *collaboration-activity* element,
- 706 • *name* attribute of the *business-partner-role* element.

707

708 7.3.5.2 xlink:type attribute

709 The *xlink:type* attribute has a FIXED value of "locator". This identifies the element as being an
710 [XLINK] locator.

711

712 7.3.5.3 xlink:href attribute

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

716

717 7.3.6 ServiceBinding element

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

721

```
722     <ServiceBinding name="SomeProcess" channelId="X03">  
723         <Packaging> <!--one or more-->  
724             ...  
725         </Packaging>  
726         <Override action="OrderAck"  
727             channelId="X04"  
728             xlink:type="locator"  
729             xlink:href="..." /> <!--zero or more-->  
730     </ServiceBinding>
```

731

732 The *ServiceBinding* element SHALL have one or more REQUIRED *Packaging* child elements
733 and zero or more *Override* child elements.

734

735 The *ServiceBinding* element has the following attributes:

- 736 • a REQUIRED *name* attribute,
- 737 • a REQUIRED *channelId* attribute.

738

739 7.3.6.1 name attribute

740 The value of the REQUIRED *name* attribute is a string value that labels the *ServiceBinding*
741 element. The value of the *name* attribute SHALL be used as the value of the *Service* element in

742 the ebXML *Message Header*[MSSSPEC].

743

744 7.3.6.2 *channelId* attribute

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

748

749 The *ServiceBinding* element has one or more *Packaging* child elements. The *Packaging*
750 element MAY appear one or more times in a *CPP* as a child of each *ServiceBinding* element and
751 SHALL appear once as a child of each *ServiceBinding* element in a *CPA*.

752

753 The packaging subtree provides specific information about how the *Message Header* and
754 payload constituent(s) are packaged for transmittal over the transport, including the crucial
755 information about what document-level security packaging is used and the way in which security
756 features have been applied. Typically the subtree under the *Packaging* element indicates the
757 specific way in which constituent parts of the *Message* are organized. MIME processing
758 capabilities are typically the capabilities or agreements described in this subtree.

759

760 Following is an example of the *Packaging* element:

761

```
762     <Packaging> <!--one or more-->
763     <!--The triplet of child elements of Packaging MAY appear one
764     or more times-->
765     <ProcessingCapabilities parse="..." generate="..." />
766     <SimplePart
767         id="id" mimetype="type" /> <!--one or more-->
768     <!--The child of CompositeList is an enumeration of either
769     Composite or Encapsulation. The enumeration MAY appear one
770     or more time, with the two elements intermixed-->
771     <CompositeList
772         <Composite mimetype="type"
773             id="name"
774             mimeparameters="parameter">
775             <Constituent idref="name" />
776         </Composite>
777         <Encapsulation mimetype="type" id="name">
778             <Constituent idref="name" />
779         </Encapsulation>
780     </CompositeList>
781 </Packaging>
```

782

783 The child elements of the *Packaging* element are *ProcessingCapabilities*, *SimplePart*, and
784 *CompositeList*. This set of elements MAY appear one or more times as a child of each
785 *Packaging* element in a *CPP* and SHALL appear once as a child of each *Packaging* element in a
786 *CPA*.

787

788 The *ProcessingCapabilities* element has two attributes with REQUIRED Boolean values of
789 either "true" or "false". The attributes are *parse* and *generate*. Normally these attributes will both
790 have values of "true" to indicate that the packaging constructs specified in the other child
791 elements can be both produced as well as processed at the software *Message* service layer.

792

793 The *SimplePart* element provides a repeatable list of the constituent parts, primarily identified by

794 the MIME Content-Type value. The *SimplePart* element has two REQUIRED attributes: *id* and
795 *mimetype*. The *id* attribute, type ID, provides the value that will be used later to reference this
796 *Message* part when specifying how the parts are packaged into composites, if composite
797 packaging is present. The *mimetype* attribute provides the actual value of the Content-type for
798 the simple *Message* part being specified.

799
800 The final child element of *Packaging* is *CompositeList*, which is a container for the specific way
801 in which the simple parts are combined into groups (MIME multipart) or encapsulated within
802 security-related MIME content-types. The *CompositeList* element MAY be omitted from
803 *Packaging* when no security encapsulations or composite multipart are used. When the
804 *CompositeList* element is present, the content model for *CompositeList* is a repeatable sequence
805 of choices of *Composite* or *Encapsulation* elements. The *Composite* and *Encapsulation*
806 elements MAY appear intermixed as desired.

807
808 The sequence in which the choices are presented is important because, given the recursive
809 character of MIME packaging, *Composites* or *Encapsulations* MAY include previously
810 mentioned *Composites* (or rarely, *Encapsulations*) in addition to the *Message* parts characterized
811 within the *SimplePart* subtree. Therefore, the “top-level” packaging will be described last in the
812 sequence.

813
814 The *Composite* element has the following attributes:

- 815 • a REQUIRED *mimetype* attribute,
- 816 • a REQUIRED *id* attribute,
- 817 • an IMPLIED *mimeparameters* attribute.

818
819 The *mimetype* attribute provides the value of the MIME content-type for this *Message* part, and
820 this will be some MIME composite type, such as “Multipart/related” or “Multipart/signed”. The
821 *id* attribute, type ID, provides a way to refer to this composite if it needs to be mentioned as a
822 constituent of some later element in the sequence. The *mimeparameters* attribute provides the
823 values of any significant MIME parameter (such as “type=application/vnd.eb+xml”) that is
824 needed to understand the processing demands of the content-type.

825
826 The *Composite* element has one child element, *Constituent*.

827
828 The *Constituent* element has one REQUIRED attribute, *idref*, type IDREF, and has an EMPTY
829 content model. The *idref* attribute has as its value the value of the *id* attribute of a previous
830 *Composite*, *Encapsulation*, or *SimplePart* element. The purpose of this sequence of
831 *Constituents* is to indicate both the contents and the order of what is packaged within the current
832 *Composite* or *Encapsulation*.

833
834 The *Encapsulation* element is typically used to indicate the use of MIME security mechanisms,
835 such as [S/MIME] or Open-PGP[RFC2015]. A security body part can encapsulate a MIME part
836 that has been previously characterized. For convenience, we tag all such security structures
837 under *Encapsulation*, even when technically speaking the data is not “inside” the body part. (In
838 other words, the so-called clear-signed or detached signature structures possible with MIME
839 multipart/signed are for simplicity found under the *Encapsulation* element.)

840

841 The *Encapsulation* element has the following attributes:

- 842 • a REQUIRED *mimetype* attribute,
- 843 • a REQUIRED *id* attribute,
- 844 • an IMPLIED *mimeparameters* attribute.

845

846 The *mimetype* attribute provides the value of the MIME content-type for this *Message* part, such
847 as “application/pkcs7-mime.” The *id* attribute, type ID, provides a way to refer to this
848 encapsulation if it needs to be mentioned as a constituent of some later element in the sequence.
849 The *mimeparameters* attribute provides the values of any significant MIME parameter(s)
850 needed to understand the processing demands of the content-type.

851

852 Both the *Encapsulation* attribute and the *Composite* element have child elements consisting of a
853 *Constituent* element or of a repeatable sequence of *Constituent* elements, respectively.

854

855 7.3.7 Override element

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

859

860 Each *Override* element SHALL specify a different *DeliveryChannel* for selected *Messages* that
861 are to be received by the *Party* in the context of the *Process Specification* that is associated with
862 the parent *ServiceBinding* element.

863 The *Override* element has the following attributes:

- 864 • a REQUIRED *action* attribute,
- 865 • a REQUIRED *channelId* attribute,
- 866 • an IMPLIED *xlink:href* attribute,
- 867 • a FIXED *xlink:type* attribute.

868

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

871

872 NOTE: It is possible that when a *CPA* is composed from two *CPPs*, a delivery channel in
873 one *CPP* might have an *Override* element that will not be compatible with the other *Party*.
874 This incompatibility MUST be resolved either by negotiation or by reverting to a compatible
875 default delivery channel.

876

877 7.3.7.1 action attribute

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

882

883 7.3.7.2 channelId attribute

884 The REQUIRED *channelId* attribute is an [XML] IDREF that identifies the *DeliveryChannel*

885 element that is to be associated with the *Message* that is identified by the *action* attribute.

886

887 **7.3.7.3 xlink:href attribute**

888 The IMPLIED *xlink:href* attribute MAY be present. If present, it SHALL provide an absolute
889 [XPOINTER] URI expression that specifically identifies the *BusinessTransaction* element
890 within the associated *Process-Specification* document[BPMSPEC] that is identified by the
891 *ProcessSpecification* element.

892

893 **7.3.7.4 xlink:type attribute**

894 The IMPLIED *xlink:type* attribute has a FIXED value of "locator". This identifies the element as
895 being an [XLINK] locator.

896

897 **7.3.8 Certificate element**

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

901

```
902     <Certificate certId = "N03">  
903         <ds:KeyInfo> . . . </ds:KeyInfo>  
904     </Certificate>
```

905

906 The *Certificate* element has a single REQUIRED attribute: *certId*. The *Certificate* element has a
907 single child element: *ds:KeyInfo*.

908

909 **7.3.8.1 certId attribute**

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

912

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

914

915 **7.3.8.2 ds:KeyInfo element**

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

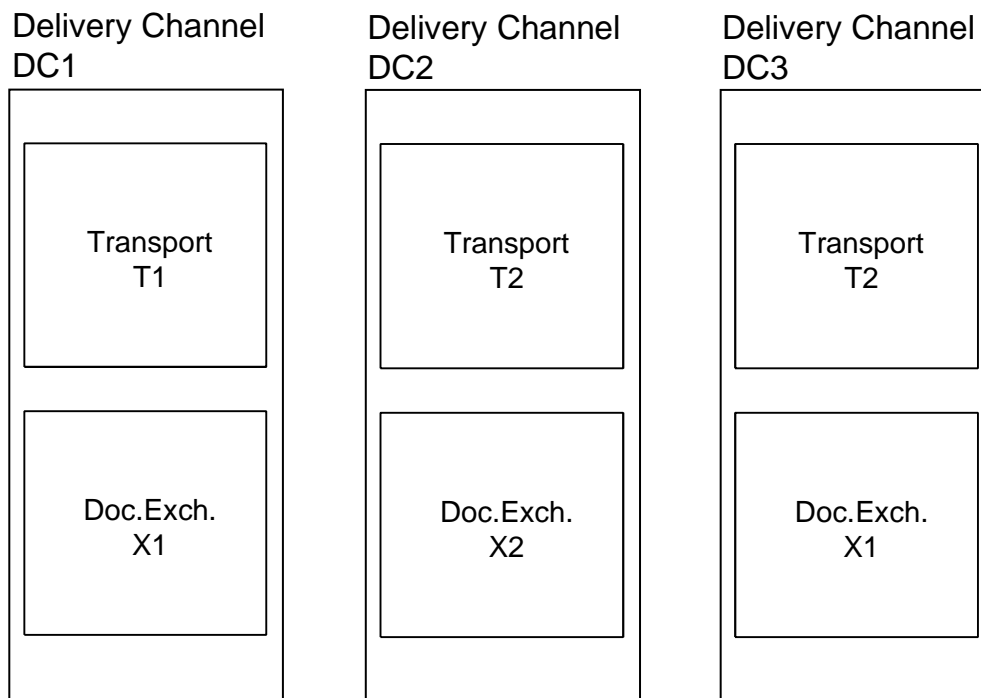
918

919 NOTE: Software for creation of *CPPs* and *CPAs* MAY recognize the *ds:KeyInfo* element
920 and insert the subelement structure necessary to define the certificate.

921

922 **7.3.9 DeliveryChannel element**

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

Figure 5: Three Delivery Channels

931
 932 The delivery channels have ID attributes with values "DC1", "DC2", and "DC3". Each delivery
 933 channel contains one transport definition and one document-exchange definition. Each transport
 934 definition and each document-exchange definition also has a name as shown in the figure. Note
 935 that delivery-channel DC3 illustrates that a delivery channel MAY refer to the same transport
 936 definition and document-exchange definition used by other delivery channels but a different
 937 combination. In this case delivery-channel DC3 is a combination of transport definition T2 (also
 938 referred to by delivery-channel DC2) and document-exchange definition X1 (also referred to by
 939 delivery-channel DC1).

940
 941 A specific delivery channel SHALL be associated with each *ServiceBinding* element or
 942 *Override* element (*action* attribute). Following is the delivery-channel syntax.

```
943
944     <DeliveryChannel channelId="N04" transportId="N05" docExchangeId="N06">
945         <Characteristics
946             nonrepudiationOfOrigin = "true"
947             nonrepudiationOfReceipt = "true"
948             secureTransport = "true"
949             confidentiality = "true"
950             authenticated = "true"
951             authorized = "true"/>
952     </DeliveryChannel>
```

953
 954 Each *DeliveryChannel* element identifies one *Transport* element and one *DocExchange* element
 955 that make up a single delivery channel definition.

956
 957 The *DeliveryChannel* element has the following attributes:

- 958 • a REQUIRED *channelId* attribute,
- 959 • a REQUIRED *transportId* attribute,
- 960 • a REQUIRED *docExchangeId* attribute.

961

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

963

964 **7.3.9.1 channelId attribute**

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

967

968 **7.3.9.2 transportId attribute**

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

972

973 **7.3.9.3 docExchangeId attribute**

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

978

979 **7.3.10 Characteristics element**

980 The *Characteristics* element describes the security characteristics provided by the delivery
981 channel. The *Characteristics* element has the following attributes:

- 982 • an IMPLIED *nonrepudiationOfOrigin* attribute,
- 983 • an IMPLIED *nonrepudiationOfReceipt* attribute,
- 984 • an IMPLIED *secureTransport* attribute,
- 985 • an IMPLIED *confidentiality* attribute,
- 986 • an IMPLIED *authenticated* attribute,
- 987 • an IMPLIED *authorized* attribute.

988

989 **7.3.10.1 nonrepudiationOfOrigin attribute**

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

993

994 **7.3.10.2 nonrepudiationOfReceipt attribute**

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

999

1000 **7.3.10.3 secureTransport attribute**

1001 The *secureTransport* attribute is a Boolean with possible values of "true" and "false". If the
1002 value is "true" then it indicates that the delivery channel uses a secure transport protocol such as

1003 [SSL] or [IPSEC].

1004

1005 **7.3.10.4 confidentiality attribute**

1006 The *confidentiality* attribute is a Boolean with possible values of "true" and "false". If the value
1007 is "true" then it indicates that the delivery channel **REQUIRES** that the *Message* be encrypted in
1008 a persistent manner. It **MUST** be encrypted above the level of the transport and delivered,
1009 encrypted, to the application.

1010

1011 **7.3.10.5 authenticated attribute**

1012 The *authenticated* attribute is a Boolean with possible values of "true" and "false". If the value
1013 is "true" then it indicates that the delivery channel **REQUIRES** that the sender of the *Message* be
1014 authenticated before delivery to the application.

1015

1016 **7.3.10.6 authorized attribute**

1017 The *authorized* attribute is a Boolean with possible of values of "true" and "false". If the value
1018 is "true" then it indicates that the delivery channel **REQUIRES** that the sender of the *Message* be
1019 authorized before delivery to the application.

1020

1021 **7.3.11 Transport element**

1022 The *Transport* element of the *CPP* defines the *Party's* capabilities with regard to communication
1023 protocol, encoding, and transport security information.

1024

1025 The overall structure of the *Transport* element is as follows:

1026

```
1027 <Transport transportId = "N05">
1028   <!--protocols are HTTP, SMTP, and FTP-->
1029   <SendingProtocol version = "1.1">HTTP</SendingProtocol>
1030   <!--one or more SendingProtocol elements-->
1031   <ReceivingProtocol version = "1.1">HTTP</ReceivingProtocol>
1032   <!--one or more endpoints-->
1033   <Endpoint uri="http://example.com/servlet/ebxmlhandler"
1034     type = "request"/>
1035   <TransportSecurity> <!--0 or 1 times-->
1036     <Protocol version = "3.0">SSL</Protocol>
1037     <CertificateRef certId = "N03"/>
1038   </TransportSecurity>
1039 </Transport>
```

1040

1041 **7.3.11.1 TransportId attribute**

1042 The *Transport* element has a single **REQUIRED** *transportId* attribute, of type [XML] ID, that
1043 provides a unique identifier for each *Transport* element, which **SHALL** be referred to by the
1044 *transportId* IDREF attribute in a *DeliveryChannel* element elsewhere within the *CPP* or *CPA*
1045 document.

1046

1047 **7.3.12 Transport Protocol**

1048 Supported communication protocols are HTTP, SMTP, and FTP. The *CPP* **MAY** specify as
1049 many protocols as the *Party* is capable of supporting.

1050

1051 NOTE: It is the aim of this specification to enable support for any transport capable of
1052 carrying MIME content using the vocabulary defined herein.
1053

1054 **7.3.12.1 SendingProtocol element**

1055 The *SendingProtocol* element identifies the protocol that a *Party* can, or will, use to send
1056 business data to its intended collaborator. The IMPLIED *version* attribute identifies the specific
1057 version of the protocol. For example, suppose that within a *CPP*, a *Transport* element,
1058 containing *SendingProtocol* elements whose values are SMTP and HTTP, is referenced within a
1059 *DeliveryChannel* element. Suppose, further, that this *DeliveryChannel* element is referenced for
1060 the role of Seller within a purchase-ordering process. Then the party is asserting that it can send
1061 purchase orders by either SMTP or HTTP. In a *CPP*, the *SendingProtocol* element MAY appear
1062 one or more times under each *Transport* element. In a *CPA*, the *SendingProtocol* element shall
1063 appear once.
1064

1065 **7.3.12.2 ReceivingProtocol element**

1066 The *ReceivingProtocol* element identifies the protocol by which a *Party* can receive its business
1067 data from the other *Party*. The IMPLIED *version* attribute identifies the specific version of the
1068 protocol. For example, within a *CPP*, if a *Transport* element is referenced within a
1069 *DeliveryChannel* element containing a *ReceivingProtocol* element whose value is HTTP, and
1070 this *DeliveryChannel* is referenced for the role of seller within a purchase ordering process, then
1071 the party is asserting that it can receive business responses to purchase orders over HTTP.
1072

1073 Within a *CPA*, the *SendingProtocol* and *ReceivingProtocol* elements serve to indicate the actual
1074 agreement upon what transports will be used for the complementary roles of the collaborators.
1075 For example, continuing the earlier examples, the seller in a purchase-order process collaboration
1076 could specify its receiving protocol to be SMTP and its sending protocol to be HTTP. These
1077 collaborator capabilities would match the buyer capabilities indicated in the *CPP*. These matches
1078 support an interoperable transport agreement where the buyer would send purchase orders by
1079 SMTP and where the responses to purchase orders (acknowledgements, cancellations, or change
1080 requests, for example) would be sent by the seller to the buyer using HTTP.
1081

1082 To fully describe receiving transport capabilities, the receiving-protocol information needs to be
1083 combined with URLs that provide the endpoints (see below).
1084

1085 NOTE: Though the URL scheme gives information about the protocol used, an explicit
1086 *ReceivingProtocol* element remains useful for future extensibility to protocols all of
1087 whose endpoints are identified by the same URL schemes, such as distinct transport
1088 protocols that all make use of HTTP endpoints. Likewise, both URL schemes of HTTP://
1089 and HTTPS:// can be regarded as the same *ReceivingProtocol*. Therefore, the
1090 *ReceivingProtocol* element is separated from the endpoints, which are, themselves,
1091 needed to provide essential information needed for connections.
1092

1093 **7.3.13 Endpoint Element**

1094 The REQUIRED *uri* attribute of the *Endpoint* element specifies the *Party's* communication
1095 addressing information associated with the *ReceiveProtocol* element. One or more *Endpoint*
1096 elements SHALL be provided for each *Transport* element in order to provide different addresses

1097 for different purposes. The value of the *uri* attribute is a URI that contains the electronic address
1098 of the *Party* in the form REQUIRED for the selected protocol. The value of the *uri* attribute
1099 SHALL conform to the syntax for expressing URIs as defined in [RFC2396].
1100

1101 The *type* attribute identifies the purpose of this endpoint. The value of *type* is an enumeration;
1102 permissible values are "login", "request", "response", "error", and "allPurpose". There can be, at
1103 most, one of each. The *type* attribute MAY be omitted. If it is omitted, its value defaults to
1104 "allPurpose". The "login" endpoint MAY be used for the address for the initial *Message* between
1105 the two *Parties*. The "request" and "response" endpoints are used for request and response
1106 *Messages*, respectively. The "error" endpoint MAY be used as the address for error *Messages*
1107 issued by the messaging service. If no "error" endpoint is defined, these error *Messages* SHALL
1108 be sent to the "response" address, if defined, or to the "allPurpose" endpoint. To enable error
1109 *Messages* to be received, each *Transport* element SHALL contain at least one endpoint of type
1110 "error", "response", or "allPurpose".
1111

1112 7.3.14 Transport Protocols

1113 In the following sections, we discuss the specific details of each supported transport protocol.
1114

1115 7.3.14.1 HTTP

1116 HTTP is Hypertext Transfer Protocol[HTTP]. For HTTP, the address is a URI that SHALL
1117 conform to [RFC2396]. Depending on the application, there MAY be one or more endpoints,
1118 whose use is determined by the application.
1119

1120 Following is an example of an HTTP endpoint:

```
1121 <Endpoint uri="http://example.com/servlet/ebxmlhandler"  
1122     type = "request"/>  
1123
```

1124
1125 The "request" and "response" endpoints MAY be dynamically overridden for a particular
1126 request or asynchronous response by application-specified URIs exchanged in business
1127 documents exchanged under the *CPA*.
1128

1129 For a synchronous response, the "response" endpoint is ignored if present. A synchronous
1130 response is always returned on the existing connection, i.e. to the URI that is identified as the
1131 source of the connection.
1132

1133 7.3.14.2 SMTP

1134 SMTP is Simple Mail Transfer Protocol[SMTP]. For use with this standard, Multipurpose
1135 Internet Mail Extensions[MIME] MUST be supported. The MIME media type used by the
1136 SMTP transport layer is "Application" with a sub-type of "octet-stream".
1137

1138 For SMTP, the communication address is the fully qualified mail address of the destination *Party*
1139 as defined by [RFC822]. Following is an example of an SMTP endpoint:

```
1140 <Endpoint uri="mailto:ebxmlhandler@example.com"  
1141     type = "request"/>  
1142
```

1143

1144 SMTP with MIME automatically encodes or decodes the document as required, on a link-by-link
1145 basis, and presents the decoded document to the destination document-exchange function. If the
1146 application design is such that the choices in the *documentExchange* element and the
1147 *ProcessSpecification* element are intended to be independent of the choice of transport protocol,
1148 it is permissible to specify a *MessageEncoding* element under the *DocExchange* element.
1149

1150 NOTE: The SMTP mail transfer agent encodes binary data (i.e. data that are not 7-bit
1151 ASCII) unless it is aware that the upper level (mail user agent) has already encoded the
1152 data. If the data are encoded in the document-exchange level (*MessageEncoding*), the
1153 information that the data are already encoded SHOULD be passed to the mail user agent.
1154

1155 NOTE: SMTP by itself (without any authentication or encryption) is subject to denial of
1156 service and masquerading by unknown *Parties*. It is strongly suggested that those *Parties*
1157 who choose SMTP as their transport layer also choose a suitable means of encryption and
1158 authentication either in the document-exchange layer or in the transport layer such as
1159 [S/MIME].
1160

1161 NOTE: SMTP is an asynchronous protocol that does not guarantee a particular quality of
1162 service. A transport-layer acknowledgment (i.e. an SMTP acknowledgment) to the
1163 receipt of a mail *Message* constitutes an assertion on the part of the SMTP server that it
1164 knows how to deliver the mail *Message* and will attempt to do so at some point in the
1165 future. However, the *Message* is not hardened and might never be delivered to the
1166 recipient. Furthermore, the sender will see a transport-layer acknowledgment only from
1167 the nearest node. If the *Message* passes through intermediate nodes, SMTP does not
1168 provide an end-to-end acknowledgment. Therefore receipt of an SMTP
1169 acknowledgement does not guarantee that the *Message* will be delivered to the
1170 application and failure to receive an SMTP acknowledgment is not evidence that the
1171 *Message* was not delivered. It is recommended that the reliable- messaging protocol in
1172 the ebXML *Message Service* be used with SMTP.
1173

1174 7.3.14.3 FTP

1175
1176 FTP is File Transfer Protocol[RFC959].
1177

1178 Since a delivery channel specifies receive characteristics, each *Party* sends a *Message* using FTP
1179 PUT. The endpoint specifies the user id and input directory path (for PUTs to this *Party*). An
1180 example of an FTP endpoint is:

```
1181  
1182     <Endpoint uri="ftp://userid@server.foo.com"  
1183         type = "request"/>  
1184
```

1185 Since FTP must be compatible across all implementations, the FTP for ebXML will use the
1186 minimum sets of commands and parameters available for FTP as specified in [RFC959], section
1187 5.1, and modified in [RFC1123], section 4.1.2.13. The mode SHALL be stream only and the
1188 type MUST be either ASCII Non-print (AN), Image (I) (binary), or Local 8 (L 8) (binary
1189 between 8-bit machines and machines with 36 bit words – for an 8-bit machine Local 8 is the
1190 same as Image).

1191
1192 Stream mode closes the data connection upon end of file. The server side FTP MUST set control
1193 to "PASV" before each transfer command to obtain a unique port pair if there are multiple third
1194 party sessions.

1195
1196 NOTE: [RFC 959] states that User-FTP SHOULD send a PORT command to assign a
1197 non-default data port before each transfer command is issued to allow multiple transfers
1198 during a single FTP because of the long delay after a TCP connection is closed until its
1199 socket pair can be reused.

1200
1201 NOTE: The format of the 227 reply to a PASV command is not well-standardized and an
1202 FTP client may assume that the parentheses indicated in [RFC959] will be present when
1203 in some cases they are not. If the User-FTP program doesn't scan the reply for the first
1204 digit of host and port numbers, the result will be that the User-FTP might point at the
1205 wrong host. In the response, the h1, h2, h3, h4 is the IP address of the server host and the
1206 p1, p2 is a non-default data transfer port that PASV has assigned.

1207
1208 NOTE: As a recommendation for firewall transparency, [RFC1579] proposes that the
1209 client sends a PASV command, allowing the server to do a passive TCP open on some
1210 random port, and inform the client of the port number. The client can then do an active
1211 open to establish the connection.

1212
1213 NOTE: Since STREAM mode closes the data connection upon end of file, the receiving
1214 FTP may assume abnormal disconnect if a 226 or 250 control code hasn't been received
1215 from the sending machine.

1216
1217 NOTE: [RFC1579] also makes the observation that it might be worthwhile to enhance the
1218 FTP protocol to have the client send a new command APSV (all passive) at startup that
1219 would allow a server that implements this option to always perform a passive open. A
1220 new reply code 151 would be issued in response to all file transfer requests not preceded
1221 by a PORT or PASV command; this *Message* would contain the port number to use for
1222 that transfer. A PORT command could still be sent to a server that had previously
1223 received APSV; that would override the default behavior for the next transfer operation,
1224 thus permitting third-party transfers.

1225

1226 7.3.15 Transport Security

1227 The *TransportSecurity* element provides the *Party's* security specifications, associated with the
1228 *ReceivingProtocol* element, for the transport layer of the *CPP*. It MAY be omitted if transport
1229 security will not be used for any *CPAs* composed from this *CPP*. Unless otherwise specified
1230 below, transport security applies to *Messages* in both directions.

1231

1232 Following is the syntax:

1233

```
1234 <TransportSecurity>  
1235     <Protocol version = "3.0">SSL</Protocol>  
1236     <CertificateRef certId = "N03"/> <!--zero or one-->
```

1237 </TransportSecurity>

1238

1239 The *TransportSecurity* element contains two REQUIRED child elements, *Protocol* and
1240 *CertificateRef*.

1241

1242 **7.3.15.1 Protocol element**

1243 The value of the *Protocol* element can identify any transport security protocol that the *Party* is
1244 prepared to support. The IMPLIED *version* attribute identifies the version of the specified
1245 protocol.

1246

1247 The specific security properties depend on the services provided by the identified protocol. For
1248 example, SSL performs certificate-based encryption and certificate-based authentication.

1249

1250 Whether authentication is bidirectional or just from *Message* sender to *Message* recipient
1251 depends on the selected transport-security protocol.

1252

1253 **7.3.15.2 CertificateRef element**

1254 The EMPTY *CertificateRef* element contains an IMPLIED IDREF attribute, *certId* that
1255 identifies the certificate to be used by referring to the *Certificate* element (under *PartyInfo*) that
1256 has the matching ID attribute value. The *CertificateRef* element MUST be present if the
1257 transport-security protocol uses certificates. It MAY be omitted otherwise (e.g. if authentication
1258 is by password).

1259

1260 **7.3.15.3 Specifics for HTTP**

1261 For encryption with HTTP, the protocol is SSL[SSL] (Secure Socket Layer) Version 3.0, which
1262 uses public-key encryption.

1263

1264 **7.4 DocExchange element**

1265 The *DocExchange* element provides information that the *Parties* must agree on regarding
1266 exchange of documents between them. This information includes the messaging service
1267 properties (e.g. ebXML *Message Service*[MSSPEC]).

1268

1269 Following is the structure of the *DocExchange* element of the *CPP*. Subsequent sections
1270 describe each child element in greater detail.

1271

```

1272       <DocExchange docExchangeId = "N06">
1273           <ebXMLBinding version = "0.92">
1274             <MessageEncoding> <!--cardinality 0 or 1-->
1275               ...
1276             </MessageEncoding>
1277             <ReliableMessaging> <!--cardinality 0 or 1-->
1278               ...
1279             </ReliableMessaging>
1280             <NonRepudiation> <!--cardinality 0 or 1-->
1281               ...
1282             </NonRepudiation>
1283             <DigitalEnvelope> <!--cardinality 0 or 1-->
1284               ...
1285             </DigitalEnvelope>
1286             <NamespaceSupported> <!-- 1 or more -->

```

```
1287         ...
1288         </NamespaceSupported>
1289     </ebXMLBinding>
1290 </DocExchange>
```

1291
1292 The *DocExchange* element of the *CPP* defines the properties of the messaging service to be
1293 used with *CPAs* composed from the *CPP*.

1294
1295 The *DocExchange* element is comprised of a single *ebXMLBinding* child element.

1296
1297 NOTE: The document-exchange section can be extended to other messaging services by
1298 adding additional *xxxBinding* elements and their child elements that describe the other
1299 services, where *xxx* is replaced by the name of the additional binding. An example is
1300 *XPBinding*, which might define support for the future XML Protocol specification.

1302 7.4.1 docExchangeId attribute

1303 The *DocExchange* element has a single IMPLIED *docExchangeId* attribute that is an [XML] ID
1304 that provides a unique identifier which MAY be referenced from elsewhere within the *CPP*
1305 document.

1307 7.4.2 ebXMLBinding element

1308 The *ebXMLBinding* element describes properties specific to the ebXML *Message*
1309 Service[MSSPEC] The *ebXMLBinding* element is comprised of the following child elements:

- 1310 • zero or one *MessageEncoding* element which specifies how *Messages* are to be
1311 encoded by the document-exchange layer,
- 1312 • zero or one *ReliableMessaging* element which specifies the characteristics of reliable
1313 messaging,
- 1314 • zero or one *NonRepudiation* element which specifies the requirements for signing the
1315 *Message*,
- 1316 • zero or one *DigitalEnvelope* element which specifies the requirements for encryption
1317 by the digital-envelope[DIGENV] method,
- 1318 • zero or more *NamespaceSupported* elements which identify any namespace
1319 extensions supported by the messaging service implementation.

1321 7.4.3 version attribute

1322 The *ebXMLBinding* element has a single REQUIRED *version* attribute that identifies the
1323 version of the ebXML *Message* Service specification being used.

1325 7.4.4 MessageEncoding element

1326 The *MessageEncoding* element specifies how the *Messages* are to be encoded by the document-
1327 exchange layer for transmission. Encoding choices depend on the properties of the *Message*-
1328 exchange protocol specified by the *ebXMLBinding* element. An example for BASE64[MIME]
1329 is:

1331 <MessageEncoding>BASE64</MessageEncoding>

1332

1333 If the *MessageEncoding* element is omitted, there is no document-exchange encoding.

1334

1335 **7.4.5 ReliableMessaging element**

1336 The *ReliableMessaging* element specifies the properties of reliable ebXML *Message* exchange.

1337 The default that applies if the *ReliableMessaging* element is omitted is "BestEffort". See

1338 Section 7.4.5.1. The following is the element structure:

1339

1340 <ReliableMessaging deliverySemantics="OnceAndOnlyOnce"

1341 idempotency="false"

1342 persistDuration="30S">

1343 <!--The pair of elements Retries, RetryInterval

1344 has cardinality 0 or 1-->

1345 <Retries>5</Retries>

1346 <RetryInterval>60</RetryInterval> <!--time in seconds-->

1347 </ReliableMessaging>

1348

1349 The *ReliableMessaging* element is comprised of the following child elements. The pair of
1350 elements has cardinality 0 or 1. Both must be either present or absent.

- 1351 • a *Retries* element,
- 1352 • a *RetryInterval* element.

1353

1354 The *ReliableMessaging* element has attributes as follows:

- 1355 • a REQUIRED *deliverySemantics* attribute,
- 1356 • a REQUIRED *idempotency* attribute,
- 1357 • a REQUIRED *persistDuration* element.

1358

1359 **7.4.5.1 deliverySemantics attribute**

1360 The *deliverySemantics* attribute of the *ReliableMessaging* element specifies the degree of
1361 reliability of *Message* delivery. This attribute is an enumeration of possible values that include
1362 the following:

- 1363 • "OnceAndOnlyOnce",
- 1364 • "BestEffort".

1365

1366 A value of "OnceAndOnlyOnce" specifies that a *Message* must be delivered exactly once.

1367 "BestEffort" specifies that reliable-messaging semantics are not to be used.

1368

1369 **7.4.5.2 idempotency attribute**

1370 The *idempotency* attribute of the *ReliableMessaging* element specifies whether the *Party*
1371 requires that all *Messages* exchanged be subject to an idempotency test (detection and discard of
1372 duplicate *Messages*) in the document-exchange layer. The attribute is a Boolean with possible
1373 values of "true" and "false". If the value of the attribute is "true", all *Messages* are subject to the
1374 test. If the value is "false", *Messages* are not subject to an idempotency test in the document-
1375 exchange layer. Testing for duplicates is based on the *Message* identifier; other information that
1376 is carried in the *Message Header* MAY also be tested, depending on the context.

1377

1378 NOTE: Additional testing for duplicates MAY take place in the business application based
1379 on application information in the *Messages* (e.g. purchase order number).

1380
1381 The idempotency test checks whether a *Message* duplicates a prior *Message* between the same
1382 client and server. If the idempotency test is requested, the receiving messaging service passes a
1383 duplicate *Message* to the recipient *Business Process* with a "duplicate" indication. The receiving
1384 messaging service also returns a "duplicate" indication to the sender of the duplicate.

1385
1386 NOTE: One of the main purposes of this test is to aid in retry following timeouts and in
1387 recovery following node failures. In these cases, the sending *Party* might have sent
1388 request *Messages* and not received responses. The sending *Party* MAY re-send such a
1389 *Message*. If the original *Message* had been received, the receiving server discards the
1390 duplicate *Message* and re-sends the original results to the requester.

1391
1392 If a communication protocol always checks for duplicate *Messages*, the check in the
1393 communication protocol overrides any idempotency specifications in the *CPA*.

1394

1395 **7.4.5.3 persistDuration attribute**

1396 The value of the *persistDuration* attribute is the minimum length of time, expressed as a
1397 [XMLSchema] timeDuration, that data from a *Message* that is sent reliably is kept in *Persistent*
1398 *Storage* by an ebXML *Message-Service* implementation that receives that *Message*.

1399

1400 **7.4.5.4 Retries and RetryInterval elements**

1401 The *Retries* and *RetryInterval* elements specify the permitted number of retries and interval
1402 between retries (in seconds) of a request following a timeout. The purpose of the *RetryInterval*
1403 element is to improve the likelihood of success on retry by deferring the retry until any
1404 temporary conditions that caused the error might be corrected.

1405

1406 The *Retries* and *RetryInterval* elements MUST be included together or MAY be omitted
1407 together. If they are omitted, the values of the corresponding quantities (number of retries and
1408 retry interval) are a local matter at each *Party*.

1409

1410 **7.4.6 NonRepudiation element**

1411 Non-repudiation both proves who sent a *Message* and prevents later repudiation of the contents
1412 of the *Message*. Non-repudiation is based on signing the *Message* using XML Digital
1413 Signature[XMLDSIG]. The element structure is as follows:

```
1414 <NonRepudiation>  
1415     <Protocol version = "1.0">XMLDSIG</Protocol>  
1416     <HashFunction>sha1</HashFunction>  
1417     <SignatureAlgorithm>rsa</SignatureAlgorithm>  
1418     <CertificateRef certId = "N03"/>  
1419 </NonRepudiation>
```

1420

1421 If the *NonRepudiation* element is omitted, the *Messages* are not digitally signed.

1422

1423 Security at the document-exchange level applies to all *Messages* in both directions for *Business*
1424 *Transactions* for which security is enabled.

1425

1426 The *NonRepudiation* element is comprised of the following child elements:

1427 **Collaboration-Protocol Profile and Agreement Specification**

- 1428 • a REQUIRED *Protocol* element,
- 1429 • a REQUIRED *HashFunction* (e.g. SHA1, MD5) element,
- 1430 • a REQUIRED *SignatureAlgorithm* element,
- 1431 • a REQUIRED *Certificate* element.

1432

1433 7.4.6.1 Protocol element

1434 The REQUIRED *Protocol* element identifies the technology that will be used to digitally sign a
1435 *Message*. It has a single IMPLIED *version* attribute whose value is a string that identifies the
1436 version of the specified technology. An example of the *Protocol* element follows:

1437

```
1438     <Protocol version="2000/10/31">http://www.w3.org/2000/09/xmlldsig#  
1439     </Protocol>
```

1440

1441 7.4.6.2 HashFunction element

1442 The REQUIRED *HashFunction* element identifies the algorithm that is used to compute the
1443 digest of the *Message* being signed.

1444

1445 7.4.6.3 SignatureAlgorithm element

1446 The REQUIRED *SignatureAlgorithm* element identifies the algorithm that is used to compute
1447 the value of the digital signature.

1448

1449 7.4.6.4 CertificateRef element

1450 The REQUIRED *CertificateRef* element refers to one of the *Certificate* elements elsewhere
1451 within the *CPP* document, using the IMPLIED *certId* IDREF attribute.

1452

1453 7.4.7 DigitalEnvelope element

1454 The *DigitalEnvelope* element[DIGENV] is an encryption procedure in which the *Message* is
1455 encrypted by symmetric encryption (shared secret key) and the secret key is sent to the *Message*
1456 recipient encrypted with the recipient's public key. The element structure is:

1457

```
1458 <DigitalEnvelope>  
1459     <Protocol version = "2.0">S/MIME</Protocol>  
1460     <EncryptionAlgorithm>rsa</EncryptionAlgorithm>  
1461     <CertificateRef certId = "N03"/>  
1462 </DigitalEnvelope>
```

1463

1464 Security at the document-exchange level applies to all *Messages* in both directions for *Business*
1465 *Transactions* for which security is enabled.

1466

1467 7.4.7.1 Protocol element

1468 The REQUIRED *Protocol* element identifies the security protocol to be used. The FIXED
1469 *version* attribute identifies the version of the protocol.

1470

1471 7.4.7.2 EncryptionAlgorithm element

1472 The REQUIRED *EncryptionAlgorithm* element identifies the encryption algorithm to be used.

1473

1474 7.4.7.3 CertificateRef element

1475 The REQUIRED *CertificateRef* element identifies the certificate to be used by means of its

1476 **certId** attribute. The IMPLIED **certId** attribute is an attribute of type [XML] IDREF, which
1477 refers to a matching ID attribute in a **Certificate** element elsewhere in the **CPP** or **CPA**.
1478

1479 **7.4.8 NamespaceSupported element**

1480 The **NamespaceSupported** element identifies any namespace extensions supported by the
1481 messaging service implementation. Examples are Security Services Markup Language[S2ML]
1482 and Transaction Authority Markup Language[XAML]. For example, support for the S2ML
1483 namespace would be defined as follows:

```
1484  
1485     <NamespaceSupported schemaLocation = "http://www.s2ml.org/s2ml.xsd"  
1486     version = "0.8">http://www.s2ml.org/s2ml</NamespaceSupported>
```

1487

1488 **7.5 ds:Signature element**

1489 The **CPP** MAY be digitally signed using technology that conforms with the XML Digital
1490 Signature specification[XMLDSIG]. The **ds:Signature** element is the root of a subtree of
1491 elements that MAY be used for signing the **CPP**. The syntax is:

```
1492  
1493     <ds:Signature>...</ds:Signature>
```

1494

1495 The content of this element and any subelements are defined by the XML Digital Signature
1496 specification. See Section 8.8 for a detailed discussion. The following additional constraints on
1497 **ds:Signature** are imposed:

1498

- 1499 • A **CPP** MUST be considered invalid if any **ds:Signature** element fails core validation as
1500 defined by the XML Digital Signature specification[XMLDSIG].
1501
- 1502 • Whenever a **CPP** is signed, each **ds:Reference** element within a **ProcessSpecification**
1503 element MUST pass reference validation and each **ds:Signature** element MUST pass
1504 core validation.
1505

1506 NOTE: In case a **CPP** is unsigned, software MAY nonetheless validate the **ds:Reference**
1507 elements within **ProcessSpecification** elements and report any exceptions.
1508

1509 NOTE: Software for creation of **CPPs** and **CPAs** MAY recognize **ds:Signature** and
1510 automatically insert the element structure necessary to define signing of the **CPP** and **CPA**.
1511 Signature creation itself is a cryptographic process that is outside the scope of this
1512 specification.
1513

1514 NOTE: see non-normative note in Section 7.3.4.5 for a discussion of times at which validity
1515 tests MAY be made.
1516

1517 **7.6 Comment element**

1518 The **CollaborationProtocolProfile** element MAY contain zero or more **Comment** elements. The
1519 **Comment** element is a textual note that MAY be added to serve any purpose the author desires.
1520 The language of the **Comment** is identified by a REQUIRED **xml:lang** attribute. The **xml:lang**

1521 attribute **MUST** comply with the rules for identifying languages specified in [XML]. If multiple
1522 **Comment** elements are present, each **SHOULD** have a unique *xml:lang* attribute value. An
1523 example of a **Comment** element follows:

1524

```
1525     <Comment xml:lang="en-gb">yadda yadda, blah blah</Comment>
```

1526

1527 When a *CPA* is composed from two *CPPs*, all **Comment** elements from both *CPPs* **SHALL** be
1528 included in the *CPA* unless the two *Parties* agree otherwise.

1529 8 CPA Definition

1530 A *Collaboration-Protocol Agreement (CPA)* defines the capabilities that two *Parties* must agree
 1531 to enable them to engage in electronic business for the purposes of the particular *CPA*. This
 1532 section defines and discusses the details of the *CPA*. The discussion is illustrated with some
 1533 XML fragments.

1534
 1535 Most of the XML elements in this section are described in detail in section 7, "*CPP Definition*".
 1536 In general, this section does not repeat that information. The discussions in this section are
 1537 limited to those elements that are not in the *CPP* or for which additional discussion is required in
 1538 the *CPA* context. See also Appendix C and Appendix D for the DTD and XML Schema,
 1539 respectively, and Appendix B for an example of a *CPA* document.

1540

1541 8.1 CPA Structure

```

1542
1543 <CollaborationProtocolAgreement id = "N01"
1544     xmlns="http://www.ebxml.org/namespaces/tradePartner"
1545     xmlns:bpm="http://www.ebxml.org/namespaces/businessProcess"
1546     xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
1547     xmlns:xlink = "http://www.w3.org/1999/xlink">
1548     <CPAType> <!--MAY appear 0 or 1 times-->
1549         ...
1550     </CPAType>
1551     <Status value = "proposed"/>
1552     <Start>1988-04-07T18:39:09</Start>
1553     <End>1990-04-07T18:40:00</End>
1554     <!--ConversationConstraints MAY appear 0 or 1 times-->
1555     <ConversationConstraints invocationLimit = "100"
1556         concurrentConversations = "4"/>
1557     <PartyInfo>
1558         ...
1559     </PartyInfo>
1560     <PartyInfo>
1561         ...
1562     </PartyInfo>
1563     <!--ds:signature MAY appear 0 or more times-->
1564     <ds:Signature>any combination of text and elements
1565     </ds:Signature>
1566     <Comment xml:lang="en-gb">any text</Comment> <!--zero or more-->
1567 </CollaborationProtocolAgreement>
1568

```

1569 8.2 CollaborationProtocolAgreement element

1570 The *CollaborationProtocolAgreement* element is the root element of a *CPA*. It has a
 1571 REQUIRED *id* attribute of type [XML] CDATA that supplies a unique identifier for the
 1572 document. The value of the *id* attribute SHALL be assigned by one *Party* and used by both. It is
 1573 RECOMMENDED that the value of the *id* attribute be a URI. The value of the *id* attribute MAY
 1574 be used as the value of the *CPAId* element in the ebXML *Message Header*[MSSPEC].

1575

1576 NOTE: Each *Party* MAY associate a local identifier with the *id* attribute.

1577

1578 The *CollaborationProtocolAgreement* element has REQUIRED [XML] Namespace[XMLNS]
1579 declarations that are defined in Section 7, "CPP Definition".

1580

1581 The *CollaborationProtocolAgreement* element is comprised of the following child elements,
1582 each of which is described in greater detail in subsequent sections:

- 1583 • zero or one *CPAType* element that provides information about the general nature of
1584 the *CPA*,
- 1585 • a REQUIRED *Status* element that identifies the state of the process that creates the
1586 *CPA*,
- 1587 • a REQUIRED *Start* element that records the date and time that the *CPA* goes into
1588 effect,
- 1589 • a REQUIRED *End* element that records the date and time after which the *CPA* must
1590 be renegotiated by the *Parties*,
- 1591 • zero or one *ConversationConstraints* element that documents certain agreements
1592 about conversation processing,
- 1593 • two REQUIRED *PartyInfo* elements, one for each *Party* to the *CPA*,
- 1594 • one or more *ds:Signature* elements that provide signing of the *CPA* using the XML
1595 Digital Signature[XMLDSIG] standard.

1596

1597

8.3 CPAType element

1598

1599 The *CPAType* element MAY be present in a *CPA* document. It provides information about the
1600 general nature of the *CPA*. An example of this element follows:

1601

```
1602 <CPAType>  
1603     <Protocol version = "1.1">PIP3A4</Protocol>  
1604     <Type>RNIF</Type>  
1605 </CPAType>
```

1606

1607 The *CPAType* element is comprised of the following child elements:

- 1608 • a REQUIRED *Protocol* element identifies the business-level protocol. An example is
1609 PIP3A4, a RosettaNet™ Partner Interface Process.
- 1610 • a REQUIRED *Type* element provides additional information about the *Business*
1611 *Protocol*. Specific values depend on the particular protocol and its optional features.
1612 An example is RNIF (RosettaNet Implementation Framework).

1613

1614 The *Protocol* element has a REQUIRED attribute, *version*, whose value specifies the version of
1615 the protocol that is to be used.

1616

1617 NOTE: An implementation MAY use the *CPAType* element to determine whether it
1618 already has the code to support this particular protocol.

1619

1620 8.4 Status element

1621
1622 The *Status* element records the state of the composition/negotiation process that creates the *CPA*.
1623 An example of the *Status* element follows:

```
1624  
1625     <Status value = "proposed"/>
```

1626
1627 The *Status* element has a REQUIRED *value* attribute that records the current state of
1628 composition of the *CPA*. The value of this attribute is an enumeration of the following possible
1629 values:

- 1630 • "proposed", meaning that the *CPA* is still being negotiated by the *Parties*,
- 1631 • "agreed", meaning that the contents of the *CPA* have been agreed to by both *Parties*,
- 1632 • "signed", meaning that the *CPA* has been "signed" by the *Parties*. This "signing"
1633 MAY take the form of a digital signature that is described in section 8.8 below.

1634
1635 NOTE: The *Status* element MAY be used by a *CPA* composition and negotiation tool to
1636 assist in the process of building a *CPA*.

1637

1638 8.5 CPA Lifetime

1639 The lifetime of the *CPA* is given by the *Start* and *End* elements. The syntax is:

```
1640  
1641     <Start>1988-04-07T18:39:09</Start>  
1642     <End>1990-04-07T18:40:00</End>
```

1643

1644 8.5.1 Start element

1645 The *Start* element specifies the starting date and time of the *CPA*. The *Start* element SHALL be
1646 a string value that conforms to the content model of a canonical timeInstant as defined in the
1647 XML Schema Datatypes Specification[XMLSCHEMA-2]. For example, to indicate 1:20 pm
1648 UTC (Coordinated Universal Time) on May 31, 1999, a *Start* element would have the following
1649 value:

```
1650  
1651     1999-05-31T13:20:00Z
```

1652
1653 The *Start* element SHALL be represented as Coordinated Universal Time (UTC).

1654

1655 8.5.2 End element

1656 The *End* element specifies the ending date and time of the *CPA*. The *End* element SHALL be a
1657 string value that conforms to the content model of a canonical timeInstant as defined in the XML
1658 Schema Datatypes Specification[XMLSCHEMA-2]. For example, to indicate 1:20 pm UTC
1659 (Coordinated Universal Time) on May 31, 1999, an *End* element would have the following
1660 value:

```
1661  
1662     1999-05-31T13:20:00Z
```

1663

1664 The **End** element SHALL be represented as Coordinated Universal Time (UTC).

1665
1666 When the end of the *CPA's* lifetime is reached, any *Business Transactions* that are still in
1667 progress SHALL be allowed to complete and no new *Business Transactions* SHALL be started.
1668 When all in-progress *Business Transactions* on each conversation are completed, the
1669 *Conversation* shall be terminated whether or not it was completed.

1670
1671 NOTE: It should be understood that if a business application defines a conversation as
1672 consisting of multiple *Business Transactions*, such a conversation MAY be terminated
1673 with no error indication when the end of the lifetime is reached. The run-time system
1674 could provide an error indication to the application.

1675
1676 NOTE: It should be understood that it MAY not be feasible to wait for outstanding
1677 conversations to terminate before ending the *CPA* since there is no limit on how long a
1678 conversation MAY last.

1679
1680 NOTE: The runtime system SHOULD return an error indication to both *Parties* when a
1681 new *Business Transaction* is started under this *CPA* after the date and time specified in
1682 the **End** element.

1683

1684 8.6 ConversationConstraints element

1685

1686 The *ConversationConstraints* element places limits on the number of conversations under the
1687 *CPA*. An example of this element follows:

1688
1689 `<ConversationConstraints invocationLimit = "100"`
1690 `concurrentConversations = "4"/>`

1691

1692 The *ConversationConstraints* element has the following attributes:

- 1693 • an IMPLIED *invocationLimit* attribute,
- 1694 • an IMPLIED *concurrentConversations* attribute.

1695

1696 8.6.1 invocationLimit attribute

1697 The *invocationLimit* attribute defines the maximum number of conversations that can be
1698 processed under the *CPA*. When this number has been reached, the *CPA* is terminated and must
1699 be renegotiated. If no value is specified, there is no upper limit on the number of conversations
1700 and the lifetime of the *CPA* is controlled solely by the **End** element.

1701

1702 NOTE: The *invocationLimit* attribute sets a limit on the number of units of *Business* that
1703 can be performed under the *CPA*. It is a business parameter, not a performance parameter.

1704

1705 8.6.2 concurrentConversations attribute

1706 The *concurrentConversations* attribute defines the maximum number of conversations that can
1707 be in process under this *CPA* at the same time. If no value is specified, processing of concurrent

1708 conversations is strictly a local matter.

1709
1710 NOTE: The *concurrentConversations* attribute provides a parameter for the *Parties* to use
1711 when it is necessary to limit the number of conversations that can be concurrently processed
1712 under a particular *CPA*. For example, the back-end process might only support a limited
1713 number of concurrent conversations. If a request for a new conversation is received when
1714 the maximum number of conversations allowed under this *CPA* is already in process, an
1715 implementation MAY reject the new conversation or MAY enqueue the request until an
1716 existing conversation ends. If no value is given for *concurrentConversations*, how to handle
1717 a request for a new conversation for which there is no capacity is a local implementation
1718 matter.

1719

1720 8.7 PartyInfo element

1721 The general characteristics of the *PartyInfo* element are discussed in sections 7.3 and 7.3.1 .

1722

1723 The *CPA* SHALL have one *PartyInfo* element for each *Party* to the *CPA*. The *PartyInfo*
1724 element specifies the *Parties'* agreed terms for engaging in a *the Business Processes* defined by
1725 the *Process-Specification* documents *referenced by the CPA*. If a *CPP* has more than one
1726 *PartyInfo* element, the appropriate *PartyInfo* element SHALL be selected from each *CPP* when
1727 composing a *CPA*.

1728

1729 In the *CPA*, there SHALL be one *PartyId* element under each *PartyInfo* element. The value of
1730 this element is the same as the value of the *PartyId* element in the ebXML *Message Service*
1731 specification[MSSPEC]. One *PartyId* element SHALL be used within a *To* or *From Header*
1732 element of an ebXML *Message*.

1733

1734 8.7.1 ProcessSpecification element

1735 The *ProcessSpecification* element identifies the *Business Process* that the two *Parties* have
1736 agreed to perform. There MAY be one or more *ProcessSpecification* elements in a *CPA*. Each
1737 SHALL be a child element of a separate *CollaborationRole* element. See the discussion in
1738 Section 7.3.3.

1739

1740 8.8 ds:Signature element

1741 A *CPA* document MAY be digitally signed by one or more of the *Parties* as a means of ensuring
1742 its integrity as well as a means of expressing the agreement just as a corporate officer's signature
1743 would do for a paper document. If signatures are being used to digitally sign an ebXML *CPA* or
1744 *CPP* document, then it is strongly RECOMMENDED that [XMLDSIG] be used to digitally sign
1745 the document. The *ds:Signature* element is the root of a subtree of elements that MAY be used
1746 for signing the *CPP*. The syntax is:

1747

```
1748 <ds:Signature>...</ds:Signature>
```

1749

1750 The content of this element and any subelements are defined by the XML Digital Signature
1751 specification[XMLDSIG]. The following additional constraints on *ds:Signature* are imposed:

1752
1753 • A *CPA* MUST be considered invalid if any *ds:Signature* fails core validation as defined
1754 by the XML Digital Signature specification.

1755
1756 • Whenever a *CPA* is signed, each *ds:Reference* within a *ProcessSpecification* MUST
1757 pass reference validation and each *ds:Signature* MUST pass core validation.

1758
1759 NOTE: In case a *CPA* is unsigned, software MAY nonetheless validate the *ds:Reference*
1760 elements within *ProcessSpecification* elements and report any exceptions.

1761
1762 NOTE: Software for creation of *CPPs* and *CPAs* MAY recognize *ds:Signature* and
1763 automatically insert the element structure necessary to define signing of the *CPP* and *CPA*.
1764 Signature creation itself is a cryptographic process that is outside the scope of this
1765 specification.

1766
1767 NOTE: See non-normative note in section 7.3.4.5 for a discussion of times at which a *CPA*
1768 MAY be validated.

1769

1770 **8.8.1 Persistent Digital Signature**

1771 If [XMLDSIG] is used to sign an ebXML *CPP* or *CPA*, the process defined in this section of the
1772 specification SHALL be used.

1773

1774 **8.8.1.1 Signature Generation**

1775 1) Create a *SignedInfo* element, a child element of *ds:signature*. *SignedInfo* SHALL have child
1776 elements *SignatureMethod*, *CanonicalizationMethod*, and *Reference* as prescribed by
1777 [XMLDSIG].

1778 2) Canonicalize and then calculate the **SignatureValue** over *SignedInfo* based on algorithms
1779 specified in *SignedInfo* as specified in [XMLDSIG].

1780 3) Construct the *Signature* element that includes the *SignedInfo*, *KeyInfo* (RECOMMENDED),
1781 and *SignatureValue* elements as specified in [XMLDSIG].

1782 4) Include the namespace qualified *Signature* element in the document just signed, following the
1783 last *PartyInfo* element.

1784

1785 **8.8.1.2 ds:SignedInfo element**

1786 The *ds:SignedInfo* element SHALL be comprised of zero or one *ds:CanonicalizationMethod*
1787 element, the *ds:SignatureMethod* element, and one or more *ds:Reference* elements.

1788

1789 **8.8.1.3 ds:CanonicalizationMethod element**

1790 The *ds:CanonicalizationMethod* element is defined as OPTIONAL in [XMLDSIG], meaning
1791 that the element need not appear in an instance of a *ds:SignedInfo* element. The default
1792 canonicalization method that is applied to the data to be signed is [XMLC14N] in the absence of
1793 a *ds:CanonicalizationMethod* element that specifies otherwise. This default SHALL also serve
1794 as the default canonicalization method for the ebXML *CPP* and *CPA* documents.

1795

8.8.1.4 ds:SignatureMethod element

1796 The *ds:SignatureMethod* element SHALL be present and SHALL have an *Algorithm* attribute.
1797 The RECOMMENDED value for the *Algorithm* attribute is:

1799

1800 `http://www.w3.org/2000/02/xmlsig#sha1`

1801

1802 This RECOMMENDED value SHALL be supported by all compliant ebXML *CPP* or *CPA*
1803 software implementations.

1804

8.8.1.5 ds:Reference element

1806 The *ds:Reference* element for the *CPP* or *CPA* document SHALL have a REQUIRED URI
1807 attribute value of "" to provide for the signature to be applied to the document that contains the
1808 *ds:Signature* element (the *CPA* or *CPP* document). The *ds:Reference* element for the *CPP* or
1809 *CPA* document MAY include an IMPLIED *type* attribute that has a value of:

1810

1811 `"http://www.w3.org/2000/02/xmlsig#Object"`

1812

1813 in accordance with [XMLDSIG]. This attribute is purely informative. It MAY be omitted.
1814 Implementations of software designed to author or process an ebXML *CPA* or *CPP* document
1815 SHALL be prepared to handle either case. The *ds:Reference* element MAY include the *id*
1816 attribute, type ID, by which this *ds:Reference* element MAY be referenced from a *ds:Signature*
1817 element.

1818

8.8.1.6 ds:Transform element

1820 The *ds:Reference* element for the *CPA* or *CPP* document SHALL include a child *ds:Transform*
1821 element that excludes the containing *ds:Signature* element and all its descendants.

1822

8.8.1.7 ds:Xpath element

1824 The *ds:Transform* element SHALL include a child *ds:XPath* element that has a value of:

1825

1826 `/descendant-or-self::node()[not(ancestor-or-self::ds:Signature[@id='S1'])]`

1827

1828 NOTE: When digitally signing a *CPA*, it is RECOMMENDED that each *Party* sign the
1829 document in accordance with the process described above. The first *Party* that signs the
1830 *CPA* will sign only the *CPA* contents, excluding their own signature. The second *party*
1831 signs over the contents of the *CPA* as well as the *ds:Signature* element that contains the
1832 first *Party's* signature. It MAY be necessary that a notary sign over both signatures so as to
1833 provide for cryptographic closure.

1834

8.9 Comment element

1836 The *CollaborationProtocolAgreement* element MAY contain zero or more *Comment* elements.
1837 See section 7.6 for details of the syntax of the *Comment* element.

1838

1839 **8.10 Composing a CPA from Two CPPs**

1840 This section discusses normative issues in composing a *CPA* from two *CPPs*. See also Appendix
1841 F , "Composing a *CPA* from Two *CPPs* (Non-Normative)".
1842

1843 **8.10.1 ID Attribute Duplication**

1844 In composing a *CPA* from two *CPPs*, there is a hazard that ID attributes from the two *CPPs*
1845 might have duplicate values. When a *CPA* is composed from two *CPPs*, duplicate ID attribute
1846 values SHALL be tested for. If a duplicate ID attribute value is present, one of the duplicates
1847 shall be given a new value and the corresponding IDREF attribute values from the corresponding
1848 *CPP* SHALL be corrected.
1849

1850 **8.11 Modifying Parameters of the Process-Specification Document Based on** 1851 **Information in the CPA**

1852 A *Process-Specification* document contains a number of parameters, expressed as XML
1853 attributes. An example is the security attributes that are counterparts of the attributes of the *CPA*
1854 *Characteristics* element. The values of these attributes can be considered to be default values or
1855 recommendations. When a *CPA* is created, the *Parties* MAY decide to accept the
1856 recommendations in the *Process-Specification* or they MAY agree on values of these parameters
1857 that better reflect their needs.
1858

1859 When a *CPA* is used to configure a run-time system, choices specified in the *CPA* MUST always
1860 assume precedence over choices specified in the referenced *Process-Specification* document. In
1861 particular, all choices expressed in a *CPA*'s *Characteristics* and *Packaging* elements MUST be
1862 implemented as agreed to by the *Parties*. These choices SHALL override the default values
1863 expressed in the *Process-Specification* document. The process of installing the information from
1864 the *CPA* and *Process-Specification* document MUST verify that all of the resulting choices are
1865 mutually consistent and MUST signal an error if they are not.
1866

1867 NOTE: There are several ways of overriding the information in the *Process-*
1868 *Specification* document by information from the *CPA*. For example:
1869

- 1870 • A separate copy of the *Process-Specification* document can be created by the *CPA*
1871 composition tool. The tool can then directly modify the *Process-Specification*
1872 document with information from the *CPA*. One advantage of this method is that the
1873 override process is performed entirely by the *CPA* composition tool. A second
1874 advantage is that with a separate copy of the *Process-Specification* document
1875 associated with the particular *CPA*, there is no exposure to modifications of the
1876 *Process-Specification* document between the time that the *CPA* is created and the
1877 time it is installed in the *Parties*' systems.
- 1878 • A *CPA* installation tool can dynamically override parameters in the *Process-*
1879 *Specification* document with information from the corresponding parameters from the
1880 *CPA* at the time the *CPA* and *Process-Specification* document are installed in the
1881 *Parties*' systems. This eliminates the need to create a separate copy of the *Process-*
1882 *Specification* document.

- 1883
- 1884
- 1885
- Other possible methods might be based on XSLT transformations of the parameter information in the *CPA* and/or the *Process-Specification* document.

1886

9 References

1887 Some references listed below specify functions for which specific XML definitions are provided
1888 in the *CPP* and *CPA*. Other specifications are referred to in this specification in the sense that
1889 they are represented by keywords for which the *Parties* to the *CPA* MAY obtain plug-ins or
1890 write custom support software but do not require specific XML element sets in the *CPP* and
1891 *CPA*.

1892
1893 In a few cases, the only available specification for a function is a proprietary specification.
1894 These are indicated by notes within the citations below.

1895
1896

1897 [BPMSPEC] ebXML Business Process Specification Schema specification,
1898 <http://www.ebxml.org>.

1899
1900 [DIGENV] Digital Envelope, RSA Laboratories, <http://www.rsasecurity.com/rsalabs/>. NOTE:
1901 At this time, the only available specification for digital envelope appears to be the RSA
1902 Laboratories specification.

1903

1904 [EBXMLCC] ebXML Core Components and Business Process Document Overview,
1905 <http://www.ebxml.org>.

1906

1907 [EBXMLGLOSS] ebXML Glossary, <http://www.ebxml.org>.

1908

1909 [HTMLENC] HTML ver. 4.0 specification, World Wide Web Consortium,
1910 <http://www.w3.org/TR/html4/>. See section 5.3, Character References.

1911

1912 [HTTP] Hypertext Transfer Protocol, Internet Engineering Task Force RFC2616.

1913

1914 [IPSEC] IP Security Document Roadmap, Internet Engineering Task Force RFC 2411.

1915

1916 [ISO6523] Structure for the Identification of Organizations and Organization Parts, International
1917 Standards Organization ISO-6523.

1918

1919 [MIME] MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying
1920 and Describing the Format of Internet *Message* Bodies. Internet Engineering Task Force RFC
1921 1521.

1922

1923 [MSSPEC] ebXML Message Service Specification, <http://www.ebxml.org>

1924

1925 [REGREP] ebXML Registry and Repository Specification, <http://www.ebxml.org>

1926

1927 [RFC822] Standard for the Format of ARPA Internet Text Messages, Internet Engineering Task
1928 Force RFC 822.

1929

- 1930 [RFC959] File Transfer Protocol (FTP), Internet Engineering Task Force RFC 959.
1931
1932 [RFC1123] Requirements for Internet Hosts -- Application and Support, R. Braden, Internet
1933 Engineering Task Force, October, 1989.
1934
1935 [RFC1579] Firewall-Friendly FTP, S. Bellovin, Internet Engineering Task Force, February,
1936 1994.
1937
1938 [RFC2015] MIME Security with Pretty Good Privacy, M. Elkins, Internet Engineering Task
1939 Force, RFC 2015.
1940
1941 [RFC2119] Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering
1942 Task Force RFC 2119.
1943
1944 [RFC2396] Uniform Resource Identifiers (URI): Generic Syntax; T. Berners-Lee, R. Fielding, L.
1945 Masinter - August 1998
1946
1947 [S/MIME] S/MIME Version 3 Message Specification, Internet Engineering Task Force RFC
1948 2633.
1949
1950 [S2ML] Security Services Markup Language, <http://s2ml.org/>
1951
1952 [SMTP] Simple Mail Transfer Protocol, Internet Engineering Task Force RFC 821.
1953
1954 [SSL] Secure Sockets Layer, Netscape Communications Corp. <http://developer.netscape.com>.
1955 NOTE: At this time, it appears that the Netscape specification is the only available specification
1956 of SSL. Work is in progress in IETF on "Transport Layer Security", which is intended as a
1957 replacement for SSL.
1958
1959 [TECHARCH] ebXML Technical Architecture Specification, <http://www.ebxml.org>.
1960
1961 [XAML] Transaction Authority Markup Language, <http://xaml.org/>
1962
1963 [XLINK] XML Linking Language, <http://www.w3.org/TR/xlink/>
1964
1965 [XML] Extensible Markup Language (XML), World Wide Web Consortium,
1966 <http://www.w3.org>.
1967
1968 [XMLC14N] Canonical XML, Ver. 1.0, <http://www.w3.org/TR/XML-C14N/>
1969
1970 [XMLDSIG] XML Signature Syntax and Processing, Worldwide Web Consortium,
1971 <http://www.w3.org/TR/xmlsig-core/>
1972
1973 [XMLNS] Namespaces in XML, T. Bray, D. Hollander, and A. Layman, Jan. 1999,
1974 <http://www.w3.org/TR/REC-xml-names/>.
1975

- 1976 [XMLSCHEMA-1] XML Schema Part 1: Structures, <http://www.w3.org/TR/xmlschema-1/>
1977
1978 [XMLSCHEMA-2] XML Schema Part 2: Datatypes,
1979 <http://www.w3.org/TR/xmlschema-2/>
1980
1981 [XPOINTER] XML Pointer Language, ver. 1.0, <http://www.w3.org/TR/xptr>

1982 10 Conformance

1983 In order to conform to this specification, an implementation:

- 1984 a) SHALL support all the functional and interface requirements defined in this specification,
1985 b) SHALL NOT specify any requirements that would contradict or cause non-conformance
1986 to this specification.

1987

1988 A conforming implementation SHALL satisfy the conformance requirements of the applicable
1989 parts of this specification,

1990

1991 An implementation of a tool or service that creates or maintains ebXML *CPP* or *CPA* instance
1992 documents SHALL be determined to be conformant by validation of the *CPP* or *CPA* instance
1993 documents, created or modified by said tool or service, against the [XMLSCHEMA] definition of
1994 the *CPP* or *CPA* in Appendix D and available from

1995

1996 http://www.ebxml.org/schemas/cpp-cpa-v1_0.xsd

1997

1998 by using two or more validating XML Schema parsers that conform to the W3C XML Schema
1999 specifications[XMLSCHEMA-1,XMLSCHEMA-2].

2000

2001 The objective of conformance testing is to determine whether an implementation being tested
2002 conforms to the requirements stated in this specification. Conformance testing enables vendors to
2003 implement compatible and interoperable systems. Implementations and applications SHALL be
2004 tested using available test suites to verify their conformance to this specification.

2005

2006 Publicly available test suites from vendor neutral organizations such as OASIS and the U.S.A.
2007 National Institute of Science and Technology (NIST) SHOULD be used to verify the
2008 conformance of implementations, applications, and components claiming conformance to this
2009 specification. Open-source reference implementations MAY be available to allow vendors to test
2010 their products for interface compatibility, conformance, and interoperability.

2011

2012

2013

2014

2015 11 Disclaimer

2016 The views and specification expressed in this document are those of the authors and are not
2017 necessarily those of their employers. The authors and their employers specifically disclaim
2018 responsibility for any problems arising from correct or incorrect implementation or use of this
2019 design.

2020 Contact Information

2021

2022

2023 Martin W. Sachs (Team Leader)

2024 IBM T. J. Watson Research Center

2025 P.O.B. 704

2026 Yorktown Hts, NY 10598

2027 USA

2028 Phone: 914-784-7287

2029 email: mwsachs@us.ibm.com

2030

2031

2032 Chris Ferris

2033 XML Technology Development

2034 Sun Microsystems, Inc

2035 One Network Drive

2036 Burlington, Ma 01824-0903

2037 USA

2038 781-442-3063

2039 email: chris.ferris@east.sun.com

2040

2041

2042 Dale W. Moberg

2043 Sterling Commerce

2044 4600 Lakehurst Ct.

2045 Dublin, OH 43016

2046 USA

2047 Phone: 614-793-5015

2048 email: dale_moberg@stercomm.com

2049

2050 Tony Weida

2051 Edifecs

2052 2310 130th Ave. NE, Suite 100

2053 Bellevue, WA 98005

2054 USA

2055 Phone: 212-678-5265

2056 Email: TonyW@edifecs.com

2057 Copyright Statement

2058

2059 Copyright © ebXML 2001. All Rights Reserved.

2060

2061 This document and translations of it MAY be copied and furnished to others, and derivative
2062 works that comment on or otherwise explain it or assist in its implementation MAY be prepared,
2063 copied, published and distributed, in whole or in part, without restriction of any kind, provided
2064 that the above copyright notice and this paragraph are included on all such copies and derivative
2065 works. However, this document itself MAY not be modified in any way, such as by removing
2066 the copyright notice or references to ebXML, UN/CEFACT, or OASIS, except as required to
2067 translate it into languages other than English.

2068

2069 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
2070 successors or assigns.

2071

2072 This document and the information contained herein is provided on an "AS IS" basis and
2073 ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
2074 NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN
2075 WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
2076 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

2077

2078 **Appendix A** Example of *CPP* Document (Non-Normative)

2079 **This example is out of date and will be replaced along with the response to the next round**
 2080 **of public-review comments.**

```

2081
2082 <?xml version = "1.0"?>
2083 <!DOCTYPE CollaborationProtocolProfile SYSTEM "cppml%2cv0.23.dtd">
2084 <!--Generated by XML Authority.-->
2085 <CollaborationProtocolProfile id = "id"
2086     xmlns="http://www.ebxml.org/namespaces/tradePartner"
2087     xmlns:bpm = "http://www /namespaces/businessProcess"
2088     xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
2089     xmlns:xlink = "http://www.w3.org/1999/xlink">
2090     <!--(Party , (CollaborationProtocol | bpm:ProcessSpecification |
2091 bpm:BinaryCollaboration | bpm:BusinessTransactionActivity)+ , ds:Signature?)-->
2092     <Party partyId = "N01">
2093         <!--(PartyId+ , PartyDetails , Role+ , Certificate+ , DeliveryChannel+ ,
2094 Transport+ , DocExchange+)-->
2095         <PartyId type = "uriReference">urn:duns.com:duns:1234567890123</PartyId>
2096         <PartyId type = "uriReference">urn:www.example.com</PartyId>
2097         <PartyDetails xlink:type="simple"
2098 xlink:href="http://example2.com/example.com"/>
2099         <CollaborationRole roleId="N07" certId="N03">
2100             <CollaborationProtocol name = "Buy Sell" version = "1.0"
2101                 xlink:type = "locator"
2102                 xlink:href = "http://www.example.com/services/purchasing.xml"/>
2103             <Role name = "buyer" certId = "N03"
2104 xlink:href="http://www.example.com/services/purchasing.xml"/>
2105                 <!--(+)-->
2106             <ServiceBinding name="MyShopper" channelId="N04"/>
2107         </CollaborationRole>
2108         <CollaborationRole roleId="N12" certId="N03">
2109             <!--(+)-->
2110         </CollaborationRole>
2111         <Certificate certId = "N03">
2112             <!--(ds:KeyInfo)-->
2113             <ds:KeyInfo>REFERENCE [XMLDSIG]</ds:KeyInfo>
2114         </Certificate>
2115         <DeliveryChannel channelId = "N04"
2116             transportId = "N05" docExchangeId = "N06">
2117             <!--(Characteristics , ServiceBinding+)-->
2118             <Characteristics nonrepudiationOfOrigin = "true"
2119 nonrepudiationOfReceipt = "true" secureTransport = "true" confidentiality = "true"
2120 authenticated = "true" authorized = "true"/>
2121         </DeliveryChannel>
2122         <Transport transportId = "N05">
2123             <!--(Protocol , Endpoint+ , TransportTimeout? ,
2124 TransportSecurity?)-->
2125             <Protocol version = "1.1">HTTP</Protocol>
2126             <Endpoint uri = "http://example.com/servlet/ebxmlhandler" type =
2127 "request"/>

```

```
2129         <TransportSecurity>
2130             <!--(Protocol , CertificateRef?)-->
2131             <Protocol version = "3.0">SSL</Protocol>
2132             <CertificateRef certId = "N03"/>
2133         </TransportSecurity>
2134     </Transport>
2135     <DocExchange docExchangeId = "N06">
2136         <!--(ebXMLBinding)-->
2137         <ebXMLBinding version = "0.9">
2138             <!--(MessageEncoding? , ReliableMessaging , NonRepudiation?
2139 , DigitalEnvelope? , NamespaceSupported+)-->
2140             <MessageEncoding version = "base64" packagingType = "need to
2141 discuss">only text</MessageEncoding>
2142             <ReliableMessaging deliverySemantics = "BestEffort"
2143 idempotency = "false">
2144                 <!--(Timeout , Retries , RetryInterval)?-->
2145                 <Timeout>30</Timeout>
2146                 <Retries>5</Retries>
2147                 <RetryInterval>60</RetryInterval>
2148             </ReliableMessaging>
2149             <NonRepudiation>
2150                 <!--(Protocol , HashFunction , EncryptionAlgorithm ,
2151 SignatureAlgorithm , CertificateRef)-->
2152                 <Protocol version = "2.0">S/MIME</Protocol>
2153                 <HashFunction>shal</HashFunction>
2154                 <SignatureAlgorithm>rsa</SignatureAlgorithm>
2155                 <CertificateRef certId = "N03"/>
2156             </NonRepudiation>
2157             <DigitalEnvelope>
2158                 <!--(Protocol , EncryptionAlgorithm ,
2159 CertificateRef)-->
2160                 <Protocol version = "2.0">S/MIME</Protocol>
2161                 <EncryptionAlgorithm>rsa</EncryptionAlgorithm>
2162                 <CertificateRef certId = "N03"/>
2163             </DigitalEnvelope>
2164             <NamespaceSupported schemaLocation =
2165 "http://www.s2ml.org/s2ml.xsd" version =
2166 "0.7a">http://www.s2ml.org/s2ml/</NamespaceSupported>
2167         </ebXMLBinding>
2168     </DocExchange>
2169 </Party>
2170 <ds:Signature>any combination of text and elements</ds:Signature>
2171 </CollaborationProtocolProfile>
```

2172 **Appendix B** Example of *CPA* Document (Non-normative)

2173

2174 **This example is out of date and will be replaced along with the response to the next round**
2175 **of public review comments.**

2176

```

2177 <?xml version = "1.0"?>
2178 <!DOCTYPE CollaborationProtocolAgreement SYSTEM "cppml%2cv0.23.dtd">
2179 <!--Generated by XML Authority.-->
2180 <CollaborationProtocolAgreement id = "N01"
2181     xmlns="http://www.ebxml.org/namespaces/tradePartner"
2182     xmlns:bpm = "http://www.ebxml.org/namespaces/businessProcess"
2183     xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
2184     xmlns:xlink = "http://www.w3.org/1999/xlink">
2185     <!--(CPAType? , Status , Start , Duration , ConversationConstraints? , Party+ ,
2186 (CollaborationProtocol | bpm:BinaryCollaboration | bpm:BusinessTransactionActivity |
2187 bpm:ProcessSpecification)+ , ds:Signature?)-->
2188     <CPAType>
2189         <!--(Protocol , Type)-->
2190         <Protocol version = "1.1">PIP3A4</Protocol>
2191         <Type>RNIF</Type>
2192     </CPAType>
2193     <Status value = "proposed"/>
2194     <Start>1988-04-07T18:39:09</Start>
2195     <Duration>124</Duration>
2196     <ConversationConstraints invocationLimit = "100" concurrentConversations = "4"/>
2197     <Party partyId = "N01">
2198         <!--(PartyId+ , PartyDetails , Role+ , Certificate+ , DeliveryChannel+ ,
2199 Transport+ , DocExchange+)-->
2200         <PartyId type = "uriReference">urn:duns.com:duns:1234567890123</PartyId>
2201         <PartyId type = "uriReference">urn:www.example.com</PartyId>
2202         <PartyDetails xlink:type="simple"
2203 xlink:href="http://example.com/example2.com"/>
2204         <CollaborationRole roleId="N07" certId="N03">
2205             <CollaborationProtocol name = "Buy Sell" version = "1.0"
2206                 xlink:type = "locator"
2207                 xlink:href = "http://www.example.com/services/purchasing.xml"/>
2208             <Role name = "buyer" certId = "N03"
2209 xlink:href="http://www.example.com/services/purchasing.xml"/>
2210             <!--(+)-->
2211             <ServiceBinding name="MyShopper" channelId="N04"/>
2212         </CollaborationRole>
2213         <Certificate certId = "N03">
2214             <!--(ds:KeyInfo)-->
2215             <ds:KeyInfo>REFERENCE [XMLDSIG]</ds:KeyInfo>
2216         </Certificate>
2217         <DeliveryChannel channelId = "N04" transportId = "N05" docExchangeId =
2218 "N06">
2219             <!--(Characteristics , ServiceBinding+)-->
2220             <Characteristics nonrepudiationOfOrigin = "true"
2221 nonrepudiationOfReceipt = "true" secureTransport = "true" confidentiality = "true"
2222 authenticated = "true" authorized = "true"/>

```

```
2223         </DeliveryChannel>
2224         <Transport transportId = "N05">
2225             <!--(Protocol , Endpoint+ , TransportTimeout? ,
2226 TransportSecurity?)-->
2227             <Protocol version = "1.1">HTTP</Protocol>
2228             <Endpoint uri = "http://example2.com/servlet/ebxmlhandler" type =
2229 "request"/>
2230
2231             <TransportSecurity>
2232                 <!--(Protocol , CertificateRef?)-->
2233                 <Protocol version = "3.0">SSL</Protocol>
2234                 <CertificateRef certId = "N03"/>
2235             </TransportSecurity>
2236         </Transport>
2237         <DocExchange docExchangeId = "N06">
2238             <!--(ebXMLBinding)-->
2239             <ebXMLBinding version = "0.9">
2240                 <!--(MessageEncoding? , ReliableMessaging , NonRepudiation?
2241 , DigitalEnvelope? , NamespaceSupported+)-->
2242                 <MessageEncoding version = "base64" packagingType = "need to
2243 discuss">only text</MessageEncoding>
2244                 <ReliableMessaging deliverySemantics = "BestEffort"
2245 idempotency = "false">
2246                     <!--(Timeout , Retries , RetryInterval)?-->
2247                     <Timeout>30</Timeout>
2248                     <Retries>5</Retries>
2249                     <RetryInterval>60</RetryInterval>
2250                 </ReliableMessaging>
2251                 <NonRepudiation>
2252                     <!--(Protocol , HashFunction , EncryptionAlgorithm ,
2253 SignatureAlgorithm , CertificateRef)-->
2254                     <Protocol version = "2.0">S/MIME</Protocol>
2255                     <HashFunction>shal</HashFunction>
2256                     <SignatureAlgorithm>rsa</SignatureAlgorithm>
2257                     <CertificateRef certId = "N03"/>
2258                 </NonRepudiation>
2259                 <DigitalEnvelope>
2260                     <!--(Protocol , EncryptionAlgorithm ,
2261 CertificateRef)-->
2262                     <Protocol version = "2.0">S/MIME</Protocol>
2263                     <EncryptionAlgorithm>rsa</EncryptionAlgorithm>
2264                     <CertificateRef certId = "N03"/>
2265                 </DigitalEnvelope>
2266                 <NamespaceSupported schemaLocation =
2267 "http://www.s2ml.org/s2ml.xsd" version =
2268 "0.7a">http://www.s2ml.org/s2ml/</NamespaceSupported>
2269             </ebXMLBinding>
2270         </DocExchange>
2271     </Party>
2272     <Party partyId = "N01">
2273         <!--(PartyId+ , Role+ , Certificate+ , DeliveryChannel+ , Transport+ ,
2274 DocExchange+)-->
2275         <PartyId type = "uriReference">urn:duns.com:duns:1234567890123</PartyId>
2276         <PartyId type = "uriReference">urn:www.example.com</PartyId>
```

```

2277         <PartyDetails xlink:type="simple"
2278 xlink:href="http://example2.com/example.com"/>
2279         <Role certId = "N03" roleId = "N08" name = "seller">
2280             <!--(ServiceBinding+)-->
2281             <ServiceBinding collaborationId="N09" channelId="N04"/>
2282
2283         </Role>
2284         <CollaborationRole roleId="N07" certId="N03">
2285             <CollaborationProtocol name = "Buy Sell" version = "1.0"
2286                 xlink:type = "locator"
2287                 xlink:href = "http://www.example.com/services/purchasing.xml"/>
2288             <Role name = "buyer"
2289                 xlink:href="http://www.example.com/services/purchasing.xml"/>
2290             <!--(+)-->
2291             <ServiceBinding name="MyShopper" channelId="N04"/>
2292         </CollaborationRole>
2293         <Certificate certId = "N03">
2294             <!--(ds:KeyInfo)-->
2295             <ds:KeyInfo>REFERENCE [XMLDSIG]</ds:KeyInfo>
2296         </Certificate>
2297         <DeliveryChannel channelId = "N04" transportId = "N05" docExchangeId =
2298 "N06">
2299             <!--(Characteristics , ServiceBinding+)-->
2300             <Characteristics nonrepudiationOfOrigin = "true"
2301 nonrepudiationOfReceipt = "true" secureTransport = "true" confidentiality = "true"
2302 authenticated = "true" authorized = "true"/>
2303
2304         </DeliveryChannel>
2305         <Transport transportId = "N05">
2306             <!--(Protocol , Endpoint+ , TransportTimeout? ,
2307 TransportSecurity?)-->
2308             <Protocol version = "1.1">HTTP</Protocol>
2309             <Endpoint uri = "http://example.com/servlet/ebxmlhandler" type =
2310 "request"/>
2311
2312             <TransportSecurity>
2313                 <!--(Protocol , CertificateRef?)-->
2314                 <Protocol version = "3.0">SSL</Protocol>
2315                 <CertificateRef certId = "N03"/>
2316             </TransportSecurity>
2317         </Transport>
2318         <DocExchange docExchangeId = "N06">
2319             <!--(ebXMLBinding)-->
2320             <ebXMLBinding version = "0.9">
2321                 <!--(MessageEncoding? , ReliableMessaging , NonRepudiation?
2322 , DigitalEnvelope? , NamespaceSupported+)-->
2323                 <MessageEncoding version = "base64" packagingType = "need to
2324 discuss">only text</MessageEncoding>
2325                 <ReliableMessaging deliverySemantics = "BestEffort"
2326 idempotency = "false">
2327                     <!--(Timeout , Retries , RetryInterval)?-->
2328                     <Timeout>30</Timeout>
2329                     <Retries>5</Retries>
2330                     <RetryInterval>60</RetryInterval>

```

```
2331         </ReliableMessaging>
2332         <NonRepudiation>
2333             <!--(Protocol , HashFunction , EncryptionAlgorithm ,
2334 SignatureAlgorithm , CertificateRef)-->
2335             <Protocol version = "2.0">S/MIME</Protocol>
2336             <HashFunction>sha1</HashFunction>
2337             <SignatureAlgorithm>rsa</SignatureAlgorithm>
2338             <CertificateRef certId = "N03"/>
2339         </NonRepudiation>
2340         <DigitalEnvelope>
2341             <!--(Protocol , EncryptionAlgorithm ,
2342 CertificateRef)-->
2343             <Protocol version = "2.0">S/MIME</Protocol>
2344             <EncryptionAlgorithm>rsa</EncryptionAlgorithm>
2345             <CertificateRef certId = "N03"/>
2346         </DigitalEnvelope>
2347         <NamespaceSupported schemaLocation =
2348 "http://www.s2ml.org/s2ml.xsd" version =
2349 "0.7a">http://www.s2ml.org/s2ml/</NamespaceSupported>
2350     </ebXMLBinding>
2351 </DocExchange>
2352 </Party>
2353 <CollaborationProtocol version = "1.0" id = "N07" xlink:type = "locator"
2354 xlink:href = "http://www.example.com/services/purchasing.xml">Buy and Sell
2355 </CollaborationProtocol>
2356 <ds:Signature>any combination of text and elements</ds:Signature>
2357 </CollaborationProtocolAgreement>
2358
2359
2360
2361
```

2362 **Appendix C DTD Corresponding to Complete *CPPI/CPA***
 2363 **Definition (Normative)**

2364 **This DTD is out of date and will be replaced along with the response the next round of**
 2365 **public review comments.**

```

2366 <?xml version='1.0' encoding='UTF-8' ?>
2367
2368 <!--Generated by XML Authority-->
2369
2370
2371 <!ELEMENT CollaborationProtocolAgreement (CPAType? , Status , Start , End ,
2372 ConversationConstraints? , PartyInfo* , ds:Signature+ , Comment*)>
2373 <!ATTLIST CollaborationProtocolAgreement id CDATA #IMPLIED >
2374 <!ELEMENT CollaborationProtocolProfile (PartyInfo+ , ds:Signature? , Comment*)>
2375 <!ELEMENT ReceivingProtocol (#PCDATA)>
2376 <!ATTLIST ReceivingProtocol version CDATA #IMPLIED
2377 e-dtype NMTOKEN #FIXED 'string' >
2378 <!ELEMENT SendingProtocol (#PCDATA)>
2379 <!ATTLIST SendingProtocol version CDATA #IMPLIED
2380 e-dtype NMTOKEN #FIXED 'string' >
2381 <!ELEMENT Protocol (#PCDATA)>
2382 <!ATTLIST Protocol version CDATA #IMPLIED
2383 e-dtype NMTOKEN #FIXED 'string' >
2384 <!ELEMENT CollaborationRole (ProcessSpecification , Role , CertificateRef? , ServiceBinding+)>
2385 <!ATTLIST CollaborationRole id ID #REQUIRED >
2386 <!ELEMENT PartyInfo (PartyId+ , PartyRef , CollaborationRole+ , Certificate+ , DeliveryChannel+ ,
2387 Transport+ , DocExchange+)>
2388 <!ELEMENT PartyId (#PCDATA)>
2389 <!ATTLIST PartyId type CDATA #IMPLIED
2390 e-dtype NMTOKEN #FIXED 'string' >
2391 <!ELEMENT PartyRef EMPTY>
2392 <!ATTLIST PartyRef xlink:type (simple ) #FIXED 'simple'
2393 xlink:href CDATA #REQUIRED >
2394 <!ELEMENT DeliveryChannel (Characteristics)>
2395 <!ATTLIST DeliveryChannel channelId ID #REQUIRED
2396 transportId IDREF #REQUIRED
2397 docExchangeId IDREF #REQUIRED >
2398 <!ELEMENT Transport (SendingProtocol+ , ReceivingProtocol , Endpoint+ , TransportSecurity?)>
2399 <!ATTLIST Transport transportId ID #REQUIRED >
2400 <!ELEMENT Endpoint EMPTY>
2401 <!ATTLIST Endpoint uri CDATA #REQUIRED
2402 type (login | request | response | error | allPurpose ) 'allPurpose'
2403 a-dtype NMTOKENS 'uri uri' >
2404 <!ELEMENT Retries (#PCDATA)>
2405 <!ATTLIST Retries e-dtype NMTOKEN #FIXED 'string' >
2406 <!ELEMENT RetryInterval (#PCDATA)>
2407 <!ATTLIST RetryInterval e-dtype NMTOKEN #FIXED 'string' >
2408 <!ELEMENT TransportSecurity (Protocol , CertificateRef)>
2409 <!ELEMENT Certificate (ds:KeyInfo)>
2410 <!ATTLIST Certificate certId ID #REQUIRED >
2411 <!ELEMENT DocExchange (ebXMLBinding)>
2412 <!ATTLIST DocExchange docExchangeId ID #IMPLIED >
2413 <!ELEMENT ReliableMessaging (Retries , RetryInterval)?>
2414 <!ATTLIST ReliableMessaging deliverySemantics (OnceAndOnlyOnce | BestEffort ) #REQUIRED
2415 idempotency CDATA #REQUIRED
2416 persistDuration CDATA #REQUIRED
2417 a-dtype NMTOKENS 'idempotency boolean'
2418 e-dtype NMTOKEN #FIXED 'timeDuration' >
2419 <!ELEMENT NonRepudiation (Protocol , HashFunction , SignatureAlgorithm , CertificateRef)>
2420 <!ELEMENT HashFunction (#PCDATA)>
2421 <!ATTLIST HashFunction e-dtype NMTOKEN #FIXED 'string' >
2422 <!ELEMENT EncryptionAlgorithm (#PCDATA)>
2423 <!ATTLIST EncryptionAlgorithm e-dtype NMTOKEN #FIXED 'string' >
2424 <!ELEMENT SignatureAlgorithm (#PCDATA)>
2425 <!ATTLIST SignatureAlgorithm e-dtype NMTOKEN #FIXED 'string' >
2426 <!ELEMENT DigitalEnvelope (Protocol , EncryptionAlgorithm , CertificateRef)>

```



```

2427 <!ELEMENT ProcessSpecification (ds:Reference)>
2428 <!ATTLIST ProcessSpecification
2429     name ID #REQUIRED
2430     version CDATA #REQUIRED >
2431     xlink:type (simple ) #FIXED 'simple'
2432     xlink:href CDATA #REQUIRED >
2433 <!ELEMENT ds:Reference (ds:DigestMethod, ds:DigestValue)>
2434 <!ATTLIST ds:Reference ds:URI CDATA #REQUIRED
2435     type CDATA #IMPLIED >
2436 <!ELEMENT ds:DigestMethod ( #PCDATA | ds:HMACOutputLength )*>
2437 <!ATTLIST ds:DigestMethod
2438     ds:Algorithm CDATA #REQUIRED>
2439 <!ELEMENT ds:HMACOutputLength (#PCDATA)>
2440 <!ELEMENT ds:DigestValue (#PCDATA)>
2441 <!ELEMENT CertificateRef EMPTY>
2442 <!ATTLIST CertificateRef certId IDREF #IMPLIED
2443     e-dtype NMTOKEN #FIXED 'string' >
2444 <!ELEMENT MessageEncoding (#PCDATA)>
2445 <!ATTLIST MessageEncoding version CDATA #REQUIRED
2446     packagingType CDATA #IMPLIED
2447     e-dtype NMTOKEN #FIXED 'string' >
2448 <!ELEMENT ebXMLBinding (MessageEncoding? , ReliableMessaging? , NonRepudiation? ,
2449 DigitalEnvelope? , NamespaceSupported+)>
2450 <!ATTLIST ebXMLBinding version CDATA #REQUIRED >
2451 <!ELEMENT ds:KeyInfo EMPTY>
2452 <!ELEMENT ds:Signature EMPTY>
2453 <!ELEMENT NamespaceSupported (#PCDATA)>
2454 <!ATTLIST NamespaceSupported schemaLocation CDATA #IMPLIED
2455     version CDATA #REQUIRED
2456     e-dtype NMTOKEN #FIXED 'uri'
2457     a-dtype NMTOKENS 'schemaLocation uri' >
2458 <!ELEMENT EMPTY>
2459 <!ATTLIST Characteristics nonrepudiationOfOrigin CDATA #IMPLIED
2460     nonrepudiationOfReceipt CDATA #IMPLIED
2461     secureTransport CDATA #IMPLIED
2462     confidentiality CDATA #IMPLIED
2463     authenticated CDATA #IMPLIED
2464     authorized CDATA #IMPLIED
2465     a-dtype NMTOKENS 'nonrepudiationOfOrigin boolean
2466     nonrepudiationOfReceipt boolean
2467     secureTransport boolean
2468     confidentiality boolean
2469     authenticated boolean
2470     authorized boolean' >
2471 <!ELEMENT ServiceBinding (Packaging+ , Override*)>
2472 <!ATTLIST ServiceBinding channelId ID REF #REQUIRED
2473     name CDATA #IMPLIED >
2474 <!ELEMENT CPAType (Protocol , Type)>
2475 <!ELEMENT Status EMPTY>
2476 <!ATTLIST Status value (signed |agreed | proposed ) #REQUIRED >
2477 <!ELEMENT Start (#PCDATA)>
2478 <!ATTLIST Start e-dtype NMTOKEN #FIXED 'timeInstant' >
2479 <!ELEMENT End (#PCDATA)>
2480 <!ATTLIST End e-dtype NMTOKEN #FIXED 'timeInstant' >
2481 <!ELEMENT Type (#PCDATA)>
2482 <!ATTLIST Type e-dtype NMTOKEN #FIXED 'string' >
2483 <!ELEMENT ConversationConstraints EMPTY>
2484 <!ATTLIST ConversationConstraints invocationLimit CDATA #IMPLIED
2485     concurrentConversations CDATA #IMPLIED
2486     a-dtype NMTOKENS
2487     'invocationLimit i4
2488     concurrentConversations i4' >
2489 <!ELEMENT Override EMPTY>
2490 <!ATTLIST Override action CDATA #REQUIRED
2491     channelId ID #REQUIRED
2492     xlink:href CDATA #IMPLIED
2493     xlink:type (simple ) #FIXED 'simple' >
2494 <!ELEMENT Role (#PCDATA)>
2495 <!ATTLIST Role name CDATA #IMPLIED
2496     xlink:href CDATA #IMPLIED
2497     xlink:type (simple ) #FIXED 'simple' >
2498 <!ELEMENT Packaging (ProcessingCapabilities , SimplePart+ , CompositeList?)+>

```

```
2499 <!ELEMENT Comment ANY>
2500 <!ELEMENT Composite (Constituent+)>
2501 <!ATTLIST Composite mimetype CDATA #REQUIRED
2502 id ID #REQUIRED
2503 mimeparameters CDATA #IMPLIED >
2504 <!ELEMENT Constituent EMPTY>
2505 <!ATTLIST Constituent idref IDREF #REQUIRED >
2506 <!ELEMENT Encapsulation (Constituent)>
2507 <!ATTLIST Encapsulation mimetype CDATA #REQUIRED
2508 id ID #REQUIRED
2509 mimeparameters CDATA #IMPLIED >
2510 <!ELEMENT CompositeList (Encapsulation | Composite)+>
2511 <!ELEMENT XMLMetaDataInformation EMPTY>
2512 <!ATTLIST XMLMetaDataInformation URI CDATA #IMPLIED
2513 MetadataDescriptionType (dtd | xsd ) #REQUIRED >
2514 <!ELEMENT MimeHeader EMPTY>
2515 <!ATTLIST MimeHeader HeaderName CDATA #REQUIRED >
2516 <!ELEMENT MimeParameter EMPTY>
2517 <!ATTLIST MimeParameter parameterAttribute CDATA #REQUIRED
2518 parameterValue CDATA #IMPLIED >
2519 <!ELEMENT SimplePart EMPTY>
2520 <!ATTLIST SimplePart id ID #REQUIRED
2521 mimetype CDATA #REQUIRED >
2522 <!ELEMENT ProcessingCapabilities EMPTY>
2523 <!ATTLIST ProcessingCapabilities parse CDATA #REQUIRED
2524 generate CDATA #REQUIRED >
2525 <!ELEMENT ds:Reference EMPTY>
2526
2527
2528
```

2529 **Appendix D XML Schema Document Corresponding to**
 2530 **Complete CPA Definition (Normative)**

2531
 2532 **This schema is out of date and will be replaced along with the response to the next round of**
 2533 **public review comments.**

```

2534
2535 <?xml version = "1.0" encoding = "UTF-8"?>
2536 <!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2000/10/XMLSchema-->
2537 <xsd:schema xmlns:xlink = "http://www.w3.org/1999/xlink"
2538     xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
2539     xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema">
2540   <xsd:import namespace = "http://www.w3.org/1999/xlink" schemaLocation =
2541 "http://www.w3.org/1999/xlink"/>
2542   <xsd:import namespace = "http://www.w3.org/2000/09/xmldsig#" schemaLocation =
2543 "file:///C:/My%20Documents/ebXML/xmldsig-core-schema.xsd"/>
2544   <xsd:element name = "CollaborationProtocolAgreement">
2545     <xsd:complexType>
2546       <xsd:sequence>
2547         <xsd:element ref = "CPAType" minOccurs = "0"/>
2548         <xsd:element ref = "Status"/>
2549         <xsd:element ref = "Start"/>
2550         <xsd:element ref = "Duration"/>
2551         <xsd:element ref = "ConversationConstraints" minOccurs = "0"/>
2552         <xsd:element ref = "PartyInfo" minOccurs = "0" maxOccurs =
2553 "unbounded"/>
2554         <xsd:element ref = "ds:Signature" minOccurs = "0"/>
2555         <xsd:element ref = "Comment" minOccurs = "0" maxOccurs =
2556 "unbounded"/>
2557       </xsd:sequence>
2558       <xsd:attribute name = "id" type = "xsd:string"/>
2559     </xsd:complexType>
2560   </xsd:element>
2561   <xsd:element name = "CollaborationProtocolProfile">
2562     <xsd:complexType>
2563       <xsd:sequence>
2564         <xsd:element ref = "PartyInfo" maxOccurs = "unbounded"/>
2565         <xsd:element ref = "ds:Signature" minOccurs = "0"/>
2566         <xsd:element ref = "Comment" minOccurs = "0" maxOccurs =
2567 "unbounded"/>
2568       </xsd:sequence>
2569     </xsd:complexType>
2570   </xsd:element>
2571   <xsd:element name = "ReceivingProtocol">
2572     <xsd:complexType>
2573       <xsd:simpleContent>
2574         <xsd:extension base = "xsd:string">
2575           <xsd:attribute name = "version" type = "xsd:string"/>
2576         </xsd:extension>
2577       </xsd:simpleContent>
2578     </xsd:complexType>
2579   </xsd:element>
2580   <xsd:element name = "SendingProtocol">
2581     <xsd:complexType>
2582       <xsd:simpleContent>
2583         <xsd:extension base = "xsd:string">
2584           <xsd:attribute name = "version" type = "xsd:string"/>
2585         </xsd:extension>
2586       </xsd:simpleContent>
2587     </xsd:complexType>
2588   </xsd:element>
2589   <xsd:element name = "Protocol" type = "xsd:string"/>
2590   <xsd:element name = "CollaborationRole">
2591     <xsd:complexType>
2592       <xsd:sequence>

```

```

2593         <xsd:element ref = "ProcessSpecification"/>
2594         <xsd:element ref = "Role"/>
2595         <xsd:element ref = "CertificateRef" minOccurs = "0"/>
2596         <xsd:element ref = "ServiceBinding" maxOccurs = "unbounded"/>
2597     </xsd:sequence>
2598     <xsd:attribute name = "roleId" use = "required" type = "xsd:ID"/>
2599 </xsd:complexType>
2600 </xsd:element>
2601 <xsd:element name = "PartyInfo">
2602     <xsd:complexType>
2603         <xsd:sequence>
2604             <xsd:element ref = "PartyId" maxOccurs = "unbounded"/>
2605             <xsd:element ref = "PartyRef"/>
2606             <xsd:element ref = "CollaborationRole" maxOccurs = "unbounded"/>
2607             <xsd:element ref = "Certificate" maxOccurs = "unbounded"/>
2608             <xsd:element ref = "DeliveryChannel" maxOccurs = "unbounded"/>
2609             <xsd:element ref = "Transport" maxOccurs = "unbounded"/>
2610             <xsd:element ref = "DocExchange" maxOccurs = "unbounded"/>
2611         </xsd:sequence>
2612     </xsd:complexType>
2613 </xsd:element>
2614 <xsd:element name = "PartyId">
2615     <xsd:complexType>
2616         <xsd:simpleContent>
2617             <xsd:extension base = "xsd:string">
2618                 <xsd:attribute name = "type" type = "xsd:string"/>
2619             </xsd:extension>
2620         </xsd:simpleContent>
2621     </xsd:complexType>
2622 </xsd:element>
2623 <xsd:element name = "PartyRef">
2624     <xsd:complexType>
2625         <xsd:sequence/>
2626         <xsd:attribute name = "xlink:type" use = "fixed" value = "simple">
2627             <xsd:simpleType>
2628                 <xsd:restriction base = "xsd:NMTOKEN">
2629                     <xsd:enumeration value = "simple"/>
2630                 </xsd:restriction>
2631             </xsd:simpleType>
2632         </xsd:attribute>
2633         <xsd:attribute name = "xlink:href" use = "required" type = "xsd:string"/>
2634     </xsd:complexType>
2635 </xsd:element>
2636 <xsd:element name = "DeliveryChannel">
2637     <xsd:complexType>
2638         <xsd:sequence>
2639             <xsd:element ref = "Characteristics"/>
2640         </xsd:sequence>
2641         <xsd:attribute name = "channelId" use = "required" type = "xsd:ID"/>
2642         <xsd:attribute name = "transportId" use = "required" type = "xsd:IDREF"/>
2643         <xsd:attribute name = "docExchangeId" type = "xsd:IDREF"/>
2644     </xsd:complexType>
2645 </xsd:element>
2646 <xsd:element name = "Transport">
2647     <xsd:complexType>
2648         <xsd:sequence>
2649             <xsd:element ref = "SendingProtocol"/>
2650             <xsd:element ref = "ReceivingProtocol"/>
2651             <xsd:element ref = "Endpoint" maxOccurs = "unbounded"/>
2652             <xsd:element ref = "TransportSecurity" minOccurs = "0"/>
2653         </xsd:sequence>
2654         <xsd:attribute name = "transportId" use = "required" type = "xsd:ID"/>
2655     </xsd:complexType>
2656 </xsd:element>
2657 <xsd:element name = "Endpoint">
2658     <xsd:complexType>
2659         <xsd:sequence/>
2660         <xsd:attribute name = "uri" use = "required" type = "xsd:uriReference"/>
2661         <xsd:attribute name = "type" use = "default" value = "allPurpose">
2662             <xsd:simpleType>
2663                 <xsd:restriction base = "xsd:NMTOKEN">

```

```

2664         <xsd:enumeration value = "login"/>
2665         <xsd:enumeration value = "request"/>
2666         <xsd:enumeration value = "response"/>
2667         <xsd:enumeration value = "error"/>
2668         <xsd:enumeration value = "allPurpose"/>
2669     </xsd:restriction>
2670 </xsd:simpleType>
2671 </xsd:attribute>
2672 </xsd:complexType>
2673 </xsd:element>
2674 <xsd:element name = "TransportEncoding" type = "xsd:string"/>
2675 <xsd:element name = "Retries" type = "xsd:string"/>
2676 <xsd:element name = "RetryInterval" type = "xsd:timePeriod"/>
2677 <xsd:element name = "TransportSecurity">
2678     <xsd:complexType>
2679         <xsd:sequence>
2680             <xsd:element ref = "Protocol"/>
2681             <xsd:element ref = "CertificateRef" minOccurs = "0"/>
2682         </xsd:sequence>
2683     </xsd:complexType>
2684 </xsd:element>
2685 <xsd:element name = "Certificate">
2686     <xsd:complexType>
2687         <xsd:sequence>
2688             <xsd:element ref = "ds:KeyInfo"/>
2689         </xsd:sequence>
2690         <xsd:attribute name = "certId" use = "required" type = "xsd:ID"/>
2691     </xsd:complexType>
2692 </xsd:element>
2693 <xsd:element name = "DocExchange">
2694     <xsd:complexType>
2695         <xsd:sequence>
2696             <xsd:element ref = "ebXMLBinding"/>
2697         </xsd:sequence>
2698         <xsd:attribute name = "docExchangeId" type = "xsd:ID"/>
2699     </xsd:complexType>
2700 </xsd:element>
2701 <xsd:element name = "ReliableMessaging">
2702     <xsd:complexType>
2703         <xsd:sequence minOccurs = "0">
2704             <xsd:element ref = "Retries"/>
2705             <xsd:element ref = "RetryInterval"/>
2706         </xsd:sequence>
2707         <xsd:attribute name = "deliverySemantics" use = "required">
2708             <xsd:simpleType>
2709                 <xsd:restriction base = "xsd:NMTOKEN">
2710                     <xsd:enumeration value = "OnceAndOnlyOnce"/>
2711                     <xsd:enumeration value = "BestEffort"/>
2712                 </xsd:restriction>
2713             </xsd:simpleType>
2714         </xsd:attribute>
2715         <xsd:attribute name = "idempotency" use = "required" type =
2716 "xsd:boolean"/>
2717     </xsd:complexType>
2718 </xsd:element>
2719 <xsd:element name = "NonRepudiation">
2720     <xsd:complexType>
2721         <xsd:sequence>
2722             <xsd:element ref = "Protocol"/>
2723             <xsd:element ref = "HashFunction"/>
2724             <xsd:element ref = "SignatureAlgorithm"/>
2725             <xsd:element ref = "CertificateRef"/>
2726         </xsd:sequence>
2727     </xsd:complexType>
2728 </xsd:element>
2729 <xsd:element name = "HashFunction" type = "xsd:string"/>
2730 <xsd:element name = "EncryptionAlgorithm" type = "xsd:string"/>
2731 <xsd:element name = "SignatureAlgorithm" type = "xsd:string"/>
2732 <xsd:element name = "DigitalEnvelope">
2733     <xsd:complexType>
2734         <xsd:sequence>

```

```

2735             <xsd:element ref = "Protocol"/>
2736             <xsd:element ref = "EncryptionAlgorithm"/>
2737             <xsd:element ref = "CertificateRef"/>
2738         </xsd:sequence>
2739     </xsd:complexType>
2740 </xsd:element>
2741 <xsd:element name = "ProcessSpecification">
2742     <xsd:complexType>
2743         <xsd:sequence>
2744             <xsd:element ref = "ds:Reference"/>
2745         </xsd:sequence>
2746         <xsd:attribute name = "version" use = "required" type = "xsd:string"/>
2747         <xsd:attribute name = "name" use = "required" type = "xsd:string"/>
2748     </xsd:complexType>
2749 </xsd:element>
2750 <xsd:element name = "CertificateRef">
2751     <xsd:complexType>
2752         <xsd:simpleContent>
2753             <xsd:extension base = "xsd:string">
2754                 <xsd:attribute name = "certId" type = "xsd:IDREF"/>
2755             </xsd:extension>
2756         </xsd:simpleContent>
2757     </xsd:complexType>
2758 </xsd:element>
2759 <xsd:element name = "MessageEncoding">
2760     <xsd:complexType>
2761         <xsd:simpleContent>
2762             <xsd:extension base = "xsd:string">
2763                 <xsd:attribute name = "version" use = "required" type =
2764 "xsd:string"/>
2765                 <xsd:attribute name = "packagingType" type = "xsd:string"/>
2766             </xsd:extension>
2767         </xsd:simpleContent>
2768     </xsd:complexType>
2769 </xsd:element>
2770 <xsd:element name = "ebXMLBinding">
2771     <xsd:complexType>
2772         <xsd:sequence>
2773             <xsd:element ref = "MessageEncoding" minOccurs = "0"/>
2774             <xsd:element ref = "ReliableMessaging" minOccurs = "0"/>
2775             <xsd:element ref = "NonRepudiation" minOccurs = "0"/>
2776             <xsd:element ref = "DigitalEnvelope" minOccurs = "0"/>
2777             <xsd:element ref = "NamespaceSupported" maxOccurs = "unbounded"/>
2778         </xsd:sequence>
2779         <xsd:attribute name = "version" use = "required" type = "xsd:string"/>
2780     </xsd:complexType>
2781 </xsd:element>
2782 <xsd:element name = "ds:KeyInfo">
2783     <xsd:complexType>
2784         <xsd:sequence/>
2785     </xsd:complexType>
2786 </xsd:element>
2787 <xsd:element name = "ds:Signature">
2788     <xsd:complexType>
2789         <xsd:sequence/>
2790     </xsd:complexType>
2791 </xsd:element>
2792 <xsd:element name = "NamespaceSupported">
2793     <xsd:complexType>
2794         <xsd:simpleContent>
2795             <xsd:extension base = "xsd:uriReference">
2796                 <xsd:attribute name = "schemaLocation" type =
2797 "xsd:uriReference"/>
2798                 <xsd:attribute name = "version" use = "required" type =
2799 "xsd:string"/>
2800             </xsd:extension>
2801         </xsd:simpleContent>
2802     </xsd:complexType>
2803 </xsd:element>
2804 <xsd:element name = "Characteristics">
2805     <xsd:complexType>

```

```

2806         <xsd:sequence/>
2807         <xsd:attribute name = "nonrepudiationOfOrigin" type = "xsd:boolean"/>
2808         <xsd:attribute name = "nonrepudiationOfReceipt" type = "xsd:boolean"/>
2809         <xsd:attribute name = "secureTransport" type = "xsd:boolean"/>
2810         <xsd:attribute name = "confidentiality" type = "xsd:boolean"/>
2811         <xsd:attribute name = "authenticated" type = "xsd:boolean"/>
2812         <xsd:attribute name = "authorized" type = "xsd:boolean"/>
2813     </xsd:complexType>
2814 </xsd:element>
2815 <xsd:element name = "ServiceBinding">
2816     <xsd:complexType>
2817         <xsd:sequence>
2818             <xsd:element ref = "Packaging" maxOccurs = "unbounded"/>
2819             <xsd:element ref = "Override" minOccurs = "0"/>
2820         </xsd:sequence>
2821         <xsd:attribute name = "channelId" use = "required" type = "xsd:ID"/>
2822         <xsd:attribute name = "name" type = "xsd:string"/>
2823     </xsd:complexType>
2824 </xsd:element>
2825 <xsd:element name = "CPAType">
2826     <xsd:complexType>
2827         <xsd:sequence>
2828             <xsd:element ref = "Protocol"/>
2829             <xsd:element ref = "Type"/>
2830         </xsd:sequence>
2831     </xsd:complexType>
2832 </xsd:element>
2833 <xsd:element name = "Status">
2834     <xsd:complexType>
2835         <xsd:sequence/>
2836         <xsd:attribute name = "value" use = "required">
2837             <xsd:simpleType>
2838                 <xsd:restriction base = "xsd:NMTOKEN">
2839                     <xsd:enumeration value = "signed"/>
2840                     <xsd:enumeration value = "proposed"/>
2841                 </xsd:restriction>
2842             </xsd:simpleType>
2843         </xsd:attribute>
2844     </xsd:complexType>
2845 </xsd:element>
2846 <xsd:element name = "Start" type = "xsd:timeInstant"/>
2847 <xsd:element name = "Duration" type = "xsd:timePeriod"/>
2848 <xsd:element name = "Type" type = "xsd:string"/>
2849 <xsd:element name = "ConversationConstraints">
2850     <xsd:complexType>
2851         <xsd:sequence/>
2852         <xsd:attribute name = "invocationLimit" type = "xsd:int"/>
2853         <xsd:attribute name = "concurrentConversations" type = "xsd:int"/>
2854     </xsd:complexType>
2855 </xsd:element>
2856 <xsd:element name = "Override">
2857     <xsd:complexType>
2858         <xsd:sequence/>
2859         <xsd:attribute name = "action" type = "xsd:string"/>
2860         <xsd:attribute name = "channelId" use = "required" type = "xsd:ID"/>
2861         <xsd:attribute name = "xlink:href" type = "xsd:string"/>
2862         <xsd:attribute name = "xlink:type" use = "fixed" value = "simple">
2863             <xsd:simpleType>
2864                 <xsd:restriction base = "xsd:NMTOKEN">
2865                     <xsd:enumeration value = "simple"/>
2866                 </xsd:restriction>
2867             </xsd:simpleType>
2868         </xsd:attribute>
2869     </xsd:complexType>
2870 </xsd:element>
2871 <xsd:element name = "Role">
2872     <xsd:complexType>
2873         <xsd:simpleContent>
2874             <xsd:extension base = "xsd:string">
2875                 <xsd:attribute name = "name" type = "xsd:string"/>
2876                 <xsd:attribute name = "xlink:href" type = "xsd:string"/>

```

```

2877                                     <xsd:attribute name = "xlink:type" use = "fixed" value =
2878 "simple">
2879                                     <xsd:simpleType>
2880                                         <xsd:restriction base = "xsd:NMTOKEN">
2881                                             <xsd:enumeration value = "simple" />
2882                                         </xsd:restriction>
2883                                     </xsd:simpleType>
2884                                     </xsd:attribute>
2885                                 </xsd:extension>
2886                                 </xsd:simpleContent>
2887                             </xsd:complexType>
2888     </xsd:element>
2889     <xsd:element name = "SecurityRisks" type = "xsd:string" />
2890     <xsd:element name = "SecurityBenefits" type = "xsd:string" />
2891     <xsd:element name = "Packaging">
2892         <xsd:complexType>
2893             <xsd:sequence maxOccurs = "unbounded">
2894                 <xsd:element ref = "ProcessingCapabilities" />
2895                 <xsd:element ref = "SimplePart" maxOccurs = "unbounded" />
2896                 <xsd:element ref = "CompositeList" minOccurs = "0" />
2897             </xsd:sequence>
2898         </xsd:complexType>
2899     </xsd:element>
2900     <xsd:element name = "Comment">
2901         <xsd:complexType>
2902             <xsd:sequence />
2903         </xsd:complexType>
2904     </xsd:element>
2905     <xsd:element name = "Composite">
2906         <xsd:complexType>
2907             <xsd:sequence>
2908                 <xsd:element ref = "Constituent" maxOccurs = "unbounded" />
2909             </xsd:sequence>
2910             <xsd:attribute name = "mimetype" use = "required" type = "xsd:string" />
2911             <xsd:attribute name = "id" use = "required" type = "xsd:string" />
2912             <xsd:attribute name = "mimeparameters" type = "xsd:string" />
2913         </xsd:complexType>
2914     </xsd:element>
2915     <xsd:element name = "Constituent">
2916         <xsd:complexType>
2917             <xsd:sequence />
2918             <xsd:attribute name = "idref" use = "required" type = "xsd:string" />
2919         </xsd:complexType>
2920     </xsd:element>
2921     <xsd:element name = "Encapsulation">
2922         <xsd:complexType>
2923             <xsd:sequence>
2924                 <xsd:element ref = "Constituent" />
2925             </xsd:sequence>
2926             <xsd:attribute name = "mimetype" use = "required" type = "xsd:string" />
2927             <xsd:attribute name = "id" use = "required" type = "xsd:string" />
2928             <xsd:attribute name = "mimeparameters" type = "xsd:string" />
2929         </xsd:complexType>
2930     </xsd:element>
2931     <xsd:element name = "CompositeList">
2932         <xsd:complexType>
2933             <xsd:choice maxOccurs = "unbounded">
2934                 <xsd:element ref = "Encapsulation" />
2935                 <xsd:element ref = "Composite" />
2936             </xsd:choice>
2937         </xsd:complexType>
2938     </xsd:element>
2939     <xsd:element name = "XMLMetaDataInformation">
2940         <xsd:complexType>
2941             <xsd:sequence />
2942             <xsd:attribute name = "URI" type = "xsd:string" />
2943             <xsd:attribute name = "MetaDataDescriptionType" use = "required">
2944                 <xsd:simpleType>
2945                     <xsd:restriction base = "xsd:NMTOKEN">
2946                         <xsd:enumeration value = "dtd" />
2947                         <xsd:enumeration value = "xsd" />

```



```
2948         </xsd:restriction>
2949         </xsd:simpleType>
2950     </xsd:attribute>
2951 </xsd:complexType>
2952 </xsd:element>
2953 <xsd:element name = "MimeType" >
2954     <xsd:complexType>
2955         <xsd:sequence/>
2956         <xsd:attribute name = "HeaderName" use = "required" type = "xsd:string"/>
2957     </xsd:complexType>
2958 </xsd:element>
2959 <xsd:element name = "MimeTypeParameter" >
2960     <xsd:complexType>
2961         <xsd:sequence/>
2962         <xsd:attribute name = "parameterAttribute" use = "required" type =
2963 "xsd:string"/>
2964         <xsd:attribute name = "parameterValue" type = "xsd:string"/>
2965     </xsd:complexType>
2966 </xsd:element>
2967 <xsd:element name = "SimplePart" >
2968     <xsd:complexType>
2969         <xsd:sequence/>
2970         <xsd:attribute name = "id" use = "required" type = "xsd:string"/>
2971         <xsd:attribute name = "mimetype" use = "required" type = "xsd:string"/>
2972     </xsd:complexType>
2973 </xsd:element>
2974 <xsd:element name = "ProcessingCapabilities" >
2975     <xsd:complexType>
2976         <xsd:sequence/>
2977         <xsd:attribute name = "parse" use = "required" type = "xsd:string"/>
2978         <xsd:attribute name = "generate" use = "required" type = "xsd:string"/>
2979     </xsd:complexType>
2980 </xsd:element>
2981 <xsd:element name = "ds:Reference" >
2982     <xsd:complexType>
2983         <xsd:sequence/>
2984     </xsd:complexType>
2985 </xsd:element>
2986 </xsd:schema>
```

2987 **Appendix E** Formats of Information in the *CPP* and *CPA*
2988 (Normative)

2989 This section defines format information that is not defined by the [XML] specification and is not
2990 defined in the descriptions of specific elements.

2991

2992 Formats of Character Strings

2993

2994 **Protocol and Version Elements**

2995

2996 Values of *Protocol*, *Version*, and similar elements are flexible. In general, any protocol and
2997 version for which the support software is available to both *Parties* to a *CPA* MAY be selected as
2998 long as the choice does not require changes to the DTD or schema and therefore a change to this
2999 specification.

3000

3001 NOTE: A possible implementation MAY be based on the use of plug-ins or exits to
3002 support the values of these elements.

3003

3004 **Alphanumeric Strings**

3005

3006 Alphanumeric strings not further defined in this section follow these rules unless otherwise
3007 stated in the description of an individual element:

3008

- 3009 • Values of elements are case insensitive unless otherwise stated.
- 3010 • Strings which represent file or directory names are case sensitive to ensure that they are
3011 acceptable to both UNIX and Windows systems.

3012

3013 **Numeric Strings**

3014

3015 A numeric string is a signed or unsigned decimal integer in the range imposed by a 32-bit binary
3016 number, i.e. -2,147,483,648 to +2,417,483,647. Negative numbers MAY or MAY not be
3017 permitted in particular elements.

3018 **Appendix F** Composing a *CPA* from Two *CPPs* (Non- 3019 Normative)

3020 3021 Overview and Limitations

3022
3023 In this appendix, we discuss the tasks involved in *CPA* formation from *CPPs*. The detailed
3024 procedures for *CPA* formation are currently left for implementers. Therefore, no normative
3025 specification is provided for algorithms for *CPA* formation. In this initial section, we provide
3026 some background on *CPA* formation tasks.

3027
3028 There are three basic reasons why we prefer to provide information about the component tasks
3029 involved in *CPA* formation rather than attempt to provide an algorithm for *CPA* formation:

- 3030 1. The precise informational inputs to the *CPA* formation procedure vary.
- 3031 2. There exist at least two distinct approaches to *CPA* formation. One useful approach for
3032 certain situations involves basing *CPA* formation from a *CPA* template; the other approach
3033 involves composition from *CPPs*.
- 3034 3. The conditions for output of a given *CPA* given two *CPPs* can involve different levels and
3035 extents of interoperability. In other words, when an optimal solution that satisfies every level
3036 of requirement and every other additional constraint does not exist, a *Party* MAY propose a
3037 *CPA* that satisfies enough of the requirements for “a good enough” implementation. User
3038 input MAY be solicited to determine what is a good enough implementation, and so MAY
3039 be as varied as there are user configuration options to express preferences. In practice,
3040 compromises MAY be made on security, reliable messaging, levels of signals and
3041 acknowledgements, and other matters in order to find some acceptable means of doing
3042 business.

3043
3044 Each of these reasons is elaborated in greater detail in the following sections.

3045 3046 3047 3048 Variability in Inputs

3049
3050 User preferences provide one source of variability in the inputs to the *CPA* formation process.
3051 Let us suppose in this section that each of the *Parties* has made its *CPP* available to potential
3052 collaborators. Normally one *Party* will have a desired *Collaboration Protocol* (defined in a
3053 *Process-Specification* document) to implement with its intended collaborator. So the information
3054 inputs will normally involve a user preference about intended *Collaboration Protocols* in
3055 addition to just the *CPPs*.

3056
3057 A *CPA* formation tool MAY have access to local user information not advertised in the *CPP* that
3058 MAY contribute to the *CPA* that is formed. A user MAY have chosen to only advertise those
3059 system capabilities that reflect nondeprecated capabilities. For example, a user MAY only
3060 advertize HTTP and omit FTP, even when capable of using FTP, because of concerns about the
3061 scalability of managing user accounts, directories, and passwords for FTP sessions. Despite not

3062 advertising a FTP capability, configuration software MAY use tacit knowledge about its own
3063 FTP capability to form a *CPA* with an intended collaborator who happens to have only an FTP
3064 capability for implementing a desired business collaboration. In other words, business interests
3065 MAY, in this case, override the deprecation policy. Both tacit knowledge as well as detailed
3066 preference information account for variability in inputs into the *CPA* formation process.

3067
3068

3069 Different Approaches

3070

3071 When a *CPA* is formed from a *CPA* template, it is typically because the capabilities of one of the
3072 *Parties* are limited, and already tacitly known. For example, if a *CPA* template were implicitly
3073 presented to a Web browser for use in an implementation using browser based forms capabilities,
3074 then the template maker can assume that the other *Party* has suitable web capabilities (or is about
3075 to download them). Therefore, all that really needs to be done is to supply ***PartyRef***, ***Certificate***,
3076 and similar items for substitution into a *CPA* template. The *CPA* template will already have all
3077 the capabilities of both *Parties* specified at the various levels, and will have placeholders for
3078 values to be supplied by one of the *Partners*. A simple form might be adequate to gather the
3079 needed information and produce a *CPA*.

3080
3081

3082 Variable Output "Satisficing" Policies

3083

3084 A *CPA* can support a fully interoperable configuration in which agreement has been reached on
3085 all technical levels needed for business collaboration. In such a case, matches in capabilities will
3086 have been found in all relevant technical levels.

3087

3088 However, there can be interoperable configurations agreed to in a *CPA* in which not all aspects
3089 of a business collaboration match. Gaps MAY exist in packaging, security, signaling, reliable
3090 messaging and other areas and yet the systems can still transport the business data, and special
3091 means can be employed to handle the exceptions. In such situations, a *CPA* MAY reflect
3092 configured policies or expressly solicited user permission to ignore some shortcomings in
3093 configurations. A system might not be capable of responding in a business collaboration so as to
3094 support a recommended ability to supply nonrepudiation of receipt, but might still be acceptable
3095 for business reasons. A system might not be able to handle all the processing required to support
3096 "multipart/related" processing with a type value of "application/vnd.eb+xml," and yet still be
3097 able to treat the multipart according to "multipart/mixed" handling and allow business
3098 collaboration to take place. In fact, short of a failure to be able to transport data and a failure to
3099 be able to provide data relevant to the *Business Process*, there are few features that might not be
3100 temporarily or indefinitely compromised about, given overriding business interests. This
3101 situation of "partial interoperability" is to be expected to persist for some time, and so interferes
3102 with formulating a "clean" algorithm for deciding on what is sufficient for interoperability.

3103

3104 In summary, the previous considerations indicate that at the present it is at best premature to seek
3105 a simple algorithm for *CPA* formation from *CPPs*. It is to be expected that as capability
3106 characterization and exchange becomes a more refined subject, that advances will be made in
3107 characterizing *CPA* formation and negotiation.

3108

3109 Despite it being too soon to propose a simple algorithm for *CPA* formation that covers all the
3110 above variations, it is currently possible to enumerate the basic tasks involved in matching
3111 capabilities within *CPPs*. This information might assist the software implementer in designing a
3112 partially automated and partially interactive software system useful for configuring business
3113 collaboration so as to arrive at satisfactorily complete levels of interoperability. To understand
3114 the context for characterizing the constituent tasks, the general perspective on *CPPs* and *CPAs*
3115 needs to be briefly recalled.

3116
3117

3118 *CPA* Formation Component Tasks

3119

3120 Technically viewed, a *CPA* provides “bindings” between *Business- Process* (BP) specifications
3121 (as defined in the *Process-Specification* document) and those services and protocols that are used
3122 to implement these BP specifications. The implementation takes place at several levels and
3123 involves varied services at these levels. A *CPA* that arrives at a fully interoperable binding of a
3124 BP to its implementing services and protocols can be thought of as arriving at interoperable,
3125 application-to-application integration. *CPAs* MAY fall short of this goal and still be useful and
3126 acceptable to the collaborating *Parties*. Certainly, if no matching data-transport capabilities can
3127 be discovered, a *CPA* would not provide much in the way of interoperable business-to-business
3128 integration. Likewise, partial *CPAs* will leave significant system work to be done before a
3129 completely satisfactory application-to-application integration is realized. Even so, partial
3130 integration MAY be sufficient to allow collaboration, and to enjoy payoffs from increased levels
3131 of automation.

3132

3133 In practice, the *CPA* formation process MAY produce a complete *CPA*, a failure result, a gap list
3134 that drives a dialog with the user, or perhaps even a *CPA* that implements partial interoperability
3135 “good enough” for the business collaborators. Because both matching capabilities and
3136 interoperability can be matters of degree, the constituent tasks are finding the matches in
3137 capabilities at different levels and for different services. We next proceed to characterize many
3138 of these constituent tasks.

3139

3140

3141 *CPA* Formation from *CPPs*: Enumeration of Tasks

3142

3143 To simplify discussion, assume in the following that we are viewing the tasks faced by a
3144 software agent when:

- 3145 1. an intended collaborator is known and the collaborator's *CPP* has been retrieved,
- 3146 2. the *Business Process* between us and our intended collaborator has been selected,
- 3147 3. the specific role that our software agent is to play in the BP is known, and
- 3148 4. the capabilities that are to be advertised in our *CPP* are known.

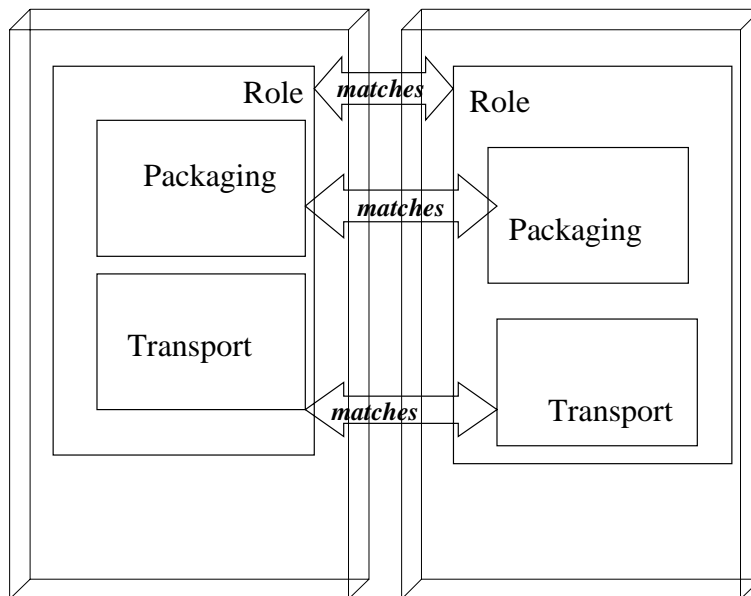
3149

3150 For vividness, we will suppose that our example agent wishes to play the role of supplier and
3151 seeks to find one of its current customers to begin a Purchase Order *Business Process* in which
3152 the intended player plays a complementary role. For simplicity, we assume that the information
3153 about capabilities is restricted to what is available in our agent's *CPP* and in the *CPP* of its
3154 intended collaborator.

3155
 3156 In general, the constituent tasks consist of finding “matches” between our capabilities and our
 3157 intended collaborator’s at the various levels of the protocol stacks and with respect to the
 3158 services supplied at these various levels.

3159
 3160 Figure 6 illustrates the basic tasks informing a *CPA* from two *CPPs*: matching roles, matching
 3161 packaging, and matching transport.
 3162

Figure 6: Basic Tasks in Forming a CPA



3163
 3164 The first task to be considered is certainly the most basic: finding that our intended collaborator
 3165 and ourselves have complementary role capabilities.

3166
 3167

3168 Matching Roles

3169
 3170 Our agent has its role already selected in the BP. So it now begins to check the *Role elements* in
 3171 its collaborator’s *CPP*. The first element to examine is the *PartyInfo* element that contains a
 3172 subtree of elements called *CollaborationRole*. This set is searched to discover a role that
 3173 complements the role of our agent within the BP that we have chosen. For simple binary
 3174 collaboration cases, it is typically sufficient to find that our intended collaborator’s
 3175 *CollaborationRole* set contains *ProcessSpecification* elements that we intend to implement and
 3176 where the role is not identical to our role. For more general collaborations, we would need to
 3177 know the list of roles available within the process, and keep track that for each of the

3178 collaborators, the roles chosen instantiate those that have been specified within the *Process-*
3179 *Specification* document. Collaborations involving more than two roles are not discussed further.

3180
3181

3182 Matching Transport

3183

3184 We now have available a list of candidate *CollaborationRole* elements with the desired
3185 *ProcessSpecification* element (Purchase Ordering) and where our intended collaborator plays the
3186 buyer role. For simplicity, we shall suppose just one *CollaborationRole* element meets these
3187 conditions within each of the relevant *CPPs* and not discuss iterating over lists. (Within these
3188 remarks, where repetition is possible, we will frame the discussion by assuming that just one
3189 element is present.)

3190

3191 Matching transport first means matching the *SendingProtocol* capabilities of our intended
3192 collaborator with the *ReceivingProtocol* capabilities found on our side. Perusal of the *CPP* DTD
3193 or Schema will reveal that the *ServiceBinding* element provides the doorway to the relevant
3194 information from each side's *CollaborationRole* element with the *channelId* attribute. This
3195 *channelId* attribute's value allows us to find *DeliveryChannels* within each *CPP*. The
3196 *DeliveryChannel* has a *transportId* attribute that allows us to find the relevant *Transport*
3197 subtrees.

3198

3199 For example, suppose that our intended buyer has a *Transport* entry:

3200

```
3201 <Transport transportId = "buyerid001">
3202     <SendingProtocol>HTTP</SendingProtocol>
3203     <ReceivingProtocol>
3204     HTTP
3205     </ReceivingProtocol>
3206     <Endpoint uri = "https://www.buyername.com/po-response"
3207         type = "allPurpose"/>
3208     <TransportSecurity>
3209         <Protocol version = "1.0">TLS</Protocol>
3210         <CertificateRef certId = certid001">BuyerName</CertificateRef>
3211     </TransportSecurity>
3212 </Transport>
```

3213

3214 and our seller has a *Transport* entry:

3215

```
3216 <Transport transportId = "sellid001">
3217     <SendingProtocol>HTTP</SendingProtocol>
3218     <ReceivingProtocol>
3219     HTTP
3220     </ReceivingProtocol>
3221     <Endpoint uri = "https://www.sellername.com/pos_here"
3222         type = "allPurpose"/>
3223     <TransportSecurity>
3224         <Protocol version = "1.0">TLS</Protocol>
3225         <CertificateRef certId = "certid002">Sellername</CertificateRef>
3226     </TransportSecurity>
3227 </Transport>
```

3228

3229 A transport match for requests involves finding the initiator role or buyer has a *SendingProtocol*
3230 that matches one of our *ReceivingProtocols*. So here, “HTTP” provides a match. A transport
3231 match for responses involves finding the responder role or seller has a *SendingProtocol* that
3232 matches one of the buyer’s *ReceivingProtocols*. So in the above example, “HTTP” again
3233 provides a match. When such matches exist, we then have discovered an interoperable solution at
3234 the transport level. If not, no *CPA* will be available, and a high-priority gap has been identified
3235 that will need to be remedied by whatever exception handling procedures are in place.
3236
3237

3238 Matching Transport Security

3239
3240 Matches in transport security, such as in the above, will reflect agreement in versions and values
3241 of protocols. Software can supply some knowledge here so that if one side has SSL-3 and the
3242 other TLS-1, it can guess that security is available by means of a fallback of TLS to SSL.
3243
3244

3245 Matching Document Packaging

3246
3247 Probably one of the most complex matching problems arises when it comes to finding whether
3248 there are matches in document-packaging capabilities. Here both security and other MIME
3249 handling capabilities can combine to create complexity for appraising whether full
3250 interoperability can be attained.
3251

3252 Access to the information needed for undertaking this task is found under the *ServiceBinding*
3253 elements, and again we suppose that each side has just one *ServiceBinding* element. However,
3254 we will initially suppose that two *Packaging* elements are available to consider under each role.
3255 Several quite different ways of thinking about the matching task are available, and several
3256 methods for the tasks MAY be performed when assessing whether a good enough match exists.
3257

3258 To continue our previous purchase-ordering example, we recall that the packaging is the
3259 particular combination of body parts, XML instances (*Headers* and payloads), and security
3260 encapsulations used in assembling the *Message* from its data sources. Both requests and
3261 responses will have packaging. The most complete specification of packaging, which MAY not
3262 always be needed, would consist of:

- 3263
3264 1. the buyer asserting what packaging it can generate for its purchase order, and what
3265 packaging it can parse for its purchase order response *Messages*.
3266 2. the seller asserting what packaging it can generate for its purchase order responses and
3267 what packaging it can parse for received purchase orders.
3268

3269 Matching by structural comparison would then involve comparing the packaging details of the
3270 purchase orders generated by the seller with the purchase orders parsable by the buyer. The
3271 comparison would seek to establish that the MIME types of the *SimpleParts* of corresponding
3272 subtrees match and would then proceed to check that the *CompositeList* matched in MIME types
3273 and in sequence of composition.
3274

3275 For example, if each *CPP* contained the packaging subtrees below, and under the appropriate

3276 **ServiceBindings**, then there would be a straightforward match by structural comparison:

```

3277
3278 <Packaging>
3279     <ProcessingCapabilities parse = "true" generate = "true"/>
3280     <SimplePart id = "P1" mimetype = "application/vnd.eb+xml"/>
3281     <SimplePart id = "P2" mimetype = "application/po+xml"/>
3282     <CompositeList>
3283         <Composite mimetype = "multipart/related" id = "P3"
3284             mimeparameters = "type=application/eb+xml">
3285             <Constituent idref = "P1"/>
3286             <Constituent idref = "P2"/>
3287         </Composite>
3288     </CompositeList>
3289 </Packaging>
3290 <Packaging>
3291     <ProcessingCapabilities parse = "true" generate = "true"/>
3292     <SimplePart id = "P11" mimetype = "application/vnd.eb+xml"/>
3293     <SimplePart id = "P12" mimetype = "application/po-ack+xml"/>
3294     <CompositeList>
3295         <Composite mimetype = "multipart/related" id = "P13"
3296             mimeparameters = "type=application/eb+xml">
3297             <Constituent idref = "P11"/>
3298             <Constituent idref = "P12"/>
3299         </Composite>
3300     </CompositeList>
3301 </Packaging>

```

3302
3303 However, it is to be expected that over time it might become possible to only assert what
3304 packaging is *generated* within each **ServiceBinding** for the requester and responder roles. This
3305 simplification assumes that each side has knowledge of what MIME types it handles correctly,
3306 what encapsulations it handles correctly, and what composition modes it handles correctly. By
3307 scanning the packaging specifications against its lists of internal capabilities, it can then look up
3308 whether other side's generated packaging scheme is one it can process and accept it under those
3309 conditions. Knowing what generated packaging style was produced by the other side could
3310 enable the software agent to propose a packaging scheme using only the MIME types and
3311 packaging styles used in the incoming *Message*. Such a packaging scheme would be likely to be
3312 acceptable to the other side when included within a proposed *CPA*. Over time, and as proposal
3313 and negotiation conventions get established, it is to be expected that the methods used for
3314 determining a match in packaging capabilities will move away from structural comparison to
3315 simpler methods, using more economical representations.

3316
3317 In the near term, however, more explicit specifications and the more elaborate structural
3318 comparisons will be most likely to give trustworthy matching assessments.

3319
3320

3321 Matching Document-Level Security

3322
3323 Although the matching task for document-level security is a subtask of the Packaging-matching
3324 task, it is useful to discuss some specifics tied to the three major document-level security
3325 approaches found in [S/MIME], OpenPGP[RFC2015], and XMLDsig[XMLDSIG].

3326

3327 XMLDsig matching capability can be inferred from document-matching capabilities when the
3328 use of ebXML *Message Service*[MSSPEC] packaging is present. However, there are other
3329 sources that should be checked to confirm this match. The *DeliveryChannel* element has a
3330 subtree under the *DocExchange* element that, for the *ebXMLBinding* element, has a
3331 *NameSpacesSupported* element. XMLDsig capability should be found there. Likewise, a
3332 detailed check on this match should examine the information under the *NonRepudiation* element
3333 to check for compatibility in hash functions and algorithms.
3334

3335 The existence of several radically different approaches to document-level security, together with
3336 the fact that it is unusual at present for a given *Party* to commit to more than one form of such
3337 security, means that there can be basic failures to match security frameworks. Therefore, there
3338 might be no match in capabilities that supports full interoperability at all levels. For the moment,
3339 we assume that document-level security matches will require both sides able to handle the same
3340 security composites (multipart/signed using S/MIME, for example.)
3341

3342 However, suppose that there are matches at the transport and transport layer security levels, but
3343 that the two sides have failures at the document-security layer because one side makes use of
3344 PGP signatures while the other uses S/MIME. Does this mean that no *CPA* can be proposed?
3345 That is not necessarily the case.
3346

3347 Both S/MIME and OpenPGP permit signatures to be packaged within “multipart/signed”
3348 composites. In such a case, it MAY be possible to extract the data and arrive at a partial
3349 implementation that falls short with respect to nonrepudiation. While neither side could check
3350 the other's signatures, it might still be possible to have confidential document transmission and
3351 transport-level authentication for the business data. Eventually *CPA*-formation software MAY be
3352 created that is able to identify these exceptional situations and “salvage” a proposed *CPA* with
3353 downgraded security features. Whether the other side would accept such a proposed *CPA* would,
3354 naturally, involve what their preferences are with respect to initiating a business collaboration
3355 and sacrificing some security features. *CPA*-formation software MAY eventually be capable of
3356 these adaptations, but it is to be expected that human assistance will be required for such
3357 situations in the near term.
3358

3359 Of course, an implementation MAY simply decide to terminate looking for a *CPA* when a match
3360 fails in any crucial factor for an interoperable implementation. At the very least, the users should
3361 be warned that the only *CPAs* that can be proposed will be missing security or other normally
3362 desirable features or features recommended by the BP's *Process Specification*.
3363
3364

3365 Other Considerations

3366
3367 Handling Preferences among multiple matching capabilities involves
3368

- 3369 1. Preferences: tiebreaker needed.
- 3370 2. Ranking: one might convert ranks to numerical order, add values, and decide that lowest
3371 value wins; in case of a tie, the choice is the lowest value that reflects the BP responder
3372 values.