



Creating A Single Global Electronic Market

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

Collaboration-Protocol Profile and Agreement Specification

Version 1.~~0506~~0506

OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee

~~24 January~~ 4 February 2002

1 Status of this Document

This document specifies an ebXML SPECIFICATION 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.oasis-open.org/committees/ebxml-cppa/documents/working_drafts/ebCPP-1_0506.pdf

Previous version:

<http://www.ebxml.org/specs/ebCCP.pdf>

34 2 Technical Committee Members

35 Selem Aissi (Voting Membership Pending as of San Francisco F2F)

36 Arvola Chan

37 James Bryce Clark

38 David Fischer (Voting Membership Pending as of San Francisco F2F)

39 Tony Fletcher

40 Brian Hayes

41 Neelakantan Kartha

42 Kevin Liu

43 Pallavi Malu

44 Dale Moberg

45 Himagiri Mukkamala

46 Peter Ogden

47 Marty Sachs

48 Yukinori Saito

49 David Smiley

50 Tony Weida

51 Pete Wenzel

52 Jean Zheng

53 Sinisa Zimek

54

55 3 ebXML Participants

56 The authors wish to recognize the following for their significant ~~participation to the development~~
57 ~~of this document~~ in developing the Collaboration Protocol Profile and Agreement Specification,
58 Version 1.0.

59
60
61 David Burdett, CommerceOne
62 Tim Chiou, United World Chinese Commercial Bank
63 Chris Ferris, Sun
64 Scott Hinkelman, IBM
65 Maryann Hondo, IBM
66 Sam Hunting, ECOM XML
67 John Ibbotson, IBM
68 Kenji Itoh, JASTPRO
69 Ravi Kacker, eXcelon Corp.
70 Thomas Limanek, iPlanet
71 Daniel Ling, VCHEQ
72 Henry Lowe, OMG
73 Dale Moberg, Cyclone Commerce
74 Duane Nickull, XMLGlobal Technologies
75 Stefano Pogliani, Sun
76 Rebecca Reed, Mercator
77 Karsten Riemer, Sun
78 Marty Sachs, IBM
79 Yukinori Saito, ECOM
80 Tony Weida, Edifecs

81

82 **34** Table of Contents

83	<u>1</u>	<u>Status of this Document.....</u>	<u>1</u>
84	<u>2</u>	<u>Technical Committee Members</u>	<u>2</u>
85	<u>3</u>	<u>ebXML Participants</u>	<u>3</u>
86	<u>4</u>	<u>Table of Contents.....</u>	<u>4</u>
87	<u>5</u>	<u>Introduction.....</u>	<u>9</u>
88		<u>5.1 Summary of Contents of Document</u>	<u>9</u>
89		<u>5.2 Document Conventions.....</u>	<u>9</u>
90		<u>5.3 Version of the Specification.....</u>	<u>10</u>
91		<u>5.4 Definitions.....</u>	<u>10</u>
92		<u>5.5 Audience.....</u>	<u>10</u>
93		<u>5.6 Assumptions.....</u>	<u>10</u>
94		<u>5.7 Related Documents.....</u>	<u>11</u>
95	<u>6</u>	<u>Design Objectives.....</u>	<u>12</u>
96	<u>7</u>	<u>System Overview</u>	<u>13</u>
97		<u>7.1 What This Specification Does</u>	<u>13</u>
98		<u>7.2 Forming a CPA from Two CPPs.....</u>	<u>14</u>
99		<u>7.3 How the CPA Works.....</u>	<u>17</u>
100		<u>7.4 Where the CPA May Be Implemented.....</u>	<u>18</u>
101		<u>7.5 Definition and Scope.....</u>	<u>18</u>
102	<u>8</u>	<u>CPP Definition</u>	<u>19</u>
103		<u>8.1 Globally-Unique Identifier of CPP Instance Document</u>	<u>20</u>
104		<u>8.2 SchemaLocation Attribute.....</u>	<u>20</u>
105		<u>8.3 CPP Structure</u>	<u>20</u>
106		<u>8.4 CollaborationProtocolProfile element.....</u>	<u>21</u>
107		<u>8.5 PartyInfo Element.....</u>	<u>22</u>
108		<u>8.5.1 PartyId element</u>	<u>23</u>
109		<u>8.5.2 PartyRef element.....</u>	<u>24</u>
110		<u>8.5.3 CollaborationRole element.....</u>	<u>25</u>
111		<u>8.5.4 ProcessSpecification element</u>	<u>27</u>
112		<u>8.5.5 Role element</u>	<u>31</u>
113		<u>8.5.6 ApplicationCertificateRef element</u>	<u>31</u>
114		<u>8.5.7 ApplicationSecurityDetailsRef element.....</u>	<u>32</u>
115		<u>8.5.8 ServiceBinding element.....</u>	<u>32</u>
116		<u>8.5.9 Service element</u>	<u>34</u>
117		<u>8.5.10 WillInitiate element</u>	<u>35</u>
118		<u>8.5.11 WillRespond element.....</u>	<u>35</u>
119		<u>8.5.12 ThisPartyActionBinding element and OtherPartyActionBinding element.....</u>	<u>36</u>
120		<u>8.5.13 ActionContext element</u>	<u>37</u>
121		<u>8.5.14 CollaborationActivity element</u>	<u>38</u>
122		<u>8.5.15 Certificate element</u>	<u>39</u>
123		<u>8.5.16 SecurityDetails element.....</u>	<u>39</u>
124		<u>8.5.17 TrustAnchors element.....</u>	<u>40</u>
125		<u>8.5.18 SecurityPolicy element</u>	<u>40</u>
126		<u>8.5.19 DeliveryChannel element.....</u>	<u>40</u>
127		<u>8.5.20 BusinessProcessCharacteristics element</u>	<u>42</u>
128		<u>8.5.21 MessagingCharacteristics element.....</u>	<u>44</u>
129		<u>8.5.22 Transport element.....</u>	<u>46</u>
130		<u>8.5.23 TransportSender element.....</u>	<u>48</u>
131		<u>8.5.24 TransportProtocol element.....</u>	<u>48</u>
132		<u>8.5.25 TransportClientSecurity element</u>	<u>48</u>
133		<u>8.5.26 TransportSecurityProtocol element</u>	<u>49</u>

134	8.5.27 ClientCertificateRef element	49
135	8.5.28 ServerSecurityDetailsRef element	49
136	8.5.29 TransportReceiver element	49
137	8.5.30 Endpoint element	5049
138	8.5.31 TransportServerSecurity element	50
139	8.5.32 ServerCertificateRef element	50
140	8.5.33 ClientSecurityDetailsRef element	5150
141	8.5.34 Transport protocols	51
142	8.5.35 DocExchange Element	53
143	8.5.36 ebXMLSenderBinding element	54
144	8.5.37 ReliableMessaging element	55
145	8.5.38 Retries and RetryInterval elements	55
146	8.5.39 PersistDuration element	5655
147	8.5.40 MessageOrderSemantics element	5655
148	8.5.41 SenderNonRepudiation element	56
149	8.5.42 NonRepudiationProtocol element	5756
150	8.5.43 HashFunction element	5756
151	8.5.44 SignatureAlgorithm element	57
152	8.5.45 SigningCertificateRef element	57
153	8.5.46 SenderDigitalEnvelope element	57
154	8.5.47 DigitalEnvelopeProtocol element	5857
155	8.5.48 EncryptionAlgorithm element	5857
156	8.5.49 EncryptionSecurityDetailsRef element	5857
157	8.5.50 NamespaceSupported element	58
158	8.5.51 ebXMLReceiverBinding element	58
159	8.5.52 ReceiverNonRepudiation element	5958
160	8.5.53 SigningSecurityDetailsRef element	59
161	8.5.54 ReceiverDigitalEnvelope element	59
162	8.5.55 EncryptionCertificateRef element	6059
163	8.5.56 OverrideMshActionBinding element	6059
164	8.6 SimplePart element	60
165	8.7 Packaging element	6160
166	8.7.1 ProcessingCapabilities element	61
167	8.7.2 CompositeList element	6261
168	8.8 ds:Signature element	6362
169	8.9 Comment element	6463
170	9 CPA Definition	6564
171	9.1 CPA Structure	6564
172	9.2 CollaborationProtocolAgreement element	6665
173	9.3 Status Element	6765
174	9.4 CPA Lifetime	6766
175	9.4.1 Start element	6766
176	9.4.2 End element	6766
177	9.5 ConversationConstraints Element	6867
178	9.5.1 invocationLimit attribute	6867
179	9.5.2 concurrentConversations attribute	6867
180	9.6 PartyInfo Element	6968
181	9.6.1 ProcessSpecification element	6968
182	9.7 SimplePart element	6968
183	9.8 Packaging element	6968
184	9.9 ds:Signature element	6968
185	9.9.1 Persistent Digital Signature	7069
186	9.10 Comment element	7271
187	9.11 Composing a CPA from Two CPPs	7271
188	9.11.1 ID Attribute Duplication	7271

189	9.12 Modifying Parameters of the Process-Specification Document Based on Information in the CPA	7372
190	10 References	7473
191	11 Conformance	7776
192	12 Disclaimer	7877
193	13 Contact Information	7978
194	Notices	8079
195	Copyright Statement	8180
196	Appendix A Example of CPP Document (Non-Normative)	8281
197	Appendix B Example of CPA Document (Non-Normative)	8382
198	Appendix C Business Process Specification Corresponding to Complete CPP/CPA Definition (Non-	
199	Normative)	8483
200	Appendix D W3C XML Schema Document Corresponding to Complete CPP and CPA Definition (Normative)	
201		8685
202	Appendix E Formats of Information in the CPP and CPA (Normative)	8786
203	Appendix F Composing a CPA from Two CPPs (Non-Normative)	8887
204	Appendix G Correspondence Between CPA and ebXML Messaging Parameters (Normative)	9796
205	Appendix H Glossary of Terms	10099
206	1 Status of this Document	1
207	2 ebXML Participants	2
208	3 Table of Contents	3
209	4 Introduction	8
210	4.1 Summary of Contents of Document	8
211	4.2 Document Conventions	8
212	4.3 Version of the Specification	9
213	4.4 Definitions	9
214	4.5 Audience	9
215	4.6 Assumptions	9
216	4.7 Related Documents	9
217	5 Design Objectives	11
218	6 System Overview	12
219	6.1 What This Specification Does	12
220	6.2 Forming a CPA from Two CPPs	13
221	6.3 How the CPA Works	16
222	6.4 Where the CPA May Be Implemented	17
223	6.5 Definition and Scope	17
224	7 CPP Definition	18
225	7.1 Globally Unique Identifier of CPP Instance Document	19
226	7.2 SchemaLocation Attribute	19
227	7.3 CPP Structure	19
228	7.4 CollaborationProtocolProfile element	20
229	7.5 PartyInfo Element	21
230	7.5.1 PartyId element	22
231	7.5.2 PartyRef element	23
232	7.5.3 CollaborationRole element	24
233	7.5.4 ProcessSpecification element	26
234	7.5.5 Role element	29
235	7.5.6 ApplicationCertificateRef element	30
236	7.5.7 ApplicationSecurityDetailsRef element	30
237	7.5.8 ServiceBinding element	31
238	7.5.9 Service element	32
239	7.5.10 WillInitiate element	33
240	7.5.11 WillRespond element	33
241	7.5.12 ThisPartyActionBinding element and OtherPartyActionBinding element	33
242	7.5.13 ActionContext element	35
243	7.5.14 CollaborationActivity element	36

244	7.5.15 Certificate element	36
245	7.5.16 SecurityDetails element	37
246	7.5.17 TrustAnchors element	38
247	7.5.18 SecurityPolicy element	38
248	7.5.19 DeliveryChannel element	38
249	7.5.20 BusinessProcessCharacteristics element	40
250	7.5.21 MessagingCharacteristics element	41
251	7.5.22 Transport element	44
252	7.5.23 TransportSender element	45
253	7.5.24 TransportProtocol element	46
254	7.5.25 TransportClientSecurity element	46
255	7.5.26 TransportSecurityProtocol element	46
256	7.5.27 ClientCertificateRef element	46
257	7.5.28 ServerSecurityDetailsRef element	47
258	7.5.29 TransportReceiver element	47
259	7.5.30 Endpoint element	47
260	7.5.31 TransportServerSecurity element	48
261	7.5.32 ServerCertificateRef element	48
262	7.5.33 ClientSecurityDetailsRef element	48
263	7.5.34 Transport protocols	48
264	7.5.35 DocExchange Element	51
265	7.5.36 ebXML SenderBinding element	52
266	7.5.37 ReliableMessaging element	52
267	7.5.38 Retries and RetryInterval elements	53
268	7.5.39 PersistDuration element	53
269	7.5.40 MessageOrderSemantics element	53
270	7.5.41 SenderNonRepudiation element	54
271	7.5.42 NonRepudiationProtocol element	54
272	7.5.43 HashFunction element	54
273	7.5.44 SignatureAlgorithm element	54
274	7.5.45 SigningCertificateRef element	55
275	7.5.46 SenderDigitalEnvelope element	55
276	7.5.47 DigitalEnvelopeProtocol element	55
277	7.5.48 EncryptionAlgorithm element	55
278	7.5.49 EncryptionSecurityDetailsRef element	55
279	7.5.50 NamespaceSupported element	56
280	7.5.51 ebXML ReceiverBinding element	56
281	7.5.52 ReceiverNonRepudiation element	56
282	7.5.53 SigningSecurityDetailsRef element	57
283	7.5.54 ReceiverDigitalEnvelope element	57
284	7.5.55 EncryptionCertificateRef element	57
285	7.5.56 OverrideMshActionBinding element	57
286	7.6 SimplePart element	58
287	7.7 Packaging element	58
288	7.7.1 ProcessingCapabilities element	59
289	7.7.2 SimplePart element	59
290	7.7.3 CompositeList element	59
291	7.8 ds:Signature element	60
292	7.9 Comment element	61
293	8 CPA Definition	62
294	8.1 CPA Structure	62
295	8.2 CollaborationProtocolAgreement element	63
296	8.3 Status Element	64
297	8.4 CPA Lifetime	64
298	8.4.1 Start element	64

299	<u>8.4.2 End element</u>	64
300	<u>8.5 ConversationConstraints Element</u>	65
301	<u>8.5.1 invocationLimit attribute</u>	65
302	<u>8.5.2 concurrentConversations attribute</u>	65
303	<u>8.6 PartyInfo Element</u>	66
304	<u>8.6.1 ProcessSpecification element</u>	66
305	<u>8.7 Packaging element</u>	66
306	<u>8.8 ds:Signature element</u>	66
307	<u>8.8.1 Persistent Digital Signature</u>	67
308	<u>8.9 Comment element</u>	69
309	<u>8.10 Composing a CPA from Two CPPs</u>	69
310	<u>8.10.1 ID Attribute Duplication</u>	69
311	<u>8.11 Modifying Parameters of the Process Specification Document Based on Information in the CPA</u>	70
312	<u>9 References</u>	71
313	<u>10 Conformance</u>	73
314	<u>11 Disclaimer</u>	74
315	<u>12 Contact Information</u>	75
316	<u>Notices</u>	76
317	<u>Copyright Statement</u>	77
318	<u>Appendix A Example of CPP Document (Non-Normative)</u>	78
319	<u>Appendix B Example of CPA Document (Non-Normative)</u>	79
320	<u>Appendix C Business Process Specification Corresponding to Complete CPP/CPA Definition (Non-</u>	
321	<u>Normative)</u>	80
322	<u>Appendix D W3C XML Schema Document Corresponding to Complete CPP and CPA Definition (Normative)</u>	
323	<u>.....</u>	82
324	<u>Appendix E Formats of Information in the CPP and CPA (Normative)</u>	83
325	<u>Appendix F Composing a CPA from Two CPPs (Non-Normative)</u>	84
326	<u>Appendix G Correspondence Between CPA and ebXML Messaging Parameters (Normative)</u>	93
327	<u>Appendix H Glossary of Terms</u>	96
328		

329 4.5 Introduction

330

331 4.15.1 Summary of Contents of Document

332

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

340

341 The *Message-exchange* capabilities of a *Party* MAY be described by a *Collaboration-Protocol*
342 *Profile (CPP)* ~~within the TPP~~. The *Message-exchange* agreement between two *Parties* MAY be
343 described by a *Collaboration-Protocol Agreement (CPA)* ~~within the TPA~~. A CPA MAY be
344 created by computing the intersection of the two Partners' CPPs. Included in the *CPP* and *CPA*
345 are details of transport, messaging, security constraints, and bindings to a *Business-Process-*
346 *Specification* (or, for short, *Process-Specification*) document that contains the definition of the
347 interactions between the two *Parties* while engaging in a specified electronic *Business*
348 *Collaboration*.

349

350 This specification contains the detailed definitions of the *Collaboration-Protocol Profile (CPP)*
351 and the *Collaboration-Protocol Agreement (CPA)*.

352

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

356

357 This specification is organized as follows:

358

- Section 65 defines the objectives of this specification.
- Section 76 provides a system overview.
- Section 87 contains the definition of the *CPP*, identifying the structure and all necessary fields.
- Section 98 contains the definition of the *CPA*.
- The appendices include examples of ~~an XML Business Process Specification (non-~~
364 ~~normative),~~ *CPP* and *CPA* documents (non-normative), an example XML Business
365 Process Specification (non-normative), an XML Schema document (normative),
366 formats of information in the *CPP* and *CPA* (normative), ~~and~~ composing a *CPA* from
367 two *CPPs* (non-normative) and a Glossary of Terms.

368

369 4.25.2 Document Conventions

370 Terms in *Italics* are defined in Appendix H ~~Appendix H~~ ~~Appendix H~~ (Glossary of Terms).
Collaboration-Protocol Profile and Agreement Specification

371 Terms listed in ***Bold Italics*** represent the element and/or attribute content of the XML *CPP*, ~~CPA~~
372 *CPA*, or related definitions.

373

374 In this specification, indented paragraphs beginning with "NOTE:" provide non-normative
375 explanations or suggestions that are not mandated by the specification.

376

377 References to external documents are represented with BLOCK text enclosed in brackets, e.g.
378 [RFC2396]. The references are listed in Section [109](#), "References".

379

380 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD
381 NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be
382 interpreted as described in [RFC 2119]. They In this document, they appear in all capital letters
383 to emphasize their usage as keywords, as do the following [XML] keywords ID, IDREF,
384 IMPLIED, FIXED.

385

386 NOTE: Vendors SHOULD carefully consider support of elements with cardinalities (0 or
387 1) or (0 or more). Support of such an element means that the element is processed
388 appropriately for its defined function and not just recognized and ignored. A given *Party*
389 might use these elements in some *CPPs* or *CPAs* and not in others. Some of these elements
390 define parameters or operating modes and SHOULD be implemented by all vendors. It
391 might be appropriate to implement elective elements that represent major run-time
392 functions, such as various alternative communication protocols or security functions, by
393 means of plug-ins so that a given *Party* MAY acquire only the needed functions rather than
394 having to install all of them.

395

396 **4.35.3 Version of the Specification**

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

400

401 **4.45.4 Definitions**

402 Technical terms in this specification are defined in [Appendix H](#) ~~Appendix H~~ [H](#).

403

404 **4.55.5 Audience**

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

409

410 **4.65.6 Assumptions**

411 It is expected that the reader has an understanding of XML and is familiar with the concepts of
412 electronic *Business* (eBusiness).

413

414 **4.75.7 Related Documents**

415 Related documents include ebXML Specifications on the following topics:

- 416 • ~~ebXML Technical Architecture Specification[ebTA]~~
- 417 • ebXML *Message* Service Specification[ebMS]
- 418 • ebXML Business Process Specification Schema[ebBPSS]
- 419 • ebXML Core Component and Business Document Overview[ccOVER]
- 420 • ebXML Registry Services Specification[ebRS]

421

422 See Section [109](#) for the complete list of references.

423

424 **56** Design Objectives

425 The objective of this specification is to ensure interoperability between two *Parties* even though
426 they MAY procure application software and run-time support software from different vendors.
427 The *CPP* defines a *Party's Message*-exchange capabilities and the *Business Collaborations* that
428 it supports. The *CPA* defines the way two *Parties* will interact in performing the chosen *Business*
429 *Collaboration*. Both *Parties* SHALL use identical copies of the *CPA* to configure their run-time
430 systems. This assures that they are compatibly configured to exchange *Messages* whether or not
431 they have obtained their run-time systems from the same vendor. The configuration process
432 MAY be automated by means of a suitable tool that reads the *CPA* and performs the
433 configuration process.

434
435 In addition to supporting direct interaction between two *Parties*, this specification MAY also be
436 used to support interaction between two *Parties* through an intermediary such as a portal or
437 broker.

438
439 It is an objective of this specification that a *CPA* SHALL be capable of being composed by
440 intersecting the respective *CPPs* of the *Parties* involved. The resulting *CPA* SHALL contain
441 only those elements that are in common, or compatible, between the two *Parties*. Variable
442 quantities, such as number of retries of errors, are then negotiated between the two *Parties*. The
443 design of the *CPP* and *CPA* schemata facilitates this composition/negotiation process. However,
444 the composition and negotiation processes themselves are outside the scope of this specification.
445 [Appendix F](#) ~~Appendix F~~ ~~Appendix F~~ contains a non-normative discussion of this subject.

446
447 It is a further objective of this specification to facilitate migration of both traditional EDI-based
448 applications and other legacy applications to platforms based on the ebXML specifications. In
449 particular, the *CPP* and *CPA* are components of the migration of applications based on the X12
450 838 Trading-Partner Profile [\[X12\]](#) to more automated means of setting up *Business* relationships
451 and doing *Business* under them.

452 **6.7** System Overview

453 **6.17.1** What This Specification Does

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

458
459 The way each *Party* can exchange information, in the context of a *Business Collaboration*, can
460 be described by a *Collaboration-Protocol Profile (CPP)*. The agreement between the *Parties* can
461 be expressed as a *Collaboration-Protocol Agreement (CPA)*

462
463 A *Party* MAY describe itself in a single *CPP*. A *Party* MAY create multiple *CPPs* that describe,
464 for example, different *Business Collaborations* that it supports, its operations in different regions
465 of the world, or different parts of its organization.

466
467 To enable *Parties* wishing to do *Business* to find other *Parties* that are suitable *Business*
468 *Partners*, *CPPs* MAY be stored in a repository such as is provided by the ebXML
469 Registry[ebRS]. Using a discovery process provided as part of the specifications of a repository,
470 a *Party* MAY then use the facilities of the repository to find *Business Partners*.

471
472 The document that defines the interactions between two *Parties* is a *Process-Specification*
473 document that MAY conform to the ebXML Business Process Specification Schema[ebBPSS].
474 The *CPP* and *CPA* include references to this *Process-Specification* document. The *Process-*
475 *Specification* document MAY be stored in a repository such as the ebXML Registry. See NOTE
476 about alternative *Business-Collaboration* descriptions in Section [8.5.47.5.4](#).

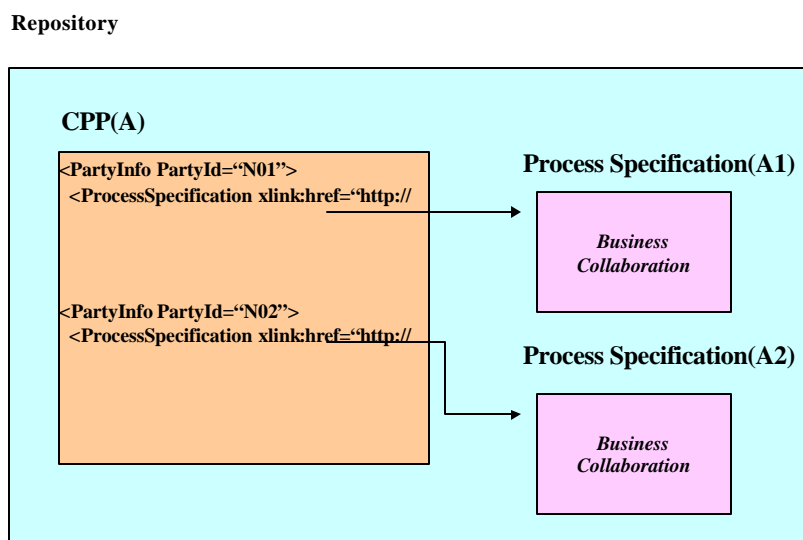
477
478 Figure 1 illustrates the relationships between a *CPP* and two *Process-Specification* documents,
479 A1 and A2, in an ebXML Registry. On the left is a *CPP*, A, which includes information about
480 two parts of an enterprise that are represented as different *Parties*. On the right are shown two
481 *Process-Specification* documents. Each of the **PartyInfo** elements in the *CPP* contains a
482 reference to one of the *Process-Specification* documents. This identifies the *Business*
483 *Collaboration* that the *Party* can perform.

484
485 This specification defines the markup language vocabulary for creating electronic *CPPs* and
486 *CPAs*. *CPPs* and *CPAs* are [XML] documents. In the appendices of this specification are two
487 sample *CPPs*, a sample *CPA* formed from the *CPPs*, a sample *Process-Specification* referenced
488 by the *CPPs* and the *CPA*, and the XML Schema governing the structures of *CPPs* and *CPAs*.

489
490 The *CPP* describes the capabilities of an individual *Party*. A *CPA* describes the capabilities that
491 two *Parties* have agreed to use to perform a particular *Business Collaboration*. These *CPAs*
492 define the "information technology terms and conditions" that enable *Business* documents to be
493 electronically interchanged between *Parties*. The information content of a *CPA* is similar to the
494 information-technology specifications sometimes included in Electronic Data Interchange (EDI)

495 *Trading Partner Agreements (TPAs)*. However, these *CPAs* are not paper documents. Rather,
 496 they are electronic documents that can be processed by computers at the *Parties'* sites in order to
 497 set up and then execute the desired *Business* information exchanges. The "legal" terms and
 498 conditions of a *Business* agreement are outside the scope of this specification and therefore are
 499 not included in the *CPP* and *CPA*.

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



500
 501 An enterprise MAY choose to represent itself as multiple *Parties*. For example, it might
 502 represent a central office supply procurement organization and a manufacturing supplies
 503 procurement organization as separate *Parties*. The enterprise MAY then construct a *CPP* that
 504 includes all of its units that are represented as separate *Parties*. In the *CPP*, each of those units
 505 would be represented by a separate *PartyInfo* element.

506
 507 In general, the *Parties* to a *CPA* can have both client and server characteristics. A client requests
 508 services and a server provides services to the *Party* requesting services. In some applications,
 509 one *Party* only requests services and one *Party* only provides services. These applications have
 510 some resemblance to traditional client-server applications. In other applications, each *Party*
 511 MAY request services of the other. In that case, the relationship between the two *Parties* can be
 512 described as a peer-peer relationship rather than a client-server relationship.

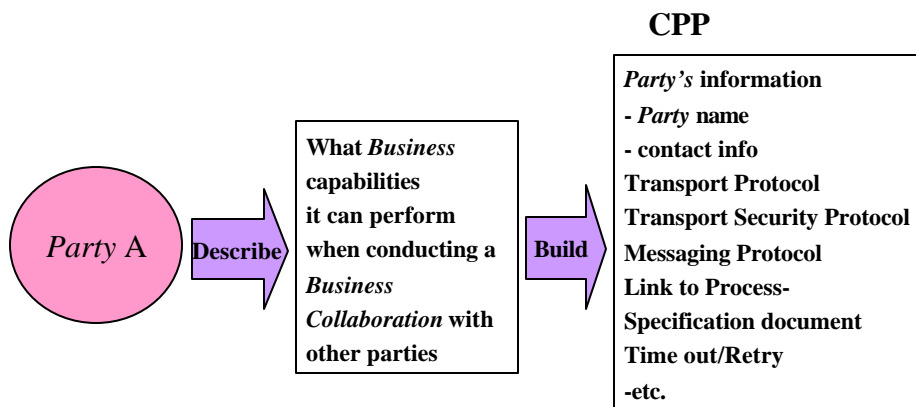
513

514 **6.27.2 Forming a CPA from Two CPPs**

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

519
520 Figure 2 illustrates forming a *CPP*. *Party A* tabulates the information to be placed in a repository
521 for the discovery process, constructs a *CPP* that contains this information, and enters it into an
522 ebXML Registry or similar repository along with additional information about the *Party*. The
523 additional information might include a description of the *Businesses* that the *Party* engages in.
524 Once *Party A*'s information is in the repository, other *Parties* can discover *Party A* by using the

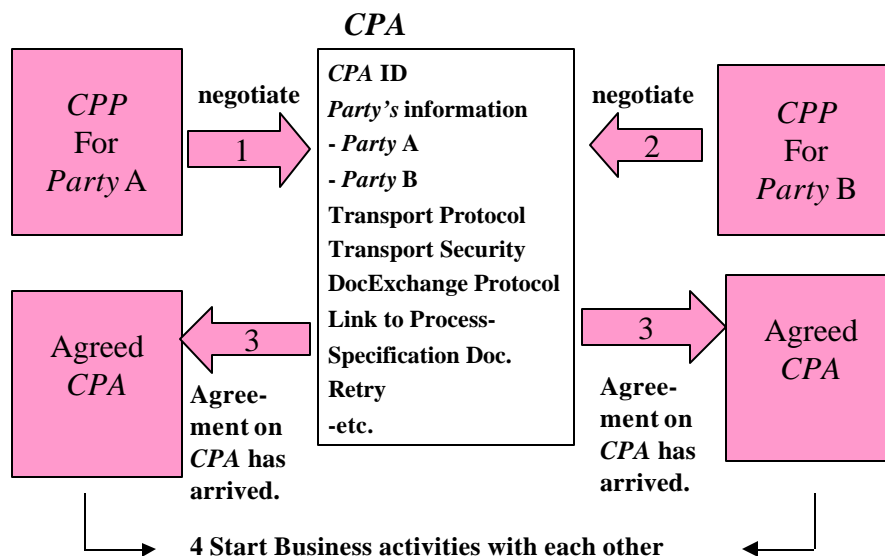
Figure 2: Overview of Collaboration-Protocol Profiles (CPP)



525 repository's discovery services.

526
527 In figure 3, *Party A* and *Party B* use their *CPPs* to jointly construct a single copy of a *CPA* by
528 calculating the intersection of the information in their *CPPs*. The resulting *CPA* defines how the
529 two *Parties* will behave in performing their *Business Collaboration*.

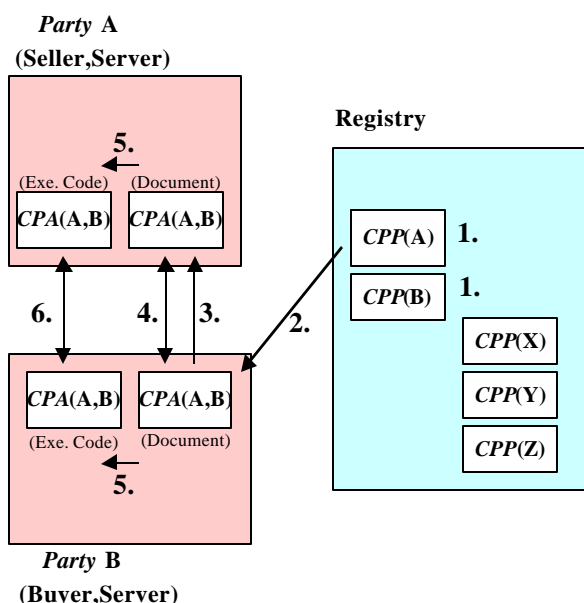
Figure 3: Overview of Collaboration-Protocol Agreements (CPA)



530
 531 Figure 4 illustrates the entire process. The steps are listed at the left. The end of the process is
 532 that the two *Parties* configure their systems from identical copies of the agreed *CPA* and they are

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

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



533 then ready to do *Business*.

534

535 NOTE: This specification makes the assumption that a *CPP* that has been registered in an
536 ebXML or other Registry will be referenced by some Registry-assigned globally-unique
537 identifier that MAY be used to distinguish among multiple *CPPs* belonging to the same
538 *Party*. See Section [8.17.1](#) for more information.

539

540 **6.37.3 How the CPA Works**

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

548

549 The information in the *CPA* is used to configure the *Parties'* systems to enable exchange of
550 *Messages* in the course of performing the selected *Business Collaboration*. Typically, the
551 software that performs the *Messages* exchanges and otherwise supports the interactions between
552 the *Parties* is middleware that can support any selected *Business Collaboration*. One component
553 of this middleware MAY be the ebXML *Message Service Handler*[ebMS]. In this specification,
554 the term "run-time system" or "run-time software" is used to denote such middleware.

555

556 The *CPA* and the *Process-Specification* document that it references define a conversation
557 between the two *Parties*. The conversation represents a single unit of *Business* as defined by the
558 *Binary-Collaboration* component of the *Process-Specification* document. The conversation
559 consists of one or more *Business Transactions*, each of which is a request *Message* from one
560 *Party* and zero or one response *Message* from the other *Party*. The *Process-Specification*
561 document defines, among other things, the request and response *Messages* for each *Business*
562 *Transaction* and the order in which the *Business Transactions* are REQUIRED to occur. See
563 [ebBPSS] for a detailed explanation.

564

565 The *CPA* MAY actually reference more than one *Process-Specification* document. When a *CPA*
566 references more than one *Process-Specification* document, each *Process-Specification* document
567 defines a distinct type of conversation. Any one conversation involves only a single *Process-*
568 *Specification* document.

569

570 A new conversation is started each time a new unit of *Business* is started. The *Business*
571 *Collaboration* also determines when the conversation ends. From the viewpoint of a *CPA*
572 between *Party A* and *Party B*, the conversation starts at *Party A* when *Party A* sends the first
573 request *Message* to *Party B*. At *Party B*, the conversation starts when it receives the first request
574 of the unit of *Business* from *Party A*. A conversation ends when the *Parties* have completed the
575 unit of *Business*.

576

577 NOTE: The run-time system SHOULD provide an interface by which the *Business*
578 application can request initiation and ending of conversations.
579

580 **6.47.4 Where the CPA May Be Implemented**

581 Conceptually, a *Business-to-Business* (B2B) server at each *Party's* site implements the CPA and
582 Process-Specification document. The B2B server includes the run-time software, i.e. the
583 middleware that supports communication with the other *Party*, execution of the functions
584 specified in the *CPA*, interfacing to each *Party's* back-end processes, and logging the interactions
585 between the *Parties* for purposes such as audit and recovery. The middleware might support the
586 concept of a long-running conversation as the embodiment of a single unit of *Business* between
587 the *Parties*. To configure the two *Parties'* systems for *Business to Business* operations, the
588 information in the copy of the *CPA* and *Process-Specification* documents at each *Party's* site is
589 installed in the run-time system. The static information MAY be recorded in a local database and
590 other information in the *CPA* and *Process-Specification* document MAY be used in generating or
591 customizing the necessary code to support the *CPA*.
592

593 NOTE: It is possible to provide a graphical *CPP/CPA*-authoring tool that understands both
594 the semantics of the *CPP/CPA* and the XML syntax. Equally important, the definitions in
595 this specification make it feasible to automatically generate, at each *Party's* site, the code
596 needed to execute the *CPA*, enforce its rules, and interface with the *Party's* back-end
597 processes.
598

599 **6.57.5 Definition and Scope**

600 This specification defines and explains the contents of the *CPP* and *CPA* XML documents. Its
601 scope is limited to these definitions. It does not define how to compose a *CPA* from two *CPPs*
602 nor does it define anything related to run-time support for the *CPP* and *CPA*. It does include
603 some non-normative suggestions and recommendations regarding [CPA composition from two](#)
604 [CPPs and](#) run-time support where these notes serve to clarify the *CPP* and *CPA* definitions. See
605 Section [1140](#) for a discussion of conformance to this specification.
606
607

608 NOTE: This specification is limited to defining the contents of the *CPP* and *CPA*, and it is
609 possible to be conformant with it merely by producing a *CPP* or *CPA* document that
610 conforms to the XML Schema document defined herein. It is, however, important to
611 understand that the value of this specification lies in its enabling a run-time system that
612 supports electronic commerce between two *Parties* under the guidance of the information in
613 the *CPA*.

614 **78** CPP Definition

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

619
620 This section defines and discusses the details in the *CPP* in terms of the individual XML
621 elements. The discussion is illustrated with some XML fragments. See [Appendix D](#) ~~Appendix D~~
622 ~~D~~ for the XML Schema, and [Appendix A](#) ~~Appendix A~~ ~~A~~ for sample *CPP* documents.

623
624 The *ProcessSpecification*, *DeliveryChannel*, *DocExchange*, and *Transport* elements of the
625 *CPP* describe the processing of a unit of *Business* (conversation). These elements form a layered
626 structure somewhat analogous to a layered communication model. ~~The remainder of this section~~
627 ~~describes both the above-mentioned elements and the corresponding run-time processing.~~

628
629 **Process-Specification layer** - The *Process-Specification* layer defines the heart of the *Business*
630 agreement between the *Parties*: the services (*Business Transactions*) which *Parties* to the *CPA*
631 can request of each other and transition rules that determine the order of requests. This layer is
632 defined by the separate *Process-Specification* document that is referenced by the *CPP* and *CPA*.

633
634 **Delivery Channels** - A delivery channel describes a *Party's* *Message*-receiving and *Message*-
635 sending characteristics. It consists of one document-exchange definition and one transport
636 definition. Several delivery channels MAY be defined in one *CPP*.

637
638 ~~**Document-Exchange layer** - The document-exchange layer accepts a *Business* document from~~
639 ~~the *Process-Specification* layer at one *Party*, encrypts it if specified, adds a digital signature for~~
640 ~~non-repudiation if specified, and passes it to the transport layer for transmission to the other~~
641 ~~*Party*. It performs the inverse steps for received *Messages*. The options selected for the~~
642 ~~document-exchange layer are complementary to those selected for the transport layer. For~~
643 ~~example, if *Message* security is desired and the selected transport protocol does not provide~~
644 ~~*Message* encryption, then it MUST be specified at the document-exchange layer. The protocol~~
645 ~~for exchanging *Messages* between two *Parties* is defined by the ebXML *Message Service*~~
646 ~~*Specification* [ebMS] or other similar messaging service.~~

647
648 ~~**Transport layer** - The transport layer is responsible for *Message* delivery using the selected~~
649 ~~transport protocol. The selected protocol affects the choices selected for the document-exchange~~
650 ~~layer. For example, some transport-layer protocols might provide encryption and authentication~~
651 ~~while others have no such facility.~~

652 **Document-Exchange Layer** - The Document-exchange layer specifies processing of the
653 business documents by the *Message*-exchange function. Properties specified include encryption,
654 digital signature, and reliable-messaging characteristics. The options selected for the Document-
655 exchange layer are complementary to those selected for the transport layer. For example, if
656 *Message* security is desired and the selected transport protocol does not provide *Message*
657 encryption, then *Message* encryption MUST be specified in the Document-exchange layer. The

658 protocol for exchanging Messages between two Parties is defined by the ebXML Message
659 Service specification[ebMS] or other similar messaging services.

661 **Transport layer** - The transport layer identifies the transport protocol to be used in sending
662 messages through the network and defines the endpoint addresses, along with various other
663 properties of the transport protocol. Choices of properties in the transport layer are
664 complementary to those in the document-exchange layer (see "Document-Exchange Layer"
665 directly above.)

666
667 Note that the functional layers encompassed by the *CPP* ~~have no understanding~~ are independent
668 of the contents of the payload of the *Business* documents.

669

670 **7.18.1 Globally-Unique Identifier of CPP Instance Document**

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

674

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

678

679 **7.28.2 SchemaLocation Attribute**

680

681 Implementations of CPP and CPA authoring tools are STRONGLY RECOMMENDED to
682 include the XMLSchema-instance namespace-qualified schemaLocation attribute in the
683 document's root element to indicate to validating parsers the location URI of the schema
684 document that SHOULD be used to validate the document. Failure to include the
685 schemaLocation attribute MAY result in interoperability issues with other tools that need to be
686 able to validate these documents.

687

688 An example of the use of the schemaLocation attribute follows:

689

```
690 <tp:CollaborationProtocolAgreement
691   xmlns:tp="http://www.oasis-open.org/committees/ebxml-
692 cppa/schema/cpp-cpa-1_1.xsd"
693   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
694   xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-
695 cppa/schema/cpp-cpa-1_1.xsd http://www.oasis-open.org/committees/ebxml-
696 cppa/schema/cpp-cpa-1_1.xsd">
697   ...
698 </tp:CollaborationProtocolAgreement>
```

699

700 **7.38.3 CPP Structure**

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

```

703
704     <tp:CollaborationProtocolProfile
705         xmlns:tp="http://www.oasis-open.org/committees/ebxml-
706 cppa/schema/cpp-cpa-1_1.xsd"
707         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
708         xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-
709 cppa/schema/cpp-cpa-1_1.xsd http://www.oasis-open.org/committees/ebxml-
710 cppa/schema/cpp-cpa-1_1.xsd"
711         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
712         xmlns:xlink="http://www.w3.org/1999/xlink"
713         tp:version="1.1">
714     <tp:PartyInfo> <!-- one or more -->
715         ...
716     </tp:PartyInfo>
717     <tp:SimplePart> <!-- one or more -->
718         ...
719     </tp:SimplePart>
720     <tp:Packaging id="ID"> <!-- one or more -->
721         ...
722     </tp:Packaging>
723     <ds:Signature> <!--zero or one-->
724         ...
725     </ds:Signature>
726     <tp:Comment>text</tp:Comment> <!--zero or more-->
727 </tp:CollaborationProtocolProfile>
728

```

729 **7.48.4 CollaborationProtocolProfile element**

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

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

- 733 • The *CPP/CPA* namespace: xmlns:tp="http://www.oasis-
- 734 open.org/committees/ebxml-cppa/schema/cpp-cpa-1_1.xsd",
- 735 • XML Digital Signature namespace:
- 736 xmlns:ds="http://www.w3.org/2000/09/xmldsig#",
- 737 • and the XLink namespace:
- 738 xmlns:xlink="http://www.w3.org/1999/xlink".

739
740 In addition, the *CollaborationProtocolProfile* element contains an IMPLIED *version* attribute
741 that indicates the version of the *CPP*. Its purpose is to provide versioning capabilities for
742 instances of an enterprise's *CPP*. The value of the version attribute SHOULD be a string
743 representation of a numeric value such as "1.0" or "2.3". The value of the version string
744 SHOULD be changed with each change made to the *CPP* document after it has been published.

745
746 NOTE: The method of assigning the version-identifier value is left to the implementation.
747

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

- 749 • One or more REQUIRED *PartyInfo* elements that identify the organization (or parts
750 of the organization) whose capabilities are described by the *CPP*,

- 751 • One or more REQUIRED *SimplePart* elements that describe the constituents used to
752 make up composite *Messages*.
- 753 • One or more REQUIRED *Packaging* elements that describe how the *Message*
754 *Header* and payload constituents are packaged for transmittal,
- 755 • Zero or one *ds:Signature* element that contains the digital signature that signs the
756 *CPP* document,
- 757 • Zero or more *Comment* elements.

758

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

764 **7.58.5 PartyInfo Element**

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

```
770
771     <tp:PartyInfo
772         tp:partyName="..." tp:defaultMshChannelId="...">
773         <tp:PartyId tp:type="..."> <!-- one or more -->
774             ...
775         </tp:PartyId>
776         <tp:PartyRef xlink:type="..." xlink:href="..." />
777         <tp:CollaborationRole> <!-- one or more -->
778             ...
779         </tp:CollaborationRole>
780         <tp:Certificate> <!-- one or more -->
781             ...
782         </tp:Certificate>
783         <tp:DeliveryChannel> <!-- one or more -->
784             ...
785         </tp:DeliveryChannel>
786         <tp:Transport> <!-- one or more -->
787             ...
788         </tp:Transport>
789         <tp:DocExchange> <!-- one or more -->
790             ...
791         </tp:DocExchange>
792         ...
793     </tp:OverrideMshActionBinding> <!-- zero or more -->
794         ...
795     </tp:OverrideMshActionBinding>
796 </tp:PartyInfo>
```

797

798 The *PartyInfo* element contains a REQUIRED *partyName* attribute that indicates the common,
799 human readable name of the organization. Unlike *PartyID*, *partyName* might not be unique;

800 however, the value of each *partyName* SHALL be meaningful enough to directly identify the
801 organization or the subdivision of an organization described in the *PartyInfo* element.
802

803 The following example illustrates two possible party names.

```
804  
805 <tp:PartyInfo tp:partyName="Example, Inc."...</tp:PartyInfo>  
806  
807 <tp:PartyInfo tp:partyName="Example, Inc. US Western Division">  
808 ...  
809 </tp:PartyInfo>
```

810
811 The *PartyInfo* element also contains a REQUIRED *defaultMshChannelId* attribute. It identifies
812 the default *DeliveryChannel* to be used for sending standalone *Message Service Handler* [\[ebMSI\]](#)
813 level messages (i.e., Acknowledgment, Error, StatusRequest, StatusResponse, Ping, Pong) that
814 are to be delivered asynchronously. When synchronous reply mode is in use, *Message Service*
815 *Handler* level messages are returned synchronously. The default can be overridden through the
816 use of *OverrideMshActionBinding* elements.

817
818 The *PartyInfo* element consists of the following child elements:

- 819 • One or more REQUIRED *PartyId* elements that provide a logical identifier for the
820 organization.
- 821 • A REQUIRED *PartyRef* element that provides a pointer to more information about
822 the *Party*.
- 823 • One or more REQUIRED *CollaborationRole* elements that identify the roles that this
824 *Party* can play in the context of a *Process Specification*.
- 825 • One or more REQUIRED *Certificate* elements that identify the certificates used by
826 this *Party* in security functions.
- 827 • One or more REQUIRED *DeliveryChannel* elements that define the characteristics of
828 each delivery channel that the *Party* can use to receive *Messages*. It includes both the
829 transport [level-protocol](#) (e.g. HTTP) and the messaging protocol (e.g. ebXML
830 *Message Service*).
- 831 • One or more REQUIRED *Transport* elements that define the characteristics of the
832 transport protocol(s) that the *Party* can support to [send and](#) receive *Messages*.
- 833 • One or more REQUIRED *DocExchange* elements that define the *Message*-exchange
834 characteristics, such as the *Message*-exchange protocol, that the *Party* can support.
- 835 • Zero or more *OverrideMshActionBinding* elements that specify the *DeliveryChannel*
836 to use for asynchronously delivered *Message Service Handler* level messages.
837

838 [7.5.18.5.1](#) *PartyId* element

839 The REQUIRED *PartyId* element provides a logical identifier that MAY be used to logically
840 identify the *Party*. Additional *PartyId* elements MAY be present under the same *PartyInfo*
841 element so as to provide for alternative logical identifiers for the *Party*. If the *Party* has
842 preferences as to which logical identifier is used, the *PartyId* elements SHOULD be listed in
843 order of preference starting with the most-preferred identifier.
844

845 In a *CPP* that contains multiple *PartyInfo* elements, different *PartyInfo* elements MAY contain
846 *PartyId* elements that define different logical identifiers. This permits a large organization, for
847 example, to have different identifiers for different purposes.

848
849 The value of the *PartyId* element is any string that provides a unique identifier. The identifier
850 MAY be any identifier that is understood by both *Parties* to a *CPA*. Typically, the identifier
851 would be listed in a well-known directory such as DUNS ([Dun and Bradstreet](#)) or in any naming
852 system specified by [ISO6523].

853
854 The *PartyId* element has a single IMPLIED attribute: *type* that has a string value.

855
856 If the *type* attribute is present, then it provides a scope or namespace for the content of the
857 *PartyId* element.

858
859 If the *type* attribute is not present, the content of the *PartyId* element MUST be a URI that
860 conforms to [RFC2396]. It is RECOMMENDED that the value of the *type* attribute be a URN
861 that defines a namespace for the value of the *PartyId* element. Typically, the URN would be
862 registered [as-in](#) a well-known directory of organization identifiers.

863
864 The following example illustrates two URI references.

```
865 <tp:PartyId tp:type="anyURI">urn:duns:123456789</tp:PartyId>  
866  
867 <tp:PartyId tp:type="anyURI"> urn:icann:example.com</tp:PartyId>
```

868
869
870 The first example is the URN for the *Party's* DUNS number, assuming that Dun and Bradstreet
871 has registered a URN for DUNS numbers with the Internet Assigned Numbers Authority
872 (IANA). The last field is the DUNS number of the organization.

873
874 The second example shows an arbitrary URN. This might be a URN that the *Party* has
875 registered with IANA to identify itself directly.

876 877 **[7.5.28.5.2](#) PartyRef element**

878 The *PartyRef* element provides a link, in the form of a URI, to additional information about the
879 *Party*. Typically, this would be the URL from which the information can be obtained. The
880 information might be at the *Party's* web site or in a publicly accessible repository such as an
881 ebXML Registry, a UDDI repository ([www.uddi.org](#)), or [an-a Lightweight Directory Access](#)
882 [Protocol\[RFC2251\]](#) (LDAP) directory. Information available at that URI MAY include contact
883 names, addresses, and phone numbers, and perhaps more information about the *Business*
884 *Collaborations* that the *Party* supports. This information MAY be in the form of an ebXML Core
885 Component[ccOVER]. It is not within the scope of this specification to define the content or
886 format of the information at that URI.

887
888 The *PartyRef* element is an [XLINK] simple link. It has the following attributes:

- 889 • a REQUIRED *xlink:type* attribute,

- 890
- a REQUIRED *xlink:href* attribute,
 - an IMPLIED *type* attribute.
- 891
- 892

893 The contents of the document referenced by the *partyRef* element are subject to change at any
 894 time. Therefore, it SHOULD NOT be cached for a long period of time. Rather, the value of the
 895 *xlink:href* SHOULD be dereferenced only when the contents of this document are needed.

896

897 **7.5.2.18.5.2.1 xlink:type attribute**

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

900

901 **7.5.2.28.5.2.2 xlink:href attribute**

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

904

905 **7.5.2.38.5.2.3 type attribute**

906 The value of the IMPLIED *type* attribute identifies the document type of the external information
 907 about the *Party*. It MUST be a URI that defines the namespace associated with the information
 908 about the *Party*. If the *type* attribute is omitted, the external information about the *Party* MUST
 909 be an HTML web page.

910

911 An example of the *PartyRef* element is:

912

```
913 <tp:PartyRef xlink:type="simple"
914             xlink:href="http://example2.com/ourInfo.xml"
915             tp:type="anyURI" />
```

916

917 **7.5.38.5.3 CollaborationRole element**

918 The *CollaborationRole* element associates a *Party* with a specific role in the *Business*
 919 *Collaboration* ~~that is defined in the *Process Specification* document [ebBPSS]~~. Generally, the
 920 *Process Specification* is defined in terms of roles such as "buyer" and "seller". The association
 921 between a specific *Party* and the role(s) it is capable of fulfilling within the context of a *Process-*
 922 *Specification* is defined in both the *CPP* and *CPA* documents. In a *CPP*, the *CollaborationRole*
 923 element identifies which role the *Party* is capable of playing in each *Process Specification*
 924 documents referenced by the *CPP*. An example of the *CollaborationRole* element, based on
 925 RosettaNet™ PIP 3A4 is: ~~element is:~~

926

```
927 <tp:CollaborationRole tp:id="BuyerId">
928   <tp:ProcessSpecification
929     tp:version="2.0"
930     tp:name="PIP3A4RequestPurchaseOrder"
931     xlink:type="simple"
932     xlink:href="http://www.rosettanet.org/processes/3A4.xml" />
933   <tp:Role
934     tp:name="Buyer"
935     xlink:type="simple"
936     xlink:href="http://www.rosettanet.org/processes/3A4.xml#Buyer" />
```

937

```

938     <tp:ApplicationCertificateRef
939         tp:certId="company1SigningCertificate"/>
940     <!-- This service binding uses an asynchronous delivery channel to
941         receive the signals and response -->
942     <tp:ServiceBinding>
943         <tp:Service
944 tp:type="anyURI">bpid:RosettaNet:PIP3A4RequestPurchaseOrder$2.0</tp:Service>
945         <tp:ActionBinding
946             tp:action="Purchase Order Confirmation Action"
947             tp:channelId="channel1"
948             tp:packageId="ResponsePackage">
949             <tp:ActionContext
950                 tp:binaryCollaboration="Request Purchase Order"
951                 tp:businessTransactionActivity="Request Purchase Order"
952                 tp:requestOrResponseAction="Purchase Order Confirmation
953 Action"/>
954             </tp:ActionBinding>
955             <!-- Receipt Acknowledgment and Exception signals are delivered
956                 using the designated delivery channels -->
957             <tp:ActionBinding
958                 tp:action="ReceiptAcknowledgment"
959                 tp:channelId="channel1"
960                 tp:packageId="ReceiptAcknowledgmentPackage"/>
961             <tp:ActionBinding
962                 tp:action="Exception"
963                 tp:channelId="channel1"
964                 tp:packageId="ExceptionPackage"/>
965             </tp:ServiceBinding>
966     </tp:CollaborationRole>
967

```

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

972
973 The **CollaborationRole** element SHALL consist of the following child elements: a REQUIRED
974 **ProcessSpecification** element, a REQUIRED **Role** element, zero or one
975 **ApplicationCertificateRef** element, zero or one **ApplicationSecurityDetailsRef** element, and one
976 or more **ServiceBinding** elements. The **ProcessSpecification** element identifies the *Process-*
977 *Specification* document that defines such role. The **Role** element identifies which role the *Party*
978 is capable of supporting. The **ApplicationCertificateRef** element identifies the certificate to be
979 used for application level signature and encryption. The **ApplicationSecurityDetailsRef** element
980 identifies the trust anchors and security policy that will be applied to any application-level
981 certificate offered by the other *Party*. Each **ServiceBinding** element provides a binding of the
982 role to a default **DeliveryChannel** (through the *defaultSignalChannelId* attribute) for sending
983 business signal messages like *Receipt Acknowledgment* and *Exception*. The **ActionBinding**
984 elements identify the **DeliveryChannel** elements that are relevant for delivering business action
985 messages received by the **Role** in question. They MAY also be used for specifying
986 **DeliveryChannels** for business signal messages.

987
988 When there are more than one **ServiceBinding** child elements of a **CollaborationRole**, then the

989 order of the **ServiceBinding** elements SHALL be treated as signifying the *Party's* preference
990 starting with highest and working towards lowest.

991

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

995

996 When a *CPA* is composed, only **ServiceBinding** elements that are compatible between the two
997 *Parties* SHALL be retained. Each *Party* SHALL have a default delivery channel for the delivery
998 of standalone *Message* Service Handler level signals like (Reliable Messaging)
999 Acknowledgments, Errors, StatusRequest, StatusResponse, etc.

1000

1001 NOTE: An implementation MAY provide the capability of dynamically assigning
1002 delivery channels on a per *Message* basis during performance of the *Business*
1003 *Collaboration*. The delivery channel selected would be chosen, based on present
1004 conditions, from those identified by **ServiceBinding** elements that refer to the *Business*
1005 *Collaboration* that is sending the *Message*. If more than one delivery channel is
1006 applicable, the one referred to by the highest-preference **ServiceBinding** element is used.

1007

1008 The **CollaborationRole** element has the following attribute:

- 1009 • a REQUIRED *id* attribute.

1010

1011 **7.5.3.18.5.3.1 id attribute**

1012 The REQUIRED *id* attribute is an [XML] ID attribute by which this **CollaborationRole** element
1013 can be referenced from elsewhere in the *CPP* document.

1014

1015

1016 **7.5.48.5.4 ProcessSpecification element**

1017 The **ProcessSpecification** element provides the link to the *Process-Specification* document that
1018 defines the interactions between the two *Parties*. It is RECOMMENDED that this *Business-*
1019 *Collaboration* description be prepared in accordance with the ebXML Business Process
1020 Specification Schema[ebBPSS]. The *Process-Specification* document MAY be kept in an
1021 ebXML Registry.

1022

1023 NOTE: A *Party* **MAY-can** describe the *Business Collaboration* using any desired
1024 alternative to the ebXML Business Process Specification Schema. When an alternative
1025 *Business-Collaboration* description is used, the *Parties* to a *CPA* MUST agree on how to
1026 interpret the *Business-Collaboration* description and how to interpret the elements in the
1027 *CPA* that reference information in the *Business-Collaboration* description. The affected
1028 elements in the *CPA* are the **Role** element, the **ActionBinding** element, **the**
1029 **ActionContext element**, and some attributes of the **BusinessProcessCharacteristics**
1030 element.

1031

1032 The syntax of the **ProcessSpecification** element is:

1033

```

1034     <tp:ProcessSpecification
1035         tp:version="2.0"
1036         tp:name="PIP3A4RequestPurchaseOrder"
1037         xlink:type="simple"
1038         xlink:href="http://www.rosettanet.org/processes/3A4.xml"
1039         <ds:Reference ds:URI="http://www.rosettanet.org/processes/3A4.xml">
1040             <ds:Transforms>
1041                 <ds:Transform
1042 ds:Algorithm="http://www.w3.org/TR/20002001/REC-xml-c14n-20010315"/>
1043             </ds:Transforms>
1044             <ds:DigestMethod
1045                 ds:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
1046             <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
1047         </ds:Reference>
1048     </tp:ProcessSpecification>

```

1049
 1050 The *ProcessSpecification* element has a single REQUIRED child element, *ds:Reference*, and the
 1051 following attributes:

- 1052 • a REQUIRED *name* attribute,
- 1053 • a REQUIRED *version* attribute,
- 1054 • a FIXED *xlink:type* attribute,
- 1055 • a REQUIRED *xlink:href* attribute.

1056
 1057 The *ds:Reference* element relates to the *xlink:type* and *xlink:href* attributes as follows. Each
 1058 *ProcessSpecification* element SHALL contain one *xlink:href* attribute and one *xlink:type*
 1059 attribute with a value of "simple", and MAY contain one *ds:Reference* element formulated
 1060 according to the XML Digital Signature specification[XMLDSIG]. In case the [CPP \(CPA\)](#)
 1061 document is signed, it each *ProcessSpecification* element MUST use-contain the *ds:Reference*
 1062 element. When the *ds:Reference* element is present, it MUST include a *ds:URI* attribute whose
 1063 value is identical to that of the *xlink:href* attribute in the enclosing *ProcessSpecification*
 1064 element. [The *ds:Reference* element specifies a digest method and digest value to enable](#)
 1065 [verification that the referenced *Process-Specification* document has not changed.](#)

1066

1067

1068 [7.5.4.18.5.4.1](#) name attribute

1069 The *ProcessSpecification* element MUST include a REQUIRED *name* attribute: a string that
 1070 identifies the *Business Process-Specification* being performed.

1071

1072 [7.5.4.28.5.4.2](#) version attribute

1073 The *ProcessSpecification* element includes a REQUIRED *version* attribute to identify the
 1074 version of the *Process-Specification* document identified by the *xlink:href* attribute (and also
 1075 identified by the *ds:Reference* element, if any).

1076

1077 [7.5.4.38.5.4.3](#) xlink:type attribute

1078 The *xlink:type* attribute has a FIXED value of "simple". This identifies the element as being an
 1079 [XLINK] simple link.

1080

1081 **7.5.4.48.5.4.4 xlink:href attribute**

1082 The REQUIRED *xlink:href* attribute SHALL have a value that identifies the *Process-*
1083 *Specification* document and is a URI that conforms to [RFC2396].

1085 **7.5.4.58.5.4.5 ds:Reference element**

1086 The *ds:Reference* element identifies the same *Process-Specification* document as the enclosing
1087 *ProcessSpecification* element's *xlink:href* attribute and additionally provides for verification that
1088 the *Process-Specification* document has not changed since the *CPP* was created, through the use
1089 of a digest method and digest value as described below.

1091 NOTE: *Parties* MAY test the validity of the *CPP* or *CPA* at any time. The following
1092 validity tests MAY be of particular interest:

- 1094 • test of the validity of a *CPP* and the referenced *Process-Specification* documents at
1095 the time composition of a *CPA* begins in case they have changed since they were
1096 created,
- 1097 • test of the validity of a *CPA* and the referenced *Process-Specification* documents at
1098 the time a *CPA* is installed into a *Party's* system,
- 1099 • test of the validity of a *CPA* at intervals after the *CPA* has been installed into a *Party's*
1100 system. The *CPA* and the referenced *Process-Specification* documents MAY be
1101 processed by an installation tool into a form suited to the particular middleware.
1102 Therefore, alterations to the *CPA* and the referenced *Process-Specification* documents
1103 do not necessarily affect ongoing run-time operations. Such alterations might not be
1104 detected until it becomes necessary to reinstall the *CPA* and the referenced *Process-*
1105 *Specification* documents.

- 1107 1. The syntax and semantics of the *ds:Reference* element and its child elements are defined
1108 in the XML Digital Signature specification[XMLDSIG]. In addition, To identify the
1109 *Process-Specification* document, the *ds:Reference* MUST include a *ds:URI* attribute
1110 whose value is identical to that of the *xlink:href* attribute in the enclosing
1111 *ProcessSpecification* element.
- 1112 2. According to [XMLDSIG], a *ds:Reference* element can have a *ds:Transforms* child
1113 element, which in turn has an ordered list of one or more *ds:Transform* child elements to
1114 specify a sequence of transforms. However, this specification currently REQUIRES the
1115 Canonical XML[XMLC14N] transform and forbids other transforms. Therefore, the
1116 following additional requirements apply to a *ds:Reference* element within a
1117 *ProcessSpecification* element:

- 1118 • The *ds:Reference* element MUST have a *ds:Transforms* child element.
- 1119 • That *ds:Transforms* element MUST have exactly one *ds:Transform* child
1120 element.
- 1121 • That *ds:Transform* element MUST specify the Canonical XML[XMLC14N]
1122 transform via the following REQUIRED value for its REQUIRED *ds:Algorithm*
1123 attribute: <http://www.w3.org/TR/2001/Rec-xml-c14n-20010315>
1124

1125

1126 Note that implementation of Canonical XML is REQUIRED by the XML Digital
1127 Signature specification[XMLDSIG].
1128

1129 To enable verification that the identified and transformed *Process-Specification* document has
1130 not changed, the *ds:DigestMethod* element specifies the digest algorithm applied to the *Process-*
1131 *Specification* document, and the *ds:DigestValue* element specifies the resulting-expected value.
1132 The *Process-Specification* document is presumed to be unchanged if and only if the result of
1133 applying the digest algorithm to the *Process-Specification* document results in the expected
1134 value.
1135

1136 A *ds:Reference* element in a *ProcessSpecification* element has implications for *CPP* validity:
1137

- 1138 • A *CPP* MUST be considered invalid if any *ds:Reference* element within a
1139 *ProcessSpecification* element fails reference validation as defined by the XML Digital
1140 Signature specification[XMLDSIG].
1141
- 1142 • A *CPP* MUST be considered invalid if any *ds:Reference* element within it cannot be
1143 dereferenced.
1144

1145 Other validity implications of such *ds:Reference* elements are specified in the description of the
1146 *ds:Signature* element.
1147

1148 NOTE: The XML Digital Signature specification[XMLDSIG] states "The signature
1149 application MAY rely upon the identification (URI) and Transforms provided by the
1150 signer in the Reference element, or it MAY obtain the content through other means such
1151 as a local cache" (emphases on MAY added). However, it is RECOMMENDED that
1152 ebXML *CPP/CPA* implementations not make use such cached results when signing or
1153 validating.
1154

1155 NOTE: It is recognized that the XML Digital Signature specification[XMLDSIG]
1156 provides for signing an XML document together with externally referenced documents.
1157 In cases where a *CPP* or *CPA* document is in fact suitably signed, that facility could also
1158 be used to ensure that the referenced *Process-Specification* documents are unchanged.
1159 However, this specification does not currently mandate that a *CPP* or *CPA* be signed.
1160

1161 NOTE: If the *Parties* to a *CPA* wish to customize a previously existing *Process-*
1162 *Specification* document, they MAY copy the existing document, modify it, and cause
1163 their *CPA* to reference the modified copy. It is recognized that for reasons of clarity,
1164 brevity, or historical record, the parties might prefer to reference a previously existing
1165 *Process-Specification* document in its original form and accompany that reference with a
1166 specification of the agreed modifications. Therefore, *CPP* usage of the *ds:Reference*
1167 element's *ds:Transforms* subelement within a *ProcessSpecification* element might be
1168 expanded in the future to allow other transforms as specified in the XML Digital
1169 Signature specification[XMLDSIG]. For example, modifications to the original
1170 document could then be expressed as XSLT transforms. After applying any transforms,

1171 it would be necessary to validate the transformed document against the ebXML Business
1172 Process Specification Schema[ebBPSS].
1173

1174 **7.5.58.5.5 Role element**

1175 The REQUIRED *Role* element identifies which role in the *Process Specification* the *Party* is
1176 capable of supporting via the *ServiceBinding* element(s) siblings within this *CollaborationRole*
1177 element.
1178

1179 The *Role* element has the following attributes:

- 1180 • a REQUIRED *name* attribute,
 - 1181 • a FIXED *xlink:type* attribute,
 - 1182 • a REQUIRED *xlink:href* attribute.
- 1183

1184 **7.5.5.18.5.5.1 name attribute**

1185 The REQUIRED *name* attribute is a string that gives a name to the *Role*. Its value is taken from
1186 one of the following sources in the *Process Specification*[ebBPSS] that is referenced by the
1187 *ProcessSpecification* element depending upon which element is the "root" (highest order) of the
1188 process referenced:

- 1189 • *name* attribute of a *BinaryCollaboration/initiatingRole* element,
 - 1190 • *name* attribute of a *BinaryCollaboration/respondingRole* element,
 - 1191 • *fromAuthorizedRole* attribute of a *BusinessTransactionActivity* element,
 - 1192 • *toAuthorizedRole* attribute of a *BusinessTransactionActivity* element,
 - 1193 • *fromAuthorizedRole* attribute of a *CollaborationActivity* element,
 - 1194 • *toAuthorizedRole* attribute of a *CollaborationActivity* element,
 - 1195 • *name* attribute of the *BusinessPartnerRole* element.
- 1196

1197 See NOTE in Section [8.5.47.5.4](#) regarding alternative *Business-Collaboration* descriptions.
1198

1199 **7.5.5.28.5.5.2 xlink:type attribute**

1200 The *xlink:type* attribute has a FIXED value of "simple". This identifies the element as being an
1201 [XLINK] simple link.
1202

1203 **7.5.5.38.5.5.3 xlink:href attribute**

1204 The REQUIRED *xlink:href* attribute SHALL have a value that is a URI that conforms to
1205 [RFC2396]. It identifies the location of the element or attribute within the *Process-Specification*
1206 document that defines the role in the context of the *Business Collaboration*. An example is:
1207

```
1208     xlink:href="http://www.rosettanet.org/processes/3A4.xml#Buyer"
```

1209

1210 Where "Buyer" is the value of the ID attribute of the element in the *Process-Specification*
1211 document that defines the role name.
1212

1213 **7.5.68.5.6 ApplicationCertificateRef element**

1214 The OPTIONAL *ApplicationCertificateRef* element identifies a signing certificate for use by the
Collaboration-Protocol Profile and Agreement Specification

1215 business process/application layer. This certificate is not used by the ebXML messaging system,
1216 but it is included in the *CPP* so that it can be considered in the *CPA* negotiation process.

1217

1218 The *ApplicationCertificateRef* element has

- 1219 • A REQUIRED *certId* attribute

1220

1221 [7.5.6.18.5.6.1](#) *certId* attribute

1222 The REQUIRED *certId* attribute is an [XML] IDREF that associates the *CollaborationRole* with
1223 a certificate. It MUST have a value equal the value of the *certId* attribute of one of the
1224 *Certificate* elements under *PartyInfo*.

1225

1226 [7.5.78.5.7](#) *ApplicationSecurityDetailsRef* element

1227 The OPTIONAL *ApplicationSecurityDetailsRef* element identifies the trust anchors and security
1228 policy that this *Party* will apply to any application-level certificate offered by the other *Party*.
1229 These trust anchors and policy are not used by the ebXML messaging system, but are included in
1230 the *CPP* so that they can be considered in the *CPA* negotiation process.

1231

1232 The *ApplicationSecurityDetailsRef* element has

- 1233 • A REQUIRED *securityId* attribute

1234

1235 [7.5.7.18.5.7.1](#) *SecurityId* attribute

1236 The REQUIRED *securityId* attribute is an [XML] IDREF that associates the *CollaborationRole*
1237 with a *SecurityDetails* element that specifies a set of trust anchors and a security policy. It
1238 MUST have a value equal to the value of the *securityId* attribute of one of the *SecurityDetails*
1239 elements under *PartyInfo*.

1240

1241 [7.5.88.5.8](#) *ServiceBinding* element

1242 The *ServiceBinding* element identifies a default *DeliveryChannel* element for all of the
1243 business signal traffic that is to be sent or received by the *Party* within the context of the
1244 identified *Process-Specification* document. An example of the *ServiceBinding* element is:

1245

```
1246 <tp:ServiceBinding tp:defaultSignalChannelId="channelA1">
```

```
1247   <tp:Service tp:type="anyURI">
```

```
1248     bpid:RosettaNet:PIP3A4RequestPurchaseOrder$2.0
```

```
1249   </tp:Service>
```

```
1250   <tp:WillInitiate>
```

```
1251     <tp:ThisPartyActionBinding tp:action="Purchase Order Request Action"
```

```
1252       tp:channelId="channelA1"
```

```
1253       tp:packageId="RequestPackage">
```

```
1254         <tp>ActionContext tp:binaryCollaboration="Request Purchase Order"
```

```
1255           tp:businessTransactionActivity="Request Purchase Order"
```

```
1256           tp:requestOrResponseAction="Purchase Order Request Action"/>
```

```
1257       </tp:ThisPartyActionBinding>
```

```
1258     <tp:OtherPartyActionBinding tp:action="Purchase Order Request Action"
```

```
1259       tp:channelId="channelB1"
```

```
1260       tp:packageId="RequestPackage">
```

```
1261         <tp>ActionContext tp:binaryCollaboration="Request Purchase Order"
```

```
1262           tp:businessTransactionActivity="Request Purchase Order"
```

```
1263           tp:requestOrResponseAction="Purchase Order Request Action"/>
```

```
1264     </tp:OtherPartyActionBinding>
```

```
1265   </tp:WillInitiate>
```


12/14/200102/03/2002

```

1266 <tp:WillInitiate>
1267 <tp:ThisPartyActionBinding tp:action="ReceiptAcknowledgement"
1268 <tp:channelId="channelA1"
1269 tp:packageId="ReceiptAcknowledgmentPackage"/>
1270 <tp:OtherPartyActionBinding tp:action="ReceiptAcknowledgement"
1271 tp:channelId="channelB1"
1272 tp:packageId="ReceiptAcknowledgmentPackage"/>
1273 </tp:WillInitiate>
1274 <tp:WillRespond>
1275 <tp:ThisPartyActionBinding tp:action="Purchase Order Confirmation Action"
1276 tp:channelId="channelA1"
1277 tp:packageId="ResponsePackage">
1278 <tp:ActionContext tp:binaryCollaboration="Request Purchase Order"
1279 tp:businessTransactionActivity="Request Purchase Order"
1280 tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
1281 </tp:ThisPartyActionBinding>
1282 <tp:OtherPartyActionBinding tp:action="Purchase Order Confirmation Action"
1283 tp:channelId="channelB1"
1284 tp:packageId="ResponsePackage">
1285 <tp:ActionContext tp:binaryCollaboration="Request Purchase Order"
1286 tp:businessTransactionActivity="Request Purchase Order"
1287 tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
1288 </tp:OtherPartyActionBinding>
1289 </tp:WillRespond>
1290 <!-- Receipt Acknowledgment and Exception signals are delivered
1291 using the designated delivery channels -->
1292 <tp:WillRespond>
1293 <tp:ThisPartyActionBinding tp:action="ReceiptAcknowledgment"
1294 tp:channelId="channelA1"
1295 tp:packageId="ReceiptAcknowledgmentPackage"/>
1296 <tp:OtherPartyActionBinding tp:action="ReceiptAcknowledgment"
1297 tp:channelId="channelB1"
1298 tp:packageId="ReceiptAcknowledgmentPackage"/>
1299 </tp:WillRespond>
1300 <tp:WillRespond>
1301 <tp:ThisPartyActionBinding tp:action="Exception"
1302 tp:channelId="channelA1"
1303 tp:packageId="ExceptionPackage"/>
1304 <tp:OtherPartyActionBinding tp:action="Exception"
1305 tp:channelId="channelB1"
1306 tp:packageId="ExceptionPackage"/>
1307 </tp:WillRespond>
1308 </tp:ServiceBinding>
1309 <tp:ServiceBinding>
1310 <tp:Service
1311 tp:type="anyURI">bpid:RosettaNet:PIP3A4RequestPurchaseOrder$2.0</tp:Service>
1312 <tp:WillInitiate>
1313 <tp:ThisPartyActionBinding tp:action="Purchase Order
1314 Request Action" tp:channelId="channelA1" tp:packageId="RequestPackage">
1315 <tp:ActionContext tp:binaryCollaboration="Request
1316 Purchase Order" tp:businessTransactionActivity="Request Purchase Order"
1317 tp:requestOrResponseAction="Purchase Order Request Action"/>
1318 </tp:ThisPartyActionBinding>
1319 <tp:OtherPartyActionBinding tp:action="Purchase Order
1320 Request Action" tp:channelId="channelB1" tp:packageId="RequestPackage">
1321 <tp:ActionContext tp:binaryCollaboration="Request
1322 Purchase Order" tp:businessTransactionActivity="Request Purchase Order"
1323 tp:requestOrResponseAction="Purchase Order Request Action"/>
1324 </tp:OtherPartyActionBinding>
1325 </tp:WillInitiate>
1326 <tp:WillInitiate>
1327 <tp:ThisPartyActionBinding
1328 tp:action="ReceiptAcknowledgement" tp:channelId="channelA1"
1329 tp:packageId="ReceiptAcknowledgmentPackage"/>

```

12/14/200102/03/2002

```

1330         -----<tp:OtherPartyActionBinding
1331 tp:action="ReceiptAcknowledgement" tp:channelId="channelB1"
1332 tp:packageId="ReceiptAcknowledgmentPackage"/>
1333         -----</tp:WillInitiate>
1334         -----<tp:WillRespond>
1335         -----<tp:ThisPartyActionBinding tp:action="Purchase Order
1336 Confirmation Action" tp:channelId="channelA1" tp:packageId="ResponsePackage">
1337         -----<tp:ActionContext tp:binaryCollaboration="Request
1338 Purchase Order" tp:businessTransactionActivity="Request Purchase Order"
1339 tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
1340         -----</tp:ThisPartyActionBinding>
1341         -----<tp:OtherPartyActionBinding tp:action="Purchase Order
1342 Confirmation Action" tp:channelId="channelB1" tp:packageId="ResponsePackage">
1343         -----<tp:ActionContext tp:binaryCollaboration="Request
1344 Purchase Order" tp:businessTransactionActivity="Request Purchase Order"
1345 tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
1346         -----</tp:OtherPartyActionBinding>
1347         -----</tp:WillRespond>
1348         -----<!-- Receipt Acknowledgment and Exception signals are delivered
1349 using the designated delivery channels -->
1350         -----<tp:WillRespond>
1351         -----<tp:ThisPartyActionBinding
1352 tp:action="ReceiptAcknowledgment" tp:channelId="channelA1"
1353 tp:packageId="ReceiptAcknowledgmentPackage"/>
1354         -----<tp:OtherPartyActionBinding
1355 tp:action="ReceiptAcknowledgment" tp:channelId="channelB1"
1356 tp:packageId="ReceiptAcknowledgmentPackage"/>
1357         -----</tp:WillRespond>
1358         -----<tp:WillRespond>
1359         -----<tp:ThisPartyActionBinding tp:action="Exception"
1360 tp:channelId="channelA1" tp:packageId="ExceptionPackage"/>
1361         -----<tp:OtherPartyActionBinding tp:action="Exception"
1362 tp:channelId="channelB1" tp:packageId="ExceptionPackage"/>
1363         -----</tp:WillRespond>
1364         -----</tp:ServiceBinding>

```

1365
1366 The *ServiceBinding* element has one child *Service* element, zero or more *WillInitiate* child
1367 elements, and zero or more *WillRespond* child elements.

1368
1369 The *ServiceBinding* element also has:

- A REQUIRED *defaultSignalChannelId* attribute.

1370 1371 7.5.8.18.5.8.1 defaultSignalChannelId attribute

1372 The REQUIRED *defaultSignalChannelId* attribute is an [XML] IDREF that identifies the
1373 *DeliveryChannel* that SHALL provide a default technical binding for the business signal
1374 *message* traffic that is received for the *Process Specification* that is referenced by the
1375 *ProcessSpecification* element.

1376
1377

1378 7.5.98.5.9 Service element

1379 The value of the *Service* element is a string that SHALL be used as the value of the *Service*
1380 element in the ebXML *Message Header*[ebMS] or a similar element in the *Message Header* of

1381 an alternative *message* service. The *Service* element has an IMPLIED *type* attribute.

1382
1383 If the *Process-Specification* document is defined by the ebXML Business Process Specification
1384 Schema[ebBPSS], then the value of the *Service* element is the *uuid* (URI) specified for the
1385 *ProcessSpecificationElement* in the Business Process Specification Schema instance document.

1386
1387 NOTE: The purpose of the *Service* element is to provide routing information for the
1388 ebXML *Message Header*. The *CollaborationRole* element and its child elements identify
1389 the information in the *ProcessSpecification* document that is relevant to the *CPP* or *CPA*.
1390 The *Service* element MAY be used along with the *WillInitiate* and *WillRespond*
1391 elements (and their descendants) to provide routing of received messages to the correct
1392 application entry point.~~to provide application routing of received messages.~~

1393 7.5.9.18.5.9.1 type attribute

1395 If the *type* attribute is present, it indicates that the *Parties* sending and receiving the *Message*
1396 know, by some other means, how to interpret the value of the *Service* element. The two *Parties*
1397 MAY use the value of the *type* attribute to assist the interpretation.

1398
1399 If the *type* attribute is not present, the value of the *Service* element MUST be a URI[RFC2396].
1400 ~~If using the ebXML Business Process Specification[ebBPSS] for defining~~ If using the ebXML
1401 Business Process Specification[ebBPSS] for defining the *Process-Specification* document, the
1402 type attribute MUST be a URI[RFC2396].

1403 7.5.108.5.10 WillInitiate element

1405 The *WillInitiate* element identifies an *action* invocation message that a *Party* will initiate or
1406 send. It has two sub-elements: *ThisPartyActionBinding* and *OtherPartyActionBinding*. The
1407 *ThisPartyActionBinding* element is REQUIRED for both *CPPs* and *CPAs*. It identifies the
1408 *DeliveryChannel* and the *Packaging* the Party described by the encompassing PartyInfo
1409 element ~~this Party~~ will use for sending the *action* invocation message in question. The
1410 *OtherPartyActionBinding* element is only used in the case of *CPAs*. It identifies the
1411 *DeliveryChannel* the other *Party* will use for receiving the *action* invocation message in
1412 question and the expected *Packaging*. Within a *CPA* and under the same *WillInitiate* element,
1413 the *DeliveryChannels* and *Packaging* used/expected by the two *Parties* MUST be compatible.

1414 7.5.118.5.11 WillRespond element

1416 The *WillRespond* element identifies an *action* invocation message that a *Party* will respond to or
1417 receive. It has two sub-elements: *ThisPartyActionBinding* and *OtherPartyActionBinding*. The
1418 *ThisPartyActionBinding* element is REQUIRED for both *CPPs* and *CPAs*. It identifies the
1419 *DeliveryChannel* ~~this the~~ Party described by the encompassing PartyInfo element will use for
1420 receiving the *action* invocation message in question and the *Packaging* it is expecting. The
1421 *OtherPartyActionBinding* element is only used in the case of *CPAs*. It identifies the
1422 *DeliveryChannel* and *Packaging* the other *Party* will use for sending the *action* invocation
1423 message in question. Within a *CPA* and under the same *WillRespond* element, the
1424 *DeliveryChannels* and *Packaging* used/expected by the two parties MUST be compatible.

1425

7.5.128.5.12 ThisPartyActionBinding element and OtherPartyActionBinding element

1427 The *ThisPartyActionBinding* and *OtherPartyActionBinding* elements have identical structure.
1428 Each specifies a *DeliveryChannel* for *Messages* for a selected *action* and the *Packaging* for
1429 those *Messages* that are to be sent or received by the *Party* in the context of the *Process*
1430 *Specification* that is associated with the parent *ServiceBinding* element.

1431

1432

1433 The *ThisPartyActionBinding* and *OtherPartyActionBinding* elements each have an OPTIONAL
1434 child *ActionContext* - element.

1435

1436 The *ThisPartyActionBinding* and *OtherPartyActionBinding* elements each have the following
1437 attributes:

- 1438 • a REQUIRED *action* attribute,
- 1439 • a REQUIRED *channelId* attribute,
- 1440 • a REQUIRED *packageId* attribute,
- 1441 • an IMPLIED *xlink:href* attribute,
- 1442 • a FIXED *xlink:type* attribute.

1443

1444 Under a given *ServiceBinding* element and among all subordinate *WillInitiate* elements, there
1445 SHALL be only one *ThisPartyActionBinding* element whose *action* attribute has a given value.
1446 Similarly, under a given *ServiceBinding* and among all subordinate *WillRespond* elements, there
1447 SHALL be only one *ThisPartyActionBinding* element whose *action* attribute has a given value.
1448 ~~Not sure if this constraint is desirable. What if the party is willing to use either HTTP or SMTP~~
1449 ~~for a particular action?~~

1450

1451 Within a *WillInitiate* element or a *WillRespond* element, when both the
1452 *ThisPartyActionBinding* and *OtherPartyActionBinding* elements are present (i.e., in a *CPA*),
1453 then both MUST have the same *action* attribute value. Also, the *DeliveryChannel* and *Packaging*
1454 that that they reference MUST be compatible.

1455

1456

7.5.12.18.5.12.1 action attribute

1458 The value of the REQUIRED *action* attribute is a string that identifies the business document
1459 exchange to be associated with the *DeliveryChannel* identified by the *channelId* attribute. The
1460 value of the *action* attribute SHALL be used as the value of the *Action* element in the ebXML
1461 *Message Header*[ebMS] or a similar element in the *Message Header* of an alternative *message*
1462 service. The purpose of the *action* attribute is to provide a mapping between the hierarchical
1463 naming associated with a *Business Process/Application* and the *Action* element in the ebXML
1464 *Message Header*[ebMS]. This mapping MAY be implemented by using the *ActionContext*
1465 element. See NOTE in section 7.5.4 regarding alternative *Business Collaboration* descriptions.

1466

1467 Business signals, when sent individually (i.e., not bundled with response documents in
1468 synchronous reply mode), SHALL use the values *ReceiptAcknowledgment*,
1469 *AcceptanceAcknowledgment*, ~~and-or~~ *Exception* as the value of their *action* attribute.

1470

1471 **[7.5.12.28.5.12.2](#) channelId attribute**

1472 The REQUIRED *channelId* attribute is an [XML] IDREF that identifies the *DeliveryChannel*
1473 element to be associated with the *Message* identified by the *action* attribute.

1474

1475 **[7.5.12.38.5.12.3](#) packageId attribute**

1476 The REQUIRED *packageId* attribute is an [XML] IDREF that identifies the *Packaging* element
1477 to be associated with the *Message* identified by the *action* attribute.

1478

1479 **[7.5.12.48.5.12.4](#) xlink:href attribute**

1480 The IMPLIED *xlink:href* attribute MAY be present. If present, it SHALL provide an absolute
1481 [XPOINTER] URI expression that specifically identifies the *RequestingBusinessActivity* or
1482 *RespondingBusinessActivity* element within the associated *Process-Specification*
1483 document[ebBPSS] that is identified by the *ProcessSpecification* element.

1484

1485 **[7.5.12.58.5.12.5](#) xlink:type attribute**

1486 The IMPLIED *xlink:type* attribute has a FIXED value of "simple". This identifies the element as
1487 being an [XLINK] simple link.

1488

1489 **[7.5.138.5.13](#) ActionContext element**

1490 The *ActionContext* element provides a mapping from the *action* attribute in the
1491 *ThisPartyActionBinding* and *OtherPartyActionBinding* elements to the corresponding *Business*
1492 *Process* implementation-specific naming strategy, if any. If the *Process-Specification* document
1493 is defined by the ebXML Business Process Specification Schema[ebBPSS], the *ActionContext*
1494 element MUST be present.

1495

1496 Any business process/application ~~layer-implementation~~ MAY use a combination of information
1497 in the *action* attribute and the *ActionContext* elements to make message routing decisions. If
1498 using ~~alternate-alternative~~ *Business-Collaboration* description schemas, the *action* attribute of
1499 the parent *ActionBinding* element and/or *wildcard* element within the *ActionContext* element
1500 MAY be used to make routing decisions ~~at the application layer~~ above the level of the *Message*
1501 *Service Handler*.

1502

1503 The *ActionContext* element MAY have the following elements:

- 1504 • an OPTIONAL *CollaborationActivity* element
- 1505 • an OPTIONAL *#wildcard* element

1506

1507 The *ActionContext* element also has the following attributes:

- 1508 • a REQUIRED *binaryCollaboration* attribute,
- 1509 • a REQUIRED *businessTransactionActivity* attribute,
- 1510 • a REQUIRED *requestOrResponseAction* attribute.

1511

1512 **[7.5.13.18.5.13.1](#) binaryCollaboration attribute**

1513 The REQUIRED *binaryCollaboration* attribute is a string that identifies the

1514 **BinaryCollaboration** for which the parent **ThisPartyActionBinding** (or
 1515 **OtherPartyActionBinding**) is defined. If the *Process-Specification* document is defined by the
 1516 ebXML Business Process Specification Schema[ebBPSS], then the value of the
 1517 **binaryCollaboration** attribute- MUST match the value of the **name** attribute of the
 1518 **BinaryCollaboration** element as defined in the ebXML Business Process Specification
 1519 Schema[ebBPSS].

1520

1521 **7.5.13.28.5.13.2 businessTransactionActivity attribute**

1522 The REQUIRED **businessTransactionActivity** attribute is a string that identifies the *Business*
 1523 *Transaction* for which the parent **ThisPartyActionBinding** (or **OtherPartyActionBinding**) is
 1524 defined. If the *Process-Specification* document is defined by the ebXML Business Process
 1525 Specification Schema[ebBPSS], the value of the **businessTransactionActivity** attribute MUST
 1526 match the value of the **name** attribute of the *BusinessTransactionActivity* element, whose parent
 1527 is the *Binary Collaboration* referred to by the **binaryCollaboration** attribute.

1528

1529 **7.5.13.38.5.13.3 requestOrResponseAction attribute**

1530 The REQUIRED **requestOrResponseAction** attribute is a string that identifies either the
 1531 *Requesting or Responding Business Activity* for which the parent **ThisPartyActionBinding** (or
 1532 **OtherPartyActionBinding**) is defined. For a **ThisPartyActionBinding** (or
 1533 **OtherPartyActionBinding**) defined for the request side of a message exchange, if the *Process-*
 1534 *Specification* document is defined by the ebXML Business Process Specification Schema
 1535 [ebBPSS], the value of the **requestOrResponseAction** attribute MUST match the value of the
 1536 **name** attribute of the *RequestingBusinessActivity* element corresponding to the
 1537 *BusinessTransaction* specified in the **businessTransactionActivity** attribute. Similarly, for the
 1538 response side of a message exchange, the value of the **requestOrResponseAction** attribute
 1539 MUST match the value of the **name** attribute of the *RespondingBusinessActivity* element
 1540 corresponding to the *BusinessTransaction* specified in the **businessTransactionActivity** attribute,
 1541 as defined in the ebXML Business Process Specification Schema[ebBPSS].

1542 **7.5.148.5.14 CollaborationActivity element**

1543 The **CollaborationActivity** element supports the *ActionContext* element by providing the ability
 1544 to map any nested *BinaryCollaborations* as defined in the ebXML Business Process
 1545 Specification Schema[ebBPSS] to the **action** attribute. The **CollaborationActivity** element
 1546 MUST be present when the *Binary Collaboration* referred to by the **binaryCollaboration**
 1547 attribute has a *Collaboration Activity* defined in the business process definition.

1548

1549 An example of the **CollaborationActivity** element is:

1550

```
1551     <tp:CollaborationActivity
1552         tp:name="Credit Check">
1553         <tp:CollaborationActivity
1554             tp:name="Credit History Check">
1555         </tp:CollaborationActivity>
1556     </tp:CollaborationActivity>
```

1557

1558 The **CollaborationActivity** element MAY have an OPTIONAL child **CollaborationActivity**
 1559 element to indicate further nesting of *Binary Collaborations*.

1560

1561 The *CollaborationActivity* element also has one attribute:

- 1562 • a REQUIRED *name* attribute

1563

1564 [7.5.14.18.5.14.1](#) name attribute

1565 The REQUIRED *name* attribute is a string that identifies the *Collaboration Activity* included in
1566 the *Binary Collaboration*. If the *Process-Specification* document is defined by the ebXML
1567 Business Process Specification Schema[ebBPSS], the value of the *name* attribute MUST match
1568 the value of the *name* attribute of the *CollaborationActivity* within the *BinaryCollaboration*, as
1569 defined in the ebXML Business Process Specification Schema[ebBPSS].

1570

1571 [7.5.158.5.15](#) Certificate element

1572 The *Certificate* element defines certificate information for use in this *CPP*. One or more
1573 *Certificate* elements MAY be provided for use in the various security functions in the *CPP*. An
1574 example of the *Certificate* element is:

1575

```
1576 <tp:Certificate tp:certId="CompanyA_SigningCert">  
1577 <ds:KeyInfo>. . .</ds:KeyInfo>  
1578 </tp:Certificate>
```

1579

1580 The *Certificate* element has a single REQUIRED attribute: *certId*. The *Certificate* element has a
1581 single child element: *ds:KeyInfo*.

1582

1583 [7.5.15.18.5.15.1](#) certId attribute

1584

1585 The REQUIRED *certId* attribute is an [XML] ID that is referred to by a *CertificateRef* element
1586 elsewhere in the *CPP*. Here is an example of how a *CertificateRef* would refer to the *Certificate*
1587 element shown in the previous section:

1588

```
1589 <tp:SigningCertificateRef tp:certId="CompanyA_SigningCert"/>
```

1590

1591 [7.5.15.28.5.15.2](#) ds:KeyInfo element

1592 The *ds:KeyInfo* element defines the certificate information. The content of this element and any
1593 subelements are defined by the XML Digital Signature specification[XMLDSIG].

1594

1595 NOTE: Software for creation of *CPPs* and *CPAs* **MAY-MUST** recognize the *ds:KeyInfo*
1596 element and insert the subelement structure necessary to define the certificate.

1597

1598 [7.5.168.5.16](#) SecurityDetails element

1599 The *SecurityDetails* element defines a set of *TrustAnchors* and an associated *SecurityPolicy* for
1600 use in this *CPP*. One or more *SecurityDetails* elements MAY be provided for use in the various
1601 security functions in the *CPP*. An example of the *SecurityDetails* element is:

1602

```
1603 <tp:SecurityDetails tp:securityId="CompanyA_MessageSecurity">  
1604 <tp:TrustAnchors tp:trustId="MessageTrustAnchors">
```

```

1605         <tp:AnchorCertificateRef tp:certId="TrustedRootCertA3"/>
1606         <tp:AnchorCertificateRef tp:certId="TrustedRootCertA5"/>
1607     </tp:TrustAnchors>
1608     <tp:SecurityPolicy> ... </tp:SecurityPolicy>
1609 </tp:SecurityDetails>

```

1610

1611 The *SecurityDetails* element has a ~~n-**OPTIONAL**~~ zero or more *TrustAnchors* elements that
 1612 ~~identifies-identify~~ a set of root certificates that are trusted by the *Party*. It also has a ~~an~~
 1613 ~~OPTIONAL~~ zero or more *SecurityPolicy* elements.

1614

1615 The *SecurityDetails* element allows agreement to be reached on what root certificates will be
 1616 used in checking the validity of the other *Party*'s certificates. It can also specify policy regarding
 1617 operation of the public key infrastructure.

1618

1619 The *SecurityDetails* element has one attribute:

- 1620 • A REQUIRED *securityId* attribute

1621

1622 7.5.16.18.5.16.1 *securityId* attribute

1623

1624 The REQUIRED *securityId* attribute is an [XML] ID that is referred to by a *SecurityDetailsRef*
 1625 element elsewhere in the *CPP*. Here is an example of how a *SecurityDetailsRef* would refer to
 1626 the *SecurityDetails* element shown in the previous section:

1627

```

1628     <tp:SigningSecurityDetailsRef
1629 tp:securityId="CompanyA_MessageSecurity"/>

```

1630

1631 7.5.178.5.17 *TrustAnchors* element

1632 The ~~OPTIONAL~~ *TrustAnchors* element contains one or more *AnchorCertificateRef* elements,
 1633 each of which refers to a *Certificate* element (under *PartyInfo*) that represents a root certificate
 1634 trusted by this *Party*. These trusted roots are used in the process of certificate path validation. If a
 1635 certificate in question does not "chain" to one of this *Party*'s trust anchors, it is considered
 1636 invalid.

1637

1638 7.5.188.5.18 *SecurityPolicy* element

1639 The ~~OPTIONAL~~ *SecurityPolicy* element is a placeholder future apparatus that will enable the
 1640 *Party* to specify its policy and compliance regarding specific components of its public key
 1641 infrastructure. For example, it might stipulate revocation checking procedures or constraints
 1642 related to name, usage, or path length.

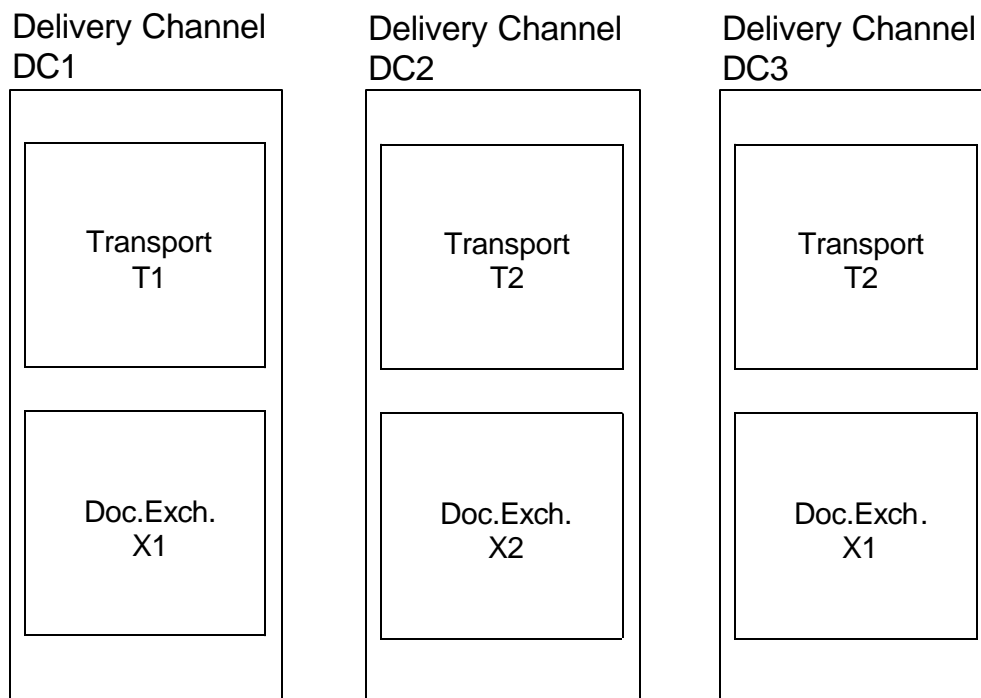
1643

1644 7.5.198.5.19 *DeliveryChannel* element

1645 A delivery channel is a combination of a *Transport* element and a *DocExchange* element that
 1646 describes the *Party's Message* communication characteristics. The *CPP* SHALL contain one or
 1647 more *DeliveryChannel* elements, one or more *Transport* elements, and one or more
 1648 *DocExchange* elements. Each delivery channel ~~MAY~~ SHALL refer to any combination of a

1649 **DocExchange** element and a **Transport** element. The same **DocExchange** element or the same
 1650 **Transport** element **MAY-can be** referred to by more than one delivery channel. Two delivery
 1651 channels **MAY-can** use the same transport protocol and the same document-exchange protocol
 1652 and differ only in details such as communication addresses or security definitions. Figure 5
 1653 illustrates three delivery channels.

Figure 5: Three Delivery Channels



1654
 1655 The delivery channels have ID attributes with values "DC1", "DC2", and "DC3". Each delivery
 1656 channel contains one transport definition and one document-exchange definition. Each transport
 1657 definition and each document-exchange definition also has a name as shown in the figure. Note
 1658 that delivery channel DC3 illustrates that a delivery channel **MAY-can** refer to the same transport
 1659 definition and document-exchange definition used by other delivery channels but a different
 1660 combination. In this case delivery channel DC3 is a combination of transport definition T2 (also
 1661 referred to by delivery channel DC2) and document-exchange definition X1 (also referred to by
 1662 delivery channel DC1).

1663
 1664 A specific delivery channel SHALL be associated with each **PartyInfo** element,
 1665 **OverrideMshActionBinding** element, **ServiceBinding** element, **ThisPartyActionBinding**, or
 1666 **OtherPartyActionBinding** element (*action* attribute). Following is the delivery-channel syntax.

```

1667 <tp:DeliveryChannel
1668     tp:channelId="channel1"
1669     tp:transportId="transport1"
1670     tp:docExchangeId="docExchange1"
1671     <tp:BusinessProcessCharacteristics
1672         tp:isNonRepudiationRequired="true"
1673 
```

```

1674         tp:isNonRepudiationReceiptRequired="false"
1675         tp:osSecureTransportRequired="true"
1676         tp:isConfidential="true"
1677         tp:isAuthenticated="true"
1678         tp:isAuthorizationRequired="false"/>
1679     <tp:MessagingCharacteristics
1680         tp:syncReplyMode="none"
1681         tp:ackRequested="always"
1682         tp:ackSignatureRequested="always"
1683         tp:duplicateElimination="always"
1684         tp:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH"/>
1685 </tp:DeliveryChannel>

```

1686

1687 Each *DeliveryChannel* element identifies one *Transport* element and one *DocExchange* element
 1688 that make up a single delivery channel definition.

1689

1690 The *DeliveryChannel* element has the following attributes:

- 1691 • a REQUIRED *channelId* attribute,
- 1692 • a REQUIRED *transportId* attribute,
- 1693 • a REQUIRED *docExchangeId* attribute.

1694

1695 The *DeliveryChannel* element has two REQUIRED child elements,
 1696 *BusinessProcessCharacteristics* and *MessagingCharacteristics*.

1697

1698 7.5.19.18.5.19.1 channelId attribute

1699 The *channelId* attribute is an [XML] ID attribute that uniquely identifies the *DeliveryChannel*
 1700 element for reference, using IDREF attributes, from other parts of the *CPP* or *CPA*.

1701

1702 7.5.19.28.5.19.2 transportId attribute

1703 The *transportId* attribute is an [XML] IDREF that identifies the *Transport* element that defines
 1704 the transport characteristics of the delivery channel. It MUST have a value that is equal to the
 1705 value of a *transportId* attribute of a *Transport* element elsewhere within the *CPP* document.

1706

1707 7.5.19.38.5.19.3 docExchangeId attribute

1708 The *docExchangeId* attribute is an [XML] IDREF that identifies the *DocExchange* element that
 1709 defines the document-exchange characteristics of the delivery channel. It MUST have a value
 1710 that is equal to the value of a *docExchangeId* attribute of a *DocExchange* element elsewhere
 1711 within the *CPP* document.

1712

1713 7.5.208.5.20 BusinessProcessCharacteristics element

1714 The *BusinessProcessCharacteristics* element describes the security characteristics and other
 1715 attributes of the delivery channel, as derived from the *ProcessSpecification(s)* whose messages
 1716 are transported using the delivery channel. The attributes of the *BusinessProcessCharacteristics*
 1717 element, ~~except *syncReplyMode*~~, MAY be used to override the values of the corresponding
 1718 attributes in the *Process-Specification* document.

1719

1720 See NOTE in Section [8.5.47.5.4](#) regarding alternative *Business-Collaboration* descriptions.

1721
1722 CPP and CPA composition tools and CPA deployment tools SHALL check the delivery channel
1723 definition (transport and document-exchange) for consistency with these attributes.

1724
1725 The *BusinessProcessCharacteristics* element has the following attributes:

- 1726 • an IMPLIED *isNonRepudiationRequired* attribute,
- 1727 • an IMPLIED *isNonRepudiationReceiptRequired* attribute,
- 1728 • an IMPLIED *isSecureTransportRequired* attribute,
- 1729 • an IMPLIED *isConfidential* attribute,
- 1730 • an IMPLIED *isAuthenticated* attribute,
- 1731 • an IMPLIED *isAuthorizationRequired* attribute.

1732
1733
1734 **7.5.20.18.5.20.1 isNonRepudiationRequired attribute**

1735 The *isNonRepudiationRequired* attribute is a Boolean with possible values of "true" and
1736 "false". If the value is "true" then the delivery channel REQUIRES-MUST specify that the
1737 *Message* is to be digitally signed by-using the certificate of the *Party* that sent the *Message*.

1738
1739 **7.5.20.28.5.20.2 isNonRepudiationReceiptRequired attribute**

1740 The *isNonRepudiationReceipt* Required attribute is a Boolean with possible values of "true"
1741 and "false". If the value is "true" then the delivery channel REQUIRES-MUST specify that the
1742 *Message* is to be acknowledged by a digitally signed *Message*, signed by-using the certificate of
1743 the *Party* that received the *Message*, that includes the digest of the *Message* being
1744 acknowledged.

1745
1746 **7.5.20.38.5.20.3 isSecureTransportRequired attribute**

1747 The *isSecureTransportRequired* attribute is a Boolean with possible values of "true" and
1748 "false". If the value is "true" then it indicates that the delivery channel uses a secure transport
1749 protocol such as [SSL] or [IPSEC].

1750
1751 **7.5.20.48.5.20.4 isConfidential attribute**

1752 The *isConfidential* attribute has the possible values of "none", "transient", "persistent",
1753 "transient-and-persistent". If the value is "persistent" or "transient-and-persistent" then it
1754 indicates that the delivery channel REQUIRES-MUST specify that the *Message* is to be
1755 encrypted in a persistent manner. It MUST be encrypted above the level of the transport
1756 messaging service and delivered, encrypted, to the application.

1757
1758 **7.5.20.58.5.20.5 isAuthenticated attribute**

1759 The *isAuthenticated* attribute is a Boolean with possible values of "true" and "false". If the
1760 value is "true" then it indicates that the delivery channel REQUIRES-MUST specify that the
1761 sender of the *Message* is to be authenticated before delivery to the application.

1762
1763 **7.5.20.68.5.20.6 isAuthorizationRequired attribute**

1764 The *isAuthorizationRequired* attribute is a Boolean with possible of values of "true" and
1765 "false". If the value is "true" then it indicates that the delivery channel REQUIRES-MUST

1766 specify that the sender of the *Message* is to be authorized before delivery to the application.
1767

1768 **7.5.218.5.21 MessagingCharacteristics element**

1769 The ~~*MessagingProcessCharacteristics*~~*MessagingCharacteristics* element describes the quality
1770 ~~of-service~~ attributes associated with messages delivered over a given delivery channel. The
1771 collaborating parties ~~MAY-can~~ stipulate that these attributes be fixed for all messages sent
1772 through the delivery channel, or they ~~MAY-can~~ agree that these attributes are to be variable on a
1773 “per message” basis.

1774
1775 CPP and CPA composition tools and CPA deployment tools SHALL check the delivery channel
1776 definition (transport and document-exchange) for consistency with these attributes.
1777

1778 The ~~*MessagingProcessCharacteristics*~~*MessagingCharacteristics* element has the following
1779 attributes:

- 1780 • An IMPLIED *syncReplyMode* attribute,
- 1781 • an IMPLIED *ackRequested* attribute,
- 1782 • an IMPLIED *ackSignatureRequested* attribute,
- 1783 • an IMPLIED *duplicateElimination* attribute,
- 1784 • an IMPLIED *actor* attribute.

1785 1786 **7.5.21.18.5.21.1 syncReplyMode attribute**

1787 The *syncReplyMode* attribute is an enumeration comprised of the following possible values:

- 1788 • "mshSignalsOnly"
- 1789 • "signalsOnly"
- 1790 • "responseOnly"
- 1791 • "signalsAndResponse"
- 1792 • "none"

1793
1794 This attribute, when present, indicates what the sending application expects in a synchronous
1795 response when bound to a synchronous communication protocol such as HTTP. The value of
1796 "mshSignalsOnly" indicates that the response returned (on the HTTP 200 response in the case of
1797 HTTP) will only contain standalone *Message Service Handler (MSH)* level messages like
1798 Acknowledgment (for Reliable Messaging) and Error messages. All other application level
1799 responses are to be returned asynchronously (using a *DeliveryChannel* determined by the
1800 Service and Action in question). The value of "signalsOnly" indicates that the response returned
1801 (on the HTTP 200 response in the case of HTTP) will only include one or more *Business* signals
1802 as defined in the *Process-Specification* document[ebBPSS], plus any piggybacked MSH level
1803 signals, but not a *Business-response Message*. If the *Process-Specification* calls for the use of a
1804 *Business-response Message*, then the latter MUST be returned asynchronously. The value of
1805 "responseOnly" indicates that any *Business* signals indicated in the *Process Specification* are to
1806 be omitted and only the *Business-response Message* will be returned synchronously, plus any
1807 piggybacked MSH level signals. The value of "signalsAndResponse" indicates that the
1808 application will synchronously return the *Business-response Message* in addition to one or more
1809 *Business* signals, plus any piggybacked *MSH* level signals. The value of "none", which is the

1810 implied default value in the absence of the *syncReplyMode* attribute, indicates that neither the
1811 *Business-response Message* nor any *Business* signal(s) will be returned synchronously. In this
1812 case, the *Business-response Message* and any *Business* signals will be returned as separate
1813 asynchronous responses.

1814
1815 The ebXML *Message Service's SyncReply* element is included in the SOAP Header whenever
1816 the *syncReplyMode* attribute has a value other than "none". If the delivery channel identifies a
1817 transport protocol that has no synchronous capabilities (such as SMTP), the
1818 *BusinessProcessCharacteristics* element SHALL NOT have a *syncReplyMode* attribute with a
1819 value other than "none".

1820
1821 When the value of the *syncReplyMode* attribute is other than "none", a synchronous
1822 delivery channel SHALL be NOTE: It is assumed that a synchronous *DeliveryChannel*
1823 is used to exchange all messages necessary for conducting a business transaction. If the
1824 *Process Specification* calls for the use of -non-repudiation of receipt for the response
1825 message, then the initiator is expected to return a signed *Receipt Acknowledgment* signal
1826 for the responder's response message. However, this is incompatible with the
1827 *syncReplyMode* values "signalsAndResponse" and "responseOnly", which make no
1828 provision for the return for such a signal.

1829
1830 NOTE: For HTTP 1.1 clients and servers, two HTTP requests and replies will be sent and
1831 received. In other cases, non-repudiation of receipt for the response message could be
1832 incompatible with the *syncReplyMode* values "signalsAndResponse" and
1833 "responseOnly".

1834
1835
1836 ~~If the delivery channel identifies a transport protocol that has no synchronous capabilities (such~~
1837 ~~as SMTP), the *BusinessProcessCharacteristics* element SHALL NOT have a *syncReplyMode*~~
1838 ~~attribute with a value other than "none".~~

1840 7.5.21.28.5.21.2 **ackRequested** attribute

1841 The IMPLIED *ackRequested* attribute is an enumeration comprised of the following possible
1842 values:

- 1843 • "always"
- 1844 • "never"
- 1845 • "perMessage"

1846
1847 This attribute has the default value "perMessage" meaning that the *AckRequested* element in the
1848 SOAP Header ~~MAY be~~ present or absent on a "per message" basis. If this attribute is set to
1849 "always", then every message sent over the delivery channel MUST have an *AckRequested*
1850 element in the SOAP Header. If this attribute is set to "never", then every message sent over the
1851 delivery channel MUST NOT have an *AckRequested* element in the SOAP Header.

1853 7.5.21.38.5.21.3 **ackSignatureRequested** attribute

1854 The IMPLIED *ackSignatureRequested* attribute is an enumeration comprised of the following

1855 values:

- 1856 • "always"
- 1857 • "never"
- 1858 • "perMessage"

1859

1860 This attribute determines how the *signed* attribute within the *AckRequested* element in the SOAP
1861 Header is to be set. It has the default value "perMessage" meaning that the *signed* attribute in the
1862 *AckRequested* element within the SOAP Header ~~MAY be~~ set to "true" or "false" on a "per
1863 message" basis. If this attribute is set to "always", then every message sent over the delivery
1864 channel that has an *AckRequested* element in the SOAP Header MUST have its *signed* attribute
1865 set to "true". If this attribute is set to "never", then every message sent over the delivery channel
1866 that has an *AckRequested* element in the SOAP Header MUST have its *signed* attribute set to
1867 "false".

1868

1869 7.5.21.48.5.21.4 duplicateElimination attribute

1870 The IMPLIED *duplicateElimination* attribute is an enumeration comprised of the following
1871 values:

- 1872 • "always"
- 1873 • "never"
- 1874 • "perMessage"

1875

1876 This attribute determines whether the *DuplicateElimination* element within the *MessageHeader*
1877 element in the SOAP Header is to be present. It has the default value "perMessage" meaning that
1878 the *DuplicateElimination* element within the SOAP Header ~~MAY be~~ present or absent on a
1879 "per message" basis. If this attribute is set to "always", then every message sent over the delivery
1880 channel MUST have a *DuplicateElimination* element in the SOAP Header. If this attribute is set
1881 to "never", then every message sent over the delivery channel MUST NOT have a
1882 *DuplicateElimination* element in the SOAP Header.

1883

1884 7.5.21.58.5.21.5 actor attribute

1885 The IMPLIED *actor* attribute is an enumeration of the following values:

- 1886 • "urn:oasis:names:tc:ebxml-msg:actor:nextMSH"
- 1887 • "urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH"

1888

1889 This is a URI that will be used as the value for the *actor* attribute in the *AckRequested* element
1890 in case the latter is present in the SOAP Header, as governed by the *ackRequested* attribute
1891 within the *MessagingCharacteristics* element in the *CPA*.

1892

1893 7.5.228.5.22 Transport element

1894 The *Transport* element defines the *Party's* network communication capabilities. One or more
1895 *Transport* elements MUST be present in a *CPP*, each of which describes a mechanism the *Party*
1896 uses to send messages, a mechanism it uses to receive messages, or both. The following example
1897 illustrates the structure of a typical *Transport* element:

1898


```

1899 <tp:Transport tp:transportId="transportA1">
1900   <tp:TransportSender> <!-- 0 or 1 times -->
1901     <tp:TransportProtocol tp:version="1.1">HTTP</tp:Protocol>
1902     <tp:TransportClientSecurity>
1903       <tp:TransportSecurityProtocol tp:version="3.0">
1904         SSL
1905       </tp:TransportSecurityProtocol>
1906       <tp:ClientCertificateRef tp:certId="CompanyA_ClientCert"/>
1907       <tp:ServerSecurityDetailsRef
1908         tp:securityId="CompanyA_TransportSecurity"/>
1909     </tp:TransportClientSecurity>
1910   </tp:TransportSender>
1911   <tp:TransportReceiver> <!-- 0 or 1 times -->
1912     <tp:TransportProtocol tp:version="1.1">HTTP</tp:Protocol>
1913     <tp:Endpoint
1914       tp:uri=https://www.CompanyA.com/servlets/ebxmlhandler
1915       tp:type="allPurpose"/>
1916     <tp:TransportServerSecurity>
1917       <tp:TransportSecurityProtocol tp:version="3.0">
1918         SSL
1919       </tp:TransportSecurityProtocol>
1920       <tp:ServerCertificateRef tp:certId="CompanyA_ServerCert"/>
1921       <tp:ClientSecurityDetailsRef
1922         tp:securityId="CompanyA_TransportSecurity"/>
1923     </tp:TransportServerSecurity>
1924   </tp:TransportReceiver>
1925 </tp:Transport>
1926

```

The **Transport** element consists of ~~an OPTIONAL~~zero or one **TransportSender** element and ~~an OPTIONAL~~zero or one **TransportReceiver** element.

A **Transport** that contains both **TransportSender** and **TransportReceiver** elements is said to be *bi-directional* in that it can be used for send and receiving messages. If the *Party* prefers to communicate in synchronous mode (where replies are returned over the same TCP connections messages are sent on), its *CPP* MUST provide a **ServiceBinding** that contains **ActionBindings** that are bound to a **DeliveryChannel** that uses a bi-directional **Transport**.

A bi-directional **Transport** whose **TransportSender** and **TransportReceiver** elements use different transport protocols is said to be *asymmetric*. In a *CPA*, an asymmetric **Transport** offered by one *Party* ~~will~~MUST be matched with a complementary asymmetric **Transport** in the other *Party's* *CPP*. For example, a **Transport** composed of an HTTP sender and an SMTP receiver would match with a **Transport** containing an SMTP sender and HTTP receiver.

NOTE: The ability of a transport to support bi-directional traffic does not imply that it will be used for synchronous communication.

A **Transport** that contains either a **TransportSender** or a **TransportReceiver** element, but not both, is said to be *unidirectional*. A unidirectional **Transport** can only be used for sending or receiving messages (not both) depending on which element it includes.

A *CPP* contains as many **Transport** elements as are needed to fully express the *Party's* inbound Collaboration-Protocol Profile and Agreement Specification

1950 and outbound communication capabilities. If, for example, the *Party* can send and receive
 1951 messages via HTTP and SMTP, its *CPP* would contain a *Transport* element containing its HTTP
 1952 properties and another *Transport* element containing its SMTP properties.

1953

1954 The *Transport* element has

- 1955 • a REQUIRED *transportId* attribute

1956

1957 **7.5.22.18.5.22.1 transportId attribute**

1958 The REQUIRED *transportId* attribute is an [XML] ID that is referred to by a *DeliveryChannel*
 1959 element elsewhere in the *CPP*. Here is an example of a *DeliveryChannel* that refers to the
 1960 *Transport* element shown in the previous section:

1961

```
1962 <tp:DeliveryChannel tp:channelId="channelA1"
1963     tp:transportId="transportA1"
1964     tp:docExchangeId="docExchangeA1">
1965     <tp:BusinessProcessCharacteristics . . . />
1966     <tp:MessagingCharacteristics . . . />
1967 </tp:DeliveryChannel>
1968
```

1969 **7.5.238.5.23 TransportSender element**

1970 The ~~OPTIONAL~~ *TransportSender* element contains properties related to the sending side of a
 1971 *DeliveryChannel*. Its REQUIRED *TransportProtocol* element specifies the transport protocol
 1972 that will be used for sending messages. The ~~OPTIONAL~~ *TransportClientSecurity* element
 1973 defines the *Party*'s provisions for client-side transport layer security.

1974

1975 The *TransportSender* element has no attributes.

1976

1977

1978 **7.5.248.5.24 TransportProtocol element**

1979 The *TransportProtocol* element identifies a transport protocol that the *Party* is capable of using
 1980 to send or receive *Business* data. The IMPLIED *version* attribute identifies the specific version
 1981 of the protocol.

1982

1983 NOTE: It is the aim of this specification to enable support for any transport capable of
 1984 carrying MIME content using the vocabulary defined herein.

1985

1986 **7.5.258.5.25 TransportClientSecurity element**

1987 The ~~OPTIONAL~~ *TransportClientSecurity* element provides information about this *Party*'s
 1988 transport client needed by the other *Party*'s transport server to enable a- secure connection to be
 1989 established between the two. It contains a REQUIRED *TransportSecurityProtocol* element, ~~an~~
 1990 ~~OPTIONAL~~ zero or one *ClientCertificateRef* element, and ~~an~~ ~~OPTIONAL~~ zero or one
 1991 *ServerSecurityDetailsRef* element.

1992

1993 **NOTE:** In asynchronous messaging mode, the sender will always be a client to the receiver's
1994 server. ~~But in~~ In synchronous messaging mode, the MSH-level reply (and maybe a bundled
1995 business signal and/or business response) is sent back over the same connection the initial
1996 business message arrived on. In such cases, where the sender is the server and the receiver is the
1997 client and the connection already exists, the sender's *TransportClientSecurity* and the receiver's
1998 *TransportServerSecurity* elements ~~are~~ **SHALL be** ignored.

1999 **7.5.268.5.26 TransportSecurityProtocol element**

2000 The *TransportSecurityProtocol* element identifies the transport layer security protocol that is
2001 supported by the parent *Transport*. The IMPLIED *version* attribute identifies the specific version
2002 of the protocol.

2003 **7.5.26.18.5.26.1 Specifics for HTTP**

2004 For encryption with HTTP, the protocol is SSL[SSL] (Secure Socket Layer) Version 3.0, which
2005 uses public-key encryption.
2006
2007

2008 **7.5.278.5.27 ClientCertificateRef element**

2009 The ~~OPTIONAL~~ *ClientCertificateRef* element identifies the certificate to be used by the client's
2010 transport security module. The REQUIRED IDREF attribute *certId* identifies the certificate to be
2011 used by referring to the *Certificate* element (under *PartyInfo*) that has the matching ID attribute
2012 value. An SSL-capable HTTP client, for example, uses this certificate to authenticate itself with
2013 receiver's secure HTTP server.

2014
2015 The *ClientCertificateRef* element, if present, indicates that mutual authentication between client
2016 and server (i.e., initiator and responder of the HTTP connection) **MUST** be performed.

2017 The *ClientCertificateRef* element has

- 2018 • A REQUIRED *certId* attribute

2019 **7.5.288.5.28 ServerSecurityDetailsRef element**

2020
2021 The ~~OPTIONAL~~ *ServerSecurityDetailsRef* element identifies the trust anchors and security
2022 policy that this *Party* will apply to the other *Party*'s server authentication certificate.

2023
2024 The *ServerSecurityDetailsRef* element has

- 2025 • A REQUIRED *securityId* attribute

2026 **7.5.298.5.29 TransportReceiver element**

2027
2028 The ~~OPTIONAL~~ *TransportReceiver* element contains properties related to the receiving side of
2029 a *DeliveryChannel*. Its REQUIRED *TransportProtocol* element specifies the transport protocol
2030 that will be used for receiving messages. One or more REQUIRED *Endpoint* elements specify
2031 logical addresses where messages can be received. ~~The OPTIONAL~~ **Zero or one**
2032 *TransportServerSecurity* element defines the *Party*'s provisions for server-side transport layer
2033
2034

2035 security.

2036

2037 The *TransportReceiver* element has no attributes.

2038

2039 **7.5.308.5.30 Endpoint element**

2040 One or more *Endpoint* elements SHALL be provided for each *TransportReceiver* element. Each
2041 *Endpoint* specifies a logical address and an indication of what kinds of messages can be received
2042 at that location.

2043

2044 Each *Endpoint* has the following attributes:

- 2045 • a REQUIRED *uri* attribute
- 2046 • an ~~OPTIONAL-IMPLIED~~ *type* attribute

2047

2048 **7.5.30.18.5.30.1 uri attribute**

2049 The REQUIRED *uri* attribute specifies a ~~Uniform Resource Indicator~~URI identifying the address
2050 of a resource. The value of the *uri* attribute SHALL conform to the syntax for expressing URIs
2051 as defined in [RFC2396].

2052

2053 **7.5.30.28.5.30.2 type attribute**

2054 The ~~OPTIONAL~~*type* attribute identifies the purpose of this endpoint. The value of *type* is an
2055 enumeration; permissible values are "login", "request", "response", "error", and "allPurpose".
2056 There can be, at most, one of each. ~~The type attribute MAY be omitted. If it the type attribute~~ is
2057 omitted, its value defaults to "allPurpose". The "login" endpoint ~~MAY be is~~ used for the address
2058 for the initial *Message* between the two *Parties*. The "request" and "response" endpoints are
2059 used for request and response *Messages*, respectively. The "error" endpoint ~~MAY be is~~ used as
2060 the address for error *Messages* issued by the messaging service. If no "error" endpoint is
2061 defined, these error *Messages* SHALL be sent to the "response" address, if defined, or to the
2062 "allPurpose" endpoint. To enable error *Messages* to be received, each *Transport* element
2063 SHALL contain at least one endpoint of type "error", "response", or "allPurpose".

2064

2065 **7.5.318.5.31 TransportServerSecurity element**

2066 The ~~OPTIONAL-TransportServerSecurity~~ element provides information about this *Party's*
2067 transport client needed by the other *Party's* transport server to enable a- secure connection to be
2068 established between the two. It contains a REQUIRED *TransportSecurityProtocol* element, a ~~n~~
2069 ~~OPTIONAL-REQUIRED~~ *ServerCertificateRef* element, and ~~an~~ ~~OPTIONAL-zero or one~~
2070 *ClientSecurityDetailsRef* element.

2071

2072 NOTE: See the note in Section [8.5.257-5.25](#) regarding the relevance of the
2073 *TransportServerSecurity* element when synchronous replies are in use.

2074

2075 **7.5.328.5.32 ServerCertificateRef element**

2076 The OPTIONAL *ServerCertificateRef* element identifies the certificate to be used by the
2077 server's transport security module. The REQUIRED IDREF attribute *certId* identifies the

2078 certificate to be used by referring to the *Certificate* element (under *PartyInfo*) that has the
2079 matching ID attribute value. An SSL-enabled HTTP server, for example, uses this certificate to
2080 authenticate itself with the sender's SSL client.

2081
2082 The *ServerCertificateRef* element MUST be present if the transport security protocol uses
2083 certificates. It MAY be omitted otherwise (e.g. if authentication is by password).

2084
2085 The *ServerCertificateRef* element has

- A REQUIRED *certId* attribute

2088 [7.5.338.5.33](#) *ClientSecurityDetailsRef* element

2089 The OPTIONAL *ClientSecurityDetailsRef* element identifies the trust anchors and security
2090 policy that this *Party* will apply to the other *Party*'s client authentication certificate.

2091
2092 The *ClientSecurityDetailsRef* element has

- A REQUIRED *securityId* attribute

2095 [7.5.348.5.34](#) Transport protocols

2096 In the following sections, we discuss the specific details of each supported transport protocol.

2098 [7.5.34.18.5.34.1](#) HTTP

2099 HTTP is Hypertext Transfer Protocol[HTTP]. For HTTP, the endpoint is a URI that SHALL
2100 conform to [RFC2396]. Depending on the application, there MAY be one or more endpoints,
2101 whose use is determined by the application.

2102
2103 Following is an example of an HTTP endpoint:

```
2104 <tp:Endpoint tp:uri="http://example.com/servlet/ebxmlhandler"  
2105 tp:type="request"/>
```

2108 The "request" and "response" endpoints MAY-can be dynamically overridden for a particular
2109 request or asynchronous response by application-specified URIs exchanged in *Business*
2110 documents exchanged under the *CPA*.

2111
2112 For a synchronous response, the "response" endpoint is ignored if present. A synchronous
2113 response is always returned on the existing connection, i.e. to the URI that is identified as the
2114 source of the connection.

2116 [7.5.34.28.5.34.2](#) SMTP

2117 SMTP is Simple Mail Transfer Protocol[SMTP]. For use with this standard, Multipurpose
2118 Internet Mail Extensions[MIME] MUST be supported. The MIME media type used by the
2119 SMTP transport layer is "Application" with a sub-type of "octet-stream".

2120
2121 For SMTP, the communication address is the fully qualified mail address of the destination *Party*

2122 as defined by [RFC822]. Following is an example of an SMTP endpoint:

```
2123 <tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"  
2124 tp:type="request"/>
```

2125
2126
2127 SMTP with MIME automatically encodes or decodes the document as needed, on each link in
2128 the path, and presents the decoded document to the destination document-exchange function.

2129
2130 NOTE: The SMTP mail transfer agent encodes binary data (i.e. data that are not 7-bit
2131 ASCII) unless it is aware that the upper level (mail user agent) has already encoded the
2132 data.

2133
2134 NOTE: SMTP by itself (without any authentication or encryption) is subject to denial of
2135 service and masquerading by unknown *Parties*. It is strongly suggested that those *Parties*
2136 who choose SMTP as their transport layer also choose a suitable means of encryption and
2137 authentication either in the document-exchange layer or in the transport layer such as
2138 [S/MIME].

2139
2140 NOTE: SMTP is an asynchronous protocol that does not guarantee a particular quality of
2141 service. A transport-layer acknowledgment (i.e. an SMTP acknowledgment) to the
2142 receipt of a mail *Message* constitutes an assertion on the part of the SMTP server that it
2143 knows how to deliver the mail *Message* and will attempt to do so at some point in the
2144 future. However, the *Message* is not hardened and might never be delivered to the
2145 recipient. Furthermore, the sender will see a transport-layer acknowledgment only from
2146 the nearest node. If the *Message* passes through intermediate nodes, SMTP does not
2147 provide an end-to-end acknowledgment. Therefore receipt of an SMTP
2148 acknowledgement does not guarantee that the *Message* will be delivered to the
2149 application and failure to receive an SMTP acknowledgment is not evidence that the
2150 *Message* was not delivered. It is RECOMMENDED that the reliable-messaging protocol
2151 in the ebXML *Message* Service be used with SMTP.

2152 [7.5.34.38.5.34.3](#) FTP

2153 FTP is File Transfer Protocol[RFC959].

2154
2155 ~~Since a delivery channel specifies receive characteristics, e~~Each *Party* sends a *Message* using
2156 FTP PUT. The endpoint specifies the user id and input directory path (for PUTs to this *Party*).
2157 An example of an FTP endpoint is:

```
2158 <tp:Endpoint uri="ftp://userid@server.foo.com"  
2159 tp:type="request"/>
```

2160
2161
2162 Since FTP needs to be compatible across all implementations, the FTP for ebXML will use the
2163 minimum sets of commands and parameters available for FTP as specified in [RFC959], Section
2164 5.1, and modified in [RFC1123], Section 4.1.2.13. The mode SHALL be stream only and the
2165 type MUST be ASCII Non-print (AN), Image (I) (binary), or Local 8 (L 8) (binary between 8-bit
2166 machines and machines with 36 bit words – for an 8-bit machine Local 8 is the same as Image).

2168
2169 Stream mode closes the data connection upon end of file. The server side FTP MUST set control
2170 to "PASV" before each transfer command to obtain a unique port pair if there are multiple third
2171 party sessions.

2172
2173 NOTE: [RFC 959] states that User-FTP SHOULD send a PORT command to assign a
2174 non-default data port before each transfer command is issued to allow multiple transfers
2175 during a single FTP because of the long delay after a TCP connection is closed until its
2176 socket pair can be reused.

2177
2178 NOTE: The format of the 227 reply to a PASV command is not well standardized and an
2179 FTP client might assume that the parentheses indicated in [RFC959] will be present when
2180 in some cases they are not. If the User-FTP program doesn't scan the reply for the first
2181 digit of host and port numbers, the result will be that the User-FTP might point at the
2182 wrong host. In the response, the h1, h2, h3, h4 is the IP address of the server host and the
2183 p1, p2 is a non-default data transfer port that PASV has assigned.

2184
2185 NOTE: As a recommendation for firewall transparency, [RFC1579] proposes that the
2186 client sends a PASV command, allowing the server to do a passive TCP open on some
2187 random port, and inform the client of the port number. The client can then do an active
2188 open to establish the connection.

2189
2190 NOTE: Since STREAM mode closes the data connection upon end of file, the receiving
2191 FTP might assume abnormal disconnect if a 226 or 250 control code hasn't been received
2192 from the sending machine.

2193
2194 NOTE: [RFC1579] also makes the observation that it might be worthwhile to enhance the
2195 FTP protocol to have the client send a new command APSV (all passive) at startup that
2196 would allow a server that implements this option to always perform a passive open. A
2197 new reply code 151 would be issued in response to all file transfer requests not preceded
2198 by a PORT or PASV command; this *Message* would contain the port number to use for
2199 that transfer. A PORT command could still be sent to a server that had previously
2200 received APSV; that would override the default behavior for the next transfer operation,
2201 thus permitting third-party transfers.

2202
2203

2204 **[7.5.358.5.35](#) DocExchange Element**

2205 The *DocExchange* element provides information that the *Parties* MUST agree on regarding
2206 exchange of documents between them. This information includes the messaging service
2207 properties (e.g. ebXML *Message Service*[ebMS]).

2208
2209 Following is the structure of the *DocExchange* element of the *CPP*. Subsequent sections
2210 describe each child element in greater detail.

2211
2212 `<tp:DocExchange tp:docExchangeId="docExchangeB1">`

```

2213     <tp:ebXMLSenderBinding tp:version="1.1">      <!-- 0 or 1 -->
2214         <tp:ReliableMessaging>                  <!-- 0 or 1 -->
2215         . . .
2216     </tp:ReliableMessaging>
2217     <tp:SenderNonRepudiation>                    <!-- 0 or 1 -->
2218     . . .
2219     </tp:SenderNonRepudiation>
2220     <tp:SenderDigitalEnvelope>                  <!-- 0 or 1 -->
2221     . . .
2222     </tp:SenderDigitalEnvelope>
2223     <tp:NamespaceSupported>                     <!-- 1 or more -->
2224     . . .
2225     </tp:NamespaceSupported>
2226 </tp:ebXMLSenderBinding>
2227 <tp:ebXMLReceiverBinding tp:version="1.1">      <!-- 0 or 1 -->
2228     <tp:ReliableMessaging>                      <!-- 0 or 1 -->
2229     . . .
2230     </tp:ReliableMessaging>
2231     <tp:ReceiverNonRepudiation>                 <!-- 0 or 1 -->
2232     . . .
2233     </tp:ReceiverNonRepudiation>
2234     <tp:ReceiverDigitalEnvelope>               <!-- 0 or 1 -->
2235     . . .
2236     </tp:ReceiverDigitalEnvelope>
2237     <tp:NamespaceSupported>                     <!-- 1 or more -->
2238     . . .
2239     </tp:NamespaceSupported>
2240 </tp:ebXMLReceiverBinding>
2241 </tp:DocExchange>

```

The *DocExchange* element is comprised of ~~an OPTIONAL~~ zero or one *ebXMLSenderBinding* element and ~~an OPTIONAL~~ zero or one *ebXMLReceiverBinding* element.

NOTE: The document-exchange section can be extended to messaging services other than the ebXML *Message* service by adding additional *xxxSenderBinding* and *xxxReceiverBinding* elements and their child elements that describe the other services, where *xxx* is replaced by the name of the additional binding. An example is *XMLP**SenderBinding*/*XMLP**ReceiverBinding*, which might define support for the future XML Protocol specification.

7.5.35.18.5.35.1 docExchangeId attribute

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

7.5.368.5.36 ebXMLSenderBinding element

The *ebXMLSenderBinding* element describes properties related to sending messages with the ebXML *Message* Service[ebMS]. The *ebXMLSenderBinding* element is comprised of the following child elements:

- zero or one *ReliableMessaging* element which specifies the characteristics of reliable

- 2263 messaging,
- 2264 • zero or one *SenderNonRepudiation* element which specifies the sender's
- 2265 requirements and certificate for message signing,
- 2266 • zero or one *SenderDigitalEnvelope* element which specifies the sender's
- 2267 requirements for encryption by the digital-envelope[DIGENV] method,
- 2268 • zero or more *NamespaceSupported* elements that identify any namespace extensions
- 2269 supported by the messaging service implementation.

2270

2271 The *ebXMLSenderBinding* element has one attribute:

- 2272 • a REQUIRED *version* attribute

2273

2274 [7.5.36.18.5.36.1](#) **version attribute**

2275 The REQUIRED *version* attribute identifies the version of the ebXML *Message* Service

2276 specification being used.

2277

2278 [7.5.37.5.37](#) **ReliableMessaging element**

2279 The *ReliableMessaging* element specifies the properties of reliable ebXML *Message* exchange.

2280 The default that applies if the *ReliableMessaging* element is omitted is "BestEffort". The

2281 following is the element structure:

2282

```
2283 <tp:ReliableMessaging>
2284   <tp:Retries>5</tp:Retries>
2285   <tp:RetryInterval>PT2H</tp:RetryInterval>
2286   <tp:PersistDuration>P1D</tp:PersistDuration>
2287   <tp:MessageOrderSemantics>Guaranteed</tp:MessageOrderSemantics>
2288 </tp:ReliableMessaging>
```

2289

2290

2291 The *ReliableMessaging* element is comprised of the following child elements.

- 2292 • ~~an OPTIONAL~~ zero or one *Retries* element,
- 2293 • ~~an OPTIONAL~~ zero or one *RetryInterval* element,
- 2294 • a REQUIRED *PersistDuration* element,
- 2295 • a REQUIRED *MessageOrderSemantics* element.

2296

2297 [7.5.38.5.38](#) **Retries and RetryInterval elements**

2298 The *Retries* and *RetryInterval* elements specify the permitted number of retries and the interval,

2299 expressed as an XML Schema[XMLSCHEMA-2] duration, between retries of sending a reliably

2300 delivered *Message* following a timeout waiting for the *Acknowledgment*. The purpose of the

2301 *RetryInterval* element is to improve the likelihood of success on retry by deferring the retry until

2302 any temporary conditions that caused the error might be corrected. The *RetryInterval* applies to

2303 the time between sending of the original message and the first retry, as well as the time between

2304 all subsequent retries.

2305

2306 The *Retries* and *RetryInterval* elements MUST be included together or MAY be omitted

2307 together. If they are omitted, the values of the corresponding quantities (number of retries and

2308 retry interval) are a local matter at each *Party*.

2309

2310 **7.5.398.5.39 PersistDuration element**

2311 The value of the *PersistDuration* element is the minimum length of time, expressed as an XML
2312 Schema[XMLSCHEMA-2] duration, that data from a *Message* that is sent reliably is kept in
2313 *Persistent Storage* by an ebXML *Message-Service* implementation that receives that *Message* to
2314 facilitate the elimination of duplicates. This duration also applies to response messages that are
2315 kept persistently to allow automatic replies to duplicate messages without their repeated
2316 processing by the application.

2317

2318 **7.5.408.5.40 MessageOrderSemantics element**

2319 The *MessageOrderSemantics* element is an enumeration comprised of the following values:

- 2320 • "Guaranteed"
- 2321 • "NotGuaranteed"

2322

2323 The presence of a *MessageOrder* element in the SOAP Header for ebXML messages determines
2324 if the ordering of messages sent from the *From Party* needs to be preserved so that the *To Party*
2325 receives those messages in the order in which they were sent. If the *MessageOrderSemantics*
2326 element is set to "Guaranteed", then the ebXML message MUST contain a *MessageOrder*
2327 element in the SOAP Header. If the *MessageOrderSemantics* element is set to "NotGuaranteed",
2328 then the ebXML message MUST NOT contain a *MessageOrder* element in the SOAP Header.

2329

2330 **7.5.418.5.41 SenderNonRepudiation element**

2331 The ~~OPTIONAL~~ *SenderNonRepudiation element* conveys the message sender's requirements
2332 and certificate for non-repudiation. Non-repudiation both proves who sent a *Message* and
2333 prevents later repudiation of the contents of the *Message*. Non-repudiation is based on signing
2334 the *Message* using XML Digital Signature[XMLDSIG]. The element structure is as follows:

2335

```
2336 <tp:SenderNonRepudiation>  
2337   <tp:NonRepudiationProtocol>  
2338     http://www.w3.org/2000/09/xmldsig#  
2339   </tp:NonRepudiationProtocol>  
2340   <tp:HashFunction>  
2341     http://www.w3.org/2000/09/xmldsig#sha1  
2342   </tp:HashFunction>  
2343   <tp:SignatureAlgorithm>  
2344     http://www.w3.org/2000/09/xmldsig#dsa-sha1  
2345   </tp:SignatureAlgorithm>  
2346   <tp:SigningCertificateRef tp:certId="CompanyA_SigningCert" />  
2347 </tp:SenderNonRepudiation>
```

2348

2349 If the *SenderNonRepudiation* element is omitted, the *Messages* are not digitally signed.

2350

2351 Security at the document-exchange level applies to all *Messages* in both directions for *Business*
2352 *Transactions* for which security is enabled.

2353
2354
2355
2356
2357
2358
2359

The *SenderNonRepudiation* element is comprised of the following child elements:

- a REQUIRED *NonRepudiationProtocol* element,
- a REQUIRED *HashFunction* (e.g. SHA1, MD5) element,
- a REQUIRED *SignatureAlgorithm* element,
- a REQUIRED *SigningCertificateRef* element

2360 **[7.5.428.5.42](#) NonRepudiationProtocol element**

2361 The REQUIRED *NonRepudiationProtocol* element identifies the technology that will be used to
2362 digitally sign a *Message*. It has a single IMPLIED *version* attribute whose value is a string that
2363 identifies the version of the specified technology.

2364 **[7.5.438.5.43](#) HashFunction element**

2365 The REQUIRED *HashFunction* element identifies the algorithm that is used to compute the
2366 digest of the *Message* being signed.
2367

2368 **[7.5.448.5.44](#) SignatureAlgorithm element**

2369 The REQUIRED *SignatureAlgorithm* element identifies the algorithm that is used to compute
2370 the value of the digital signature.
2371

2372 **[7.5.458.5.45](#) SigningCertificateRef element**

2373 The REQUIRED *SigningCertificateRef* element identifies the certificate the sender uses for
2374 signing messages. Its REQUIRED IDREF attribute, *certId* refers to the *Certificate* element
2375 (under *PartyInfo*) that has the matching ID attribute value.
2376

2377 **[7.5.468.5.46](#) SenderDigitalEnvelope element**

2378 The *SenderDigitalEnvelope* element provides the sender's requirements for message encryption
2379 using the [DIGENV] digital-envelope method. Digital-envelope is a procedure in which the
2380 *Message* is encrypted by symmetric encryption (shared secret key) and the secret key is sent to
2381 the *Message* recipient encrypted with the recipient's public key. The element structure is:

2382
2383
2384
2385
2386
2387
2388
2389
2390
2391

```
<tp:SenderDigitalEnvelope>  
  <tp:DigitalEnvelopeProtocol tp:version="2.0">  
    S/MIME  
  </tp:DigitalEnvelopeProtocol>  
  <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>  
  <tp:EncryptionSecurityDetailsRef  
    tp:securityId="CompanyA_MessageSecurity"/>  
</tp:SenderDigitalEnvelope>
```

2392 Security at the document-exchange level applies to all *Messages* in both directions for *Business*
2393 *Transactions* for which security is enabled.

2394 The *SenderDigitalEnvelope* element contains

- 2396
- 2397
- 2398
- 2399
- a REQUIRED *DigitalEnvelopeProtocol* element,
 - a REQUIRED *EncryptionAlgorithm* element
 - ~~an OPTIONAL~~zero or one *EncryptionSecurityDetailsRef* element.

2400 **7.5.478.5.47 DigitalEnvelopeProtocol element**

2401 The REQUIRED *DigitalEnvelopeProtocol* element identifies the message encryption protocol to
2402 be used. The REQUIRED *version* attribute identifies the version of the protocol.
2403

2404 **7.5.488.5.48 EncryptionAlgorithm element**

2405 The REQUIRED *EncryptionAlgorithm* element identifies the encryption algorithm to be used.
2406

2407 **7.5.498.5.49 EncryptionSecurityDetailsRef element**

2408 The ~~OPTIONAL~~*EncryptionSecurityDetailsRef* element identifies the trust anchors and security
2409 policy that this (sending) *Party* will apply to the other (receiving) *Party*'s encryption certificate.
2410 Its REQUIRED IDREF attribute, *securityId*, refers to the *SecurityDetails* element (under
2411 *PartyInfo*) that has the matching ID attribute value.
2412

2413 **7.5.508.5.50 NamespaceSupported element**

2414 The ~~OPTIONAL~~*NamespaceSupported* element identifies the namespaces supported by the
2415 messaging service implementation and by the business application. Examples are Security
2416 Services Markup Language[S2ML] and Transaction Authority Markup Language[XAML]. For
2417 example, support for the S2ML namespace would be defined as follows:
2418

```
2419 <tp:NamespaceSupported  
2420     tp:location="http://www.s2ml.org/s2ml.xsd"  
2421     tp:version="0.8">http://www.s2ml.org/s2ml  
2422 </tp:NamespaceSupported>  
2423
```

2424 **7.5.518.5.51 ebXMLReceiverBinding element**

2425 The *ebXMLReceiverBinding* element describes properties related to receiving messages with the
2426 ebXML *Message Service*[ebMS]. The *ebXMLReceiverBinding* element is comprised of the
2427 following child elements:

- 2428
- 2429
- 2430
- 2431
- 2432
- 2433
- 2434
- ~~an OPTIONAL~~zero or one *ReliableMessaging* element (see Section [8.5.377.5.37](#)),
 - ~~an OPTIONAL~~zero or one *ReceiverNonRepudiation* element which specifies the receiver's requirements for message signing,
 - zero or one *ReceiverDigitalEnvelope* element which specifies the receiver's requirements and certificate for encryption by the digital-envelope[DIGENV] method,
 - zero or more *NamespaceSupported* elements (see Section [8.5.507.5.50](#)).

2435

2436 The *ebXMLReceiverBinding* element has one attribute:

- 2437 • a REQUIRED *version* attribute (see Section [8.5.36.17-5.36.4](#))

2438 **[7.5.528.5.52](#) ReceiverNonRepudiation element**

2439 The ~~OPTIONAL~~ *ReceiverNonRepudiation element* conveys the message receiver's
2440 requirements for non-repudiation. Non-repudiation both proves who sent a *Message* and prevents
2441 later repudiation of the contents of the *Message*. Non-repudiation is based on signing the
2442 *Message* using XML Digital Signature[XMLDSIG]. The element structure is as follows:

```
2443
2444     <tp:ReceiverNonRepudiation>
2445         <tp:NonRepudiationProtocol>
2446             http://www.w3.org/2000/09/xmldsig#
2447         </tp:NonRepudiationProtocol>
2448         <tp:HashFunction>
2449             http://www.w3.org/2000/09/xmldsig#sha1
2450         </tp:HashFunction>
2451         <tp:SignatureAlgorithm>
2452             http://www.w3.org/2000/09/xmldsig#dsa-sha1
2453         </tp:SignatureAlgorithm>
2454         <tp:SigningSecurityDetailsRef
2455             tp:certId="CompanyA_MessageSecurity"/>
2456     </tp:ReceiverNonRepudiation>
```

2457
2458 If the *ReceiverNonRepudiation* element is omitted, the *Messages* are not digitally signed.

2459

2460 The *ReceiverNonRepudiation* element is comprised of the following child elements:

- 2461 • a REQUIRED *NonRepudiationProtocol* element (see Section [8.5.427.5.42](#)),
- 2462 • a REQUIRED *HashFunction* (e.g. SHA1, MD5) element (see Section [8.5.437.5.43](#)),
- 2463 • a REQUIRED *SignatureAlgorithm* element (see Section [8.5.447.5.44](#)),
- 2464 • ~~an OPTIONAL~~ *zero or one* *SigningSecurityDetailsRef* element

2465

2466 **[7.5.538.5.53](#) SigningSecurityDetailsRef element**

2467 The ~~OPTIONAL~~ *SigningSecurityDetailsRef* element identifies the trust anchors and security
2468 policy that this (receiving) *Party* will apply to the other (sending) *Party*'s signing certificate. Its
2469 REQUIRED IDREF attribute, *securityId*, refers to the *SecurityDetails* element (under
2470 *PartyInfo*) that has the matching ID attribute value.

2471

2472 **[7.5.548.5.54](#) ReceiverDigitalEnvelope element**

2473 The *ReceiverDigitalEnvelope* element provides the receiver's requirements for message
2474 encryption using the [DIGENV] digital-envelope method. Digital-envelope is a procedure in
2475 which the *Message* is encrypted by symmetric encryption (shared secret key) and the secret key
2476 is sent to the *Message* recipient encrypted with the recipient's public key. The element structure
2477 is:

2478

```
2479     <tp:ReceiverDigitalEnvelope>
2480         <tp:DigitalEnvelopeProtocol tp:version="2.0">
2481             S/MIME
2482         </tp:DigitalEnvelopeProtocol>
```

```

2483         <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
2484         <tp:EncryptionCertificateRef
2485             tp:certId="CompanyA_EncryptionCert" />
2486     </tp:ReceiverDigitalEnvelope>

```

2487
2488

2489 The *ReceiverDigitalEnvelope* element contains

- 2490 • a REQUIRED *DigitalEnvelopeProtocol* element (see Section [8.5.477-5.47](#)),
- 2491 • a REQUIRED *EncryptionAlgorithm* element (see Section [8.5.487-5.48](#)),
- 2492 • a REQUIRED *EncryptionCertificateRef* element.

2493

2494 7.5.558.5.55 EncryptionCertificateRef element

2495 The REQUIRED *EncryptionCertificateRef* element identifies the certificate the sender uses for
2496 encrypting messages. Its REQUIRED IDREF attribute, *certId* refers to the *Certificate* element
2497 (under *PartyInfo*) that has the matching ID attribute value.

2498 .

2499 7.5.568.5.56 OverrideMshActionBinding element

2500 The *OverrideMshActionBinding* element can occur zero or more times. It has two REQUIRED
2501 attributes. The *action* attribute identifies the *Message Service Handler* level action whose
2502 delivery is not to use the default *DeliveryChannel* for *Message Service Handler* actions. The
2503 *channelId* attribute specifies the *DeliveryChannel* to be used instead.

2504

2505 7.68.6 SimplePart element

2506 The *SimplePart* element provides a repeatable list of the constituent parts, primarily identified by
2507 the MIME content-type value. The *SimplePart* element has two REQUIRED attributes: *id* and
2508 *mimetype*. The *id* attribute, of type ID, provides the value that will be used later to reference this
2509 *Message* part when specifying how the parts are packaged into composites, if composite
2510 packaging is present. The *mimetype* attribute provides the actual value of content-type for the
2511 simple *Message* part being specified. It also has an IMPLIED *xlink:role* attribute which identifies
2512 some resource that describes the mime part or its purpose. If present, then it SHALL have a
2513 value that is a valid URI in accordance with the [XLINK] specification. The following are
2514 examples of SimplePart elements:

2515

```

2516         <tp:SimplePart tp:id="I001" tp:mimetype="text/xml"/>
2517         <tp:SimplePart tp:id="I002" tp:mimetype="application/xml"/>

```

2518

2519 The *SimplePart* element can have zero or more *NamespaceSupported* elements. Each of these
2520 identifies any namespace supported for the XML packaged in the parent simple body part.

2521

2522 The same *SimplePart* element MAY be referenced from (i.e., reused in) multiple *Packaging*
2523 elements.

2524

2525

2526 **7.78.7 Packaging element**

2527 The subtree of the *Packaging* element provides specific information about how the *Message*
 2528 *Header* and payload constituent(s) are packaged for transmittal over the transport, including the
 2529 crucial information about what document-level security packaging is used and the way in which
 2530 security features have been applied. Typically the subtree under the *Packaging* element indicates
 2531 the specific way in which constituent parts of the *Message* are organized. MIME processing
 2532 capabilities are typically the capabilities or agreements described in this subtree. The *Packaging*
 2533 element provides information about MIME content types, XML namespaces, security
 2534 parameters, and MIME structure of the data that is exchanged between *Parties*.

2536 ~~Following~~ The following is an example of the *Packaging* element which references the example
 2537 *SimplePart* elements given in Section 8.67.6:

```

2539 <!-- Simple ebXML S/MIME Packaging for application-based payload
2540 encryption -->
2541 <tp:Packaging>
2542   <tp:ProcessingCapabilities tp:generate="true" tp:parse="true"/>
2543   <tp:SimplePart tp:id="I001" tp:mimetype="text/xml"/>
2544   <tp:SimplePart tp:id="I002" tp:mimetype="application/xml"/>
2545   <tp:CompositeList>
2546     <tp:Encapsulation
2547       tp:id="I003"
2548       tp:mimetype="application/pkcs7-mime"
2549       tp:mimeparameters="smime-type=&quot;enveloped-data&quot;">
2550       <Constituent tp:idref="I002"/>
2551     </tp:Encapsulation>
2552     <tp:Composite tp:id="I004"
2553       tp:mimetype="multipart/related"
2554       tp:mimeparameters="type=&quot;text/xml&quot;"
2555       version=&quot;1.0&quot;">
2556       <tp:Constituent tp:idref="I001"/>
2557       <tp:Constituent tp:idref="I003"/>
2558     </tp:Composite>
2559   </tp:CompositeList>
2560 </tp:Packaging>

```

2561 See "Matching Packaging" in [Appendix F Appendix F-F](#) for a more specific example.

2564 The *Packaging* element has one attribute; the REQUIRED *id* attribute, with type ID. It is
 2565 referred to in the *ActionBinding*, by using the IDREF attribute, *packageId*.

2567 The child elements of the *Packaging* element are *ProcessingCapabilities*, ~~*SimplePart*~~, and
 2568 *CompositeList*. This set of elements **MAY-can** appear one or more times as a child of each
 2569 *Packaging* element in a *CPP* and SHALL appear once as a child of each *Packaging* element in a
 2570 *CPA*.

2572 **7.7.18.7.1 ProcessingCapabilities element**

2573 The *ProcessingCapabilities* element has two REQUIRED attributes with Boolean values of

2574 either "true" or "false". The attributes are *parse* and *generate*. Normally, these attributes will
2575 both have values of "true" to indicate that the packaging constructs specified in the other child
2576 elements can be both produced as well as processed at the software *Message* service layer.
2577 At least one of the *generate* or *parse* attributes MUST be true.
2578

2579 **7.7.2 SimplePart element**

2580 ~~The SimplePart element provides a repeatable list of the constituent parts, primarily identified by~~
2581 ~~the MIME content type value described earlier for the content model of this element. The same~~
2582 ~~SimplePart element MAY be referenced from (i.e., reused in) multiple Packaging elements.~~
2583

2584 ~~7.7.38.7.2~~ CompositeList element

2585 The final child element of *Packaging* is *CompositeList*, which is a container for the specific way
2586 in which the simple parts are combined into groups (MIME multipart) or encapsulated within
2587 security-related MIME content-types. The *CompositeList* element ~~MAY SHALL~~ be omitted
2588 from *Packaging* when no security encapsulations or composite multipart are used. When the
2589 *CompositeList* element is present, the content model for the *CompositeList* element is a
2590 repeatable sequence of choices of *Composite* or *Encapsulation* elements. The *Composite* and
2591 *Encapsulation* elements ~~MAY-can~~ appear intermixed as desired.

2592 The sequence in which the choices are presented is important because, given the recursive
2593 character of MIME packaging, composites or encapsulations ~~MAY-can~~ include previously
2594 mentioned composites (or rarely, encapsulations) in addition to the *Message* parts characterized
2595 within the *SimplePart* subtree. Therefore, the "top-level" packaging will be described last in the
2596 sequence.

2597
2598 The *Composite* element has the following attributes:

- 2599 • a REQUIRED *mimetype* attribute,
- 2600 • a REQUIRED *id* attribute,
- 2601 • an IMPLIED *mimeparameters* attribute.

2602
2603 The *mimetype* attribute provides the value of the MIME content-type for this *Message* part, and
2604 this will be some MIME composite type, such as "multipart/related" or "multipart/signed". The
2605 *id* attribute, type ID, provides a way to refer to this composite if it needs to be mentioned as a
2606 constituent of some later element in the sequence. The *mimeparameters* attribute provides the
2607 values of any significant MIME parameter (such as "type=application/ xml") that is needed to
2608 understand the processing demands of the content-type.

2609
2610 The *Composite* element has one child element, *Constituent*.

2611
2612 The *Constituent* element has one REQUIRED attribute, *idref*, type IDREF. The *idref* attribute
2613 has as its value the value of the *id* attribute of a previous *Composite*, *Encapsulation*, or
2614 *SimplePart* element. The purpose of this sequence of *Constituents* is to indicate both the
2615 contents and the order of what is packaged within the current *Composite* or *Encapsulation*.
2616

2617 The *Encapsulation* element is typically used to indicate the use of MIME security mechanisms,
Collaboration-Protocol Profile and Agreement Specification

2618 such as [S/MIME] or Open-PGP[RFC2015]. A security body part can encapsulate a MIME part
2619 that has been previously characterized. For convenience, all such security structures are under
2620 the **Encapsulation** element, even when technically speaking the data is not "inside" the body
2621 part. (In other words, the so-called clear-signed or detached signature structures possible with
2622 MIME multipart/signed are for simplicity found under the **Encapsulation** element.)
2623

2624 The **Encapsulation** element has the following attributes:

- 2625 • a REQUIRED **mimetype** attribute,
 - 2626 • a REQUIRED **id** attribute,
 - 2627 • an IMPLIED **mimeparameters** attribute.
- 2628

2629 The **mimetype** attribute provides the value of the MIME content-type for this *Message* part, such
2630 as "application/pkcs7-mime". The **id** attribute, type ID, provides a way to refer to this
2631 encapsulation if it needs to be mentioned as a constituent of some later element in the sequence.
2632 The **mimeparameters** attribute provides the values of any significant MIME parameter(s)
2633 needed to understand the processing demands of the content-type.
2634

2635 Both the **Encapsulation** element and the **Composite** element have child elements consisting of a
2636 **Constituent** element or of a repeatable sequence of **Constituent** elements, respectively.
2637

2638 **7.88.8 ds:Signature element**

2639 **The ds:Signature element (cardinality zero or one) enables the CPA to be** ~~The CPP MAY be~~
2640 digitally signed using technology that conforms with the XML Digital Signature
2641 specification[XMLDSIG]. The **ds:Signature** element is the root of a subtree of elements ~~that~~
2642 ~~MAY be~~ used for signing the *CPP*. The syntax is:

```
2643  
2644 <ds:Signature>...</ds:Signature>  
2645
```

2646 The content of this element and any subelements are defined by the XML Digital Signature
2647 specification. See Section [9.78.7](#) for a detailed discussion. The following additional constraints
2648 on **ds:Signature** are imposed:

- 2649 • A *CPP* MUST be considered invalid if any **ds:Signature** element fails core validation as
2650 defined by the XML Digital Signature specification[XMLDSIG].
2651
 - 2652 • Whenever a *CPP* is signed, each **ds:Reference** element within a **ProcessSpecification**
2653 element MUST pass reference validation and each **ds:Signature** element MUST pass
2654 core validation.
2655
- 2656

2657 NOTE: In case a *CPP* is unsigned, software ~~MAY~~-might nonetheless validate the
2658 **ds:Reference** elements within **ProcessSpecification** elements and report any exceptions.
2659

2660 NOTE: Software for creation of *CPPs* and *CPAs* MAY recognize **ds:Signature** and
2661 automatically insert the element structure necessary to define signing of the *CPP* and *CPA*.
2662 Signature creation itself is a cryptographic process that is outside the scope of this

2663 specification.

2664

2665 NOTE: See non-normative note in Section [8.5.4.57-5.4.5](#) for a discussion of times at which
2666 validity tests MAY be made.

2667

2668 **7.98.9 Comment element**

2669 The *CollaborationProtocolProfile* element ~~MAY~~ contains zero or more *Comment* elements.

2670 The *Comment* element is a textual note that ~~MAY~~ can be added to serve any purpose the author
2671 desires. The language of the *Comment* is identified by a REQUIRED *xml:lang* attribute. The
2672 *xml:lang* attribute MUST comply with the rules for identifying languages specified in [XML]. If
2673 multiple *Comment* elements are present, each ~~MAY~~ can have a different *xml:lang* attribute
2674 value. An example of a *Comment* element follows:

2675

```
2676 <tp:Comment xml:lang="en-US">This is a CPA between A and B</tp:Comment>
```

2677

2678 When a *CPA* is composed from two *CPPs*, all *Comment* elements from both *CPPs* SHALL be
2679 included in the *CPA* unless the two *Parties* agree otherwise.

2680 **89 CPA Definition**

2681 A *Collaboration-Protocol Agreement (CPA)* defines the capabilities that two *Parties* need to
 2682 agree upon to enable them to engage in electronic *Business* for the purposes of the particular
 2683 *CPA*. This section defines and discusses the details of the *CPA*. The discussion is illustrated with
 2684 some XML fragments.

2685
 2686 Most of the XML elements in this section are described in detail in Section [87](#), "CPP Definition".
 2687 In general, this section does not repeat that information. The discussions in this section are
 2688 limited to those elements that are not in the *CPP* or for which additional discussion is needed in
 2689 the *CPA* context. See also [Appendix D](#) ~~Appendix D-D~~ for the XML Schema, and [Appendix B](#)
 2690 ~~Appendix B-B~~ for an example of a *CPA* document.

2691

2692 **8.19.1 CPA Structure**

2693 Following is the overall structure of the *CPA*:

2694

```

2695     <CollaborationProtocolAgreement
2696         xmlns:tp="http://www.oasis-open.org/committees/ebxml-
2697 cppa/schema/cpp-cpa-1_1.xsd"
2698         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2699         xmlns:xlink="http://www.w3.org/1999/xlink"
2700         tp:cpaid="YoursAndMyCPA"
2701         tp:version="1.2">
2702     <tp:Status tp:value="proposed"/>
2703     <tp:Start>1988-04-07T18:39:09</Start>
2704     <tp:End>1990-04-07T18:40:00</End>
2705     <!-- ConversationConstraints MAY appear 0 or 1 times -->
2706     <tp:ConversationConstraints
2707         tp:invocationLimit="100"
2708         tp:concurrentConversations="4"/>
2709     <tp:PartyInfo>
2710         ...
2711     </tp:PartyInfo>
2712     <tp:PartyInfo>
2713         ...
2714     </tp:PartyInfo>
2715     <tp:SimplePart> <!-- one or more -->
2716         ...
2717     </tp:SimplePart>
2718     <tp:Packaging tp:id="N20"> <!-- one or more -->
2719         ...
2720     </tp:Packaging>
2721     <!-- ds:signature MAY appear 0 or more times -->
2722     <ds:Signature>
2723         ...
2724     </ds:Signature>
2725     <tp:Comment xml:lang="en-GB">any text</Comment> <!-- zero or more -
2726     -->

```

2727 </tp:CollaborationProtocolAgreement>
2728

2729 **8.29.2 CollaborationProtocolAgreement element**

2730 The *CollaborationProtocolAgreement* element is the root element of a *CPA*. It has a
2731 REQUIRED *cpaid* attribute that supplies a unique identifier for the document. The value of the
2732 *cpaid* attribute SHALL be assigned by one *Party* and used by both. It is RECOMMENDED that
2733 the value of the *cpaid* attribute be a URI. The value of the *cpaid* attribute SHALL be used as the
2734 value of the *CPAId* element in the ebXML *Message Header*[ebMS] or of a similar element in a
2735 *Message Header* of an alternative messaging service.

2736
2737 NOTE: Each *Party* **MAY-might** associate a local identifier with the *cpaid* attribute.
2738

2739 In addition, the *CollaborationProtocolAgreement* element has an IMPLIED *version* attribute.
2740 This attribute indicates the version of the *CPA*. Its purpose is to provide versioning capabilities
2741 for an instance of a *CPA* as it undergoes negotiation between the two parties. The *version*
2742 attribute SHOULD also be used to provide versioning capability for a *CPA* that has been
2743 deployed and then modified. The value of the *version* attribute SHOULD be a string
2744 representation of a numeric value such as "1.0" or "2.3". The value of the version string
2745 SHOULD be changed with each change made to the *CPA* document both during negotiation and
2746 after it has been deployed.

2747
2748 NOTE: The method of assigning version identifiers is left to the implementation.
2749

2750 The *CollaborationProtocolAgreement* element has REQUIRED [XML] Namespace[XMLNS]
2751 declarations that are defined in Section [87](#), "CPP Definition".
2752

2753 The *CollaborationProtocolAgreement* element is comprised of the following child elements,
2754 most of which are described in greater detail in subsequent sections:

- 2755 • a REQUIRED *Status* element that identifies the state of the process that creates the
2756 *CPA*,
- 2757 • a REQUIRED *Start* element that records the date and time that the *CPA* goes into
2758 effect,
- 2759 • a REQUIRED *End* element that records the date and time after which the *CPA*
2760 MUST be renegotiated by the *Parties*,
- 2761 • zero or one *ConversationConstraints* element that documents certain agreements
2762 about conversation processing,
- 2763 • two REQUIRED *PartyInfo* elements, one for each *Party* to the *CPA*,
- 2764 • one or more *SimplePart* elements,
- 2765 • one or more *Packaging* elements,
- 2766 • one or more *ds:Signature* elements that provide signing of the *CPA* using the XML
2767 Digital Signature[XMLDSIG] standard,
- 2768 • zero or more *Comment* elements.
2769

2770 **8.39.3 Status Element**

2771 The *Status* element records the state of the composition/negotiation process that creates the *CPA*.
2772 An example of the *Status* element follows:

```
2773  
2774 <tp:Status tp:value="proposed"/>
```

2775
2776 The Status element has a REQUIRED *value* attribute that records the current state of
2777 composition of the *CPA*. This attribute is an enumeration comprised of the following possible
2778 values:

- 2779 • "proposed", meaning that the *CPA* is still being negotiated by the *Parties*,
- 2780 • "agreed", meaning that the contents of the *CPA* have been agreed to by both *Parties*,
- 2781 • "signed", meaning that the *CPA* has been "signed" by the *Parties*. This "signing"
2782 ~~MAY~~ takes the form of a digital signature that is described in Section [9.78-7](#) below.

2783
2784 NOTE: The *Status* element MAY be used by a *CPA* composition and negotiation tool to
2785 assist it in the process of building a *CPA*.
2786

2787 **8.49.4 CPA Lifetime**

2788 The lifetime of the *CPA* is given by the *Start* and *End* elements. The syntax is:

```
2789  
2790 <tp:Start>1988-04-07T18:39:09Z</tp:Start>  
2791 <tp:End>1990-04-07T18:40:00Z</tp:End>
```

2792

2793 **8.4.19.4.1 Start element**

2794 The *Start* element specifies the starting date and time of the *CPA*. The *Start* element SHALL be
2795 a string value that conforms to the content model of a canonical dateTime as defined in the XML
2796 Schema Datatypes Specification[XMLSCHEMA-2]. For example, to indicate 1:20 pm UTC
2797 (Coordinated Universal Time) on May 31, 1999, a *Start* element would have the following value:

```
2798  
2799 1999-05-31T13:20:00Z
```

2800

2801 The *Start* element SHALL be represented as Coordinated Universal Time (UTC).
2802

2803 **8.4.29.4.2 End element**

2804 The *End* element specifies the ending date and time of the *CPA*. The *End* element SHALL be a
2805 string value that conforms to the content model of a canonical dateTime as defined in the XML
2806 Schema Datatypes Specification[XMLSCHEMA-2]. For example, to indicate 1:20 pm UTC
2807 (Coordinated Universal Time) on May 31, 1999, an *End* element would have the following
2808 value:

```
2809  
2810 1999-05-31T13:20:00Z
```

2811

2812 The *End* element SHALL be represented as Coordinated Universal Time (UTC).

2813

2814 When the end of the *CPA*'s lifetime is reached, any *Business Transactions* that are still in
2815 progress SHALL be allowed to complete and no new *Business Transactions* SHALL be started.
2816 When all in-progress *Business Transactions* on each conversation are completed, the
2817 *Conversation* SHALL be terminated whether or not it was completed.

2818

2819 NOTE: If a *Business* application defines a conversation as consisting of multiple *Business*
2820 *Transactions*, such a conversation MAY be terminated with no error indication when the
2821 end of the lifetime is reached. The run-time system could provide an error indication to
2822 the application.

2823

2824 NOTE: It ~~MAY~~might not be feasible to wait for outstanding conversations to terminate
2825 before ending the *CPA* since there is no limit on how long a conversation ~~MAY~~can last.

2826

2827 NOTE: The run-time system SHOULD return an error indication to both *Parties* when a
2828 new *Business Transaction* is started under this *CPA* after the date and time specified in
2829 the *End* element.

2830

2831 **8.59.5 ConversationConstraints Element**

2832 The *ConversationConstraints* element places limits on the number of conversations under the
2833 *CPA*. An example of this element follows:

2834

```
2835 <tp:ConversationConstraints tp:invocationLimit="100"  
2836 tp:concurrentConversations="4"/>
```

2837

2838 The *ConversationConstraints* element has the following attributes:

- 2839 • an IMPLIED *invocationLimit* attribute,
- 2840 • an IMPLIED *concurrentConversations* attribute.

2841

2842 **8.5.19.5.1 invocationLimit attribute**

2843 The *invocationLimit* attribute defines the maximum number of conversations that can be
2844 processed under the *CPA*. When this number has been reached, the *CPA* is terminated and
2845 MUST be renegotiated. If no value is specified, there is no upper limit on the number of
2846 conversations and the lifetime of the *CPA* is controlled solely by the *End* element.

2847

2848 NOTE: The *invocationLimit* attribute sets a limit on the number of units of *Business* that
2849 can be performed under the *CPA*. It is a *Business* parameter, not a performance
2850 parameter.

2851

2852 **8.5.29.5.2 concurrentConversations attribute**

2853 The *concurrentConversations* attribute defines the maximum number of conversations that can
2854 be in process under this *CPA* at the same time. If no value is specified, processing of concurrent
2855 conversations is strictly a local matter.

2856
2857 NOTE: The *concurrentConversations* attribute provides a parameter for the *Parties* to use
2858 when it is necessary to limit the number of conversations that can be concurrently processed
2859 under a particular *CPA*. For example, the back-end process might only support a limited
2860 number of concurrent conversations. If a request for a new conversation is received when
2861 the maximum number of conversations allowed under this *CPA* is already in process, an
2862 implementation MAY reject the new conversation or MAY enqueue the request until an
2863 existing conversation ends. If no value is given for *concurrentConversations*, how to handle
2864 a request for a new conversation for which there is no capacity is a local implementation
2865 matter.
2866

2867 **8.69.6 PartyInfo Element**

2868 The general characteristics of the *PartyInfo* element are discussed in Section [8.57.5](#).

2869
2870 The *CPA* SHALL have one *PartyInfo* element for each *Party* to the *CPA*. The *PartyInfo*
2871 element specifies the *Parties'* agreed terms for engaging in the *Business Collaborations* defined
2872 by the *Process-Specification* documents referenced by the *CPA*. If a *CPP* has more than one
2873 *PartyInfo* element, the appropriate *PartyInfo* element SHALL be selected from each *CPP* when
2874 composing a *CPA*.
2875

2876 In the *CPA*, there SHALL be one or more *PartyId* elements under each *PartyInfo* element. The
2877 values of ~~this~~ these elements ~~is~~ are the same as the values of the *PartyId* elements in the ebXML
2878 *Message Service* specification[ebMS] or similar messaging service specification. ~~One~~ These
2879 *PartyId* element s SHALL be used within a *To* or *From Header* element of an ebXML *Message*.
2880

2881 **8.6.19.6.1 ProcessSpecification element**

2882 The *ProcessSpecification* element identifies the *Business Collaboration* that the two *Parties*
2883 have agreed to perform. There ~~MAY~~ can be one or more *ProcessSpecification* elements in a
2884 *CPA*. Each SHALL be a child element of a separate *CollaborationRole* element. See the
2885 discussion in Section [8.5.37.5.3](#).
2886

2887 **8.79.7 SimplePart element**

2888 The *CollaborationProtocolAgreement* element SHALL contain one or more *SimplePart*
2889 elements. See Section [8.67.6](#) for details of the syntax of the *SimplePart* element.

2890 **9.8 Packaging element**

2891 The *CollaborationProtocolAgreement* element SHALL contain one or more *Packaging*
2892 elements. See Section [8.77.7](#) for details of the syntax of the *Packaging* element.

2893 **8.89.9 ds:Signature element**

2894 A *CPA* document ~~MAY~~ can be digitally signed by one or more of the *Parties* as a means of
2895 ensuring its integrity as well as a means of expressing the agreement just as a corporate officer's
2896 signature would do for a paper document. If signatures are being used to digitally sign an

2897 ebXML *CPA* or *CPP* document, then ~~it is strongly RECOMMENDED that~~ [XMLDSIG] SHALL
2898 be used to digitally sign the document.

2899
2900 In a *CPA* involving two *Parties*, there can be up to ~~3~~three **ds:Signature** elements. The *CPA* is
2901 initially signed by one of the two *Parties*. The other *Party* ~~MAY~~MAY~~could~~ then sign over the first
2902 *Party*'s signature. The resulting *CPA* MAY then be signed by a notary.

2903
2904 The **ds:Signature** element is the root of a subtree of elements ~~that MAY be~~ used for signing the
2905 *CPP*. The syntax is:

```
2906 <ds:Signature>...</ds:Signature>
```

2907
2908
2909 The content of this element and any subelements are defined by the XML Digital Signature
2910 specification[XMLDSIG]. The following additional constraints on **ds:Signature** are imposed:

- 2911 • A *CPA* MUST be considered invalid if any **ds:Signature** fails core validation as defined
2912 by the XML Digital Signature specification.
- 2913 • Whenever a *CPA* is signed, each **ds:Reference** within a *ProcessSpecification* MUST
2914 pass reference validation and each **ds:Signature** MUST pass core validation.

2915
2916
2917
2918 NOTE: In case a *CPA* is unsigned, software MAY nonetheless validate the **ds:Reference**
2919 elements within *ProcessSpecification* elements and report any exceptions.

2920
2921 ~~NOTE:~~ Software for creation of *CPPs* and *CPAs* ~~MAY~~MAY~~SHALL~~ recognize **ds:Signature** and
2922 automatically insert the element structure necessary to define signing of the *CPP* and *CPA*.
2923 Signature creation itself is a cryptographic process that is outside the scope of this specification.

2924
2925 NOTE: See non-normative note in Section [8.5.4.57-5.4.5](#) for a discussion of times at which
2926 a *CPA* MAY be validated.

2927 2928 **8.8.19.9.1 Persistent Digital Signature**

2929 If [XMLDSIG] is used to sign an ebXML *CPP* or *CPA*, the process defined in this section of the
2930 specification SHALL be used.

2931 2932 **8.8.1-19.9.1.1 Signature Generation**

2933 Following are the steps to create a digital signature:

- 2934 1. Create a **SignedInfo** element, a child element of **ds:Signature**. **SignedInfo** SHALL have
2935 child elements **SignatureMethod**, **CanonicalizationMethod**, and **Reference** as prescribed by
2936 [XMLDSIG].
- 2937 2. Canonicalize and then calculate the **SignatureValue** over **SignedInfo** based on algorithms
2938 specified in **SignedInfo** as specified in [XMLDSIG].
- 2939 3. Construct the **Signature** element that includes the **SignedInfo**, **KeyInfo**
2940 (RECOMMENDED), and **SignatureValue** elements as specified in [XMLDSIG].
- 2941 4. Include the namespace qualified **Signature** element in the document just signed, following

2942 the last *PartyInfo* element.

2943

2944 **8.8.1.29.9.1.2 ds:SignedInfo element**

2945 The *ds:SignedInfo* element SHALL be comprised of zero or one *ds:CanonicalizationMethod*
2946 element, the *ds:SignatureMethod* element, and one or more *ds:Reference* elements.

2947

2948 **8.8.1.39.9.1.3 ds:CanonicalizationMethod element**

2949 The *ds:CanonicalizationMethod* element is defined as OPTIONAL in [XMLDSIG], meaning
2950 that the element need not appear in an instance of a *ds:SignedInfo* element. The default
2951 canonicalization method that is applied to the data to be signed is [XMLC14N] in the absence of
2952 a *ds:CanonicalizationMethod* element that specifies otherwise. This default SHALL also serve
2953 as the default canonicalization method for the ebXML *CPP* and *CPA* documents.

2954

2955 **8.8.1.49.9.1.4 ds:SignatureMethod element**

2956 The *ds:SignatureMethod* element SHALL be present and SHALL have an *Algorithm* attribute.
2957 The RECOMMENDED value for the *Algorithm* attribute is:

2958

2959 `http://www.w3.org/2000/09/xmldsig#sha1`

2960

2961 This RECOMMENDED value SHALL be supported by all compliant ebXML *CPP* or *CPA*
2962 software implementations.

2963

2964 **8.8.1.59.9.1.5 ds:Reference element**

2965 The *ds:Reference* element for the *CPP* or *CPA* document SHALL have a REQUIRED URI
2966 attribute value of "" to provide for the signature to be applied to the document that contains the
2967 *ds:Signature* element (the *CPA* or *CPP* document). The *ds:Reference* element for the *CPP* or
2968 *CPA* document **MAY-can** include an IMPLIED *type* attribute that has a value of:

2969

2970 `"http://www.w3.org/2000/09/xmldsig#Object"`

2971

2972 in accordance with [XMLDSIG]. This attribute is purely informative. It MAY be omitted.
2973 Implementations of software designed to author or process an ebXML *CPA* or *CPP* document
2974 SHALL be prepared to handle either case. The *ds:Reference* element **MAY-can** include the *id*
2975 attribute, type ID, by which this *ds:Reference* element **MAY-be is** referenced from a
2976 *ds:Signature* element.

2977

2978 **8.8.1.69.9.1.6 ds:Transform element**

2979 The *ds:Reference* element for the *CPA* or *CPP* document SHALL include a descendant
2980 *ds:Transform* element that excludes the containing *ds:Signature* element and all its descendants.
2981 This exclusion is achieved by means of specifying the *ds:Algorithm* attribute of the *Transform*
2982 element as

2983 `"http://www.w3.org/2000/09/xmldsig#enveloped-signature"`.

2984

2985 For example:

2986 `<ds:Reference ds:URI="">`

2987 `<ds:Transforms>`

```

2988         <ds:Transform
2989         ds:Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
2990         </ds:Transforms>
2991         <ds:DigestMethod
2992         ds:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
2993         <ds:DigestValue>...</ds:DigestValue>
2994     </ds:Reference>

```

2995

8.8.1.79.9.1.7 ds:Algorithm element

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

```

2997     http://www.w3.org/2000/09/xmldsig#enveloped-signature
2998
2999

```

3000

NOTE: When digitally signing a *CPA*, it is RECOMMENDED that each *Party* sign the document in accordance with the process described above.

3001

3002

~~When the two Parties sign the CPA, the~~ ~~The~~ first *Party* that signs the *CPA* ~~will~~ SHALL sign only the *CPA* contents, excluding their own signature. The second *Party* ~~signs~~ SHALL sign over the contents of the *CPA* as well as the *ds:Signature* element that contains the first *Party's* signature. ~~It MAY be necessary that a notary~~ If necessary, a notary can then sign over both signatures.

3007

3008

3009

8.99.10 Comment element

The *CollaborationProtocolAgreement* element ~~MAY~~ contains zero or more *Comment* elements.

See Section [8.97.9](#) for details of the syntax of the *Comment* element.

3011

3012

3013

8.109.11 Composing a CPA from Two CPPs

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

3015

3016

3017

8.10.19.11.1 ID Attribute Duplication

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

3022

3023

3024

3025

3026

3027

NOTE: A party can seek to prevent ID/IDREF reassignment in the *CPA* by choosing ID and IDREF values which are likely to be unique among its trading partners. For example, the following *Certificate* element found in a *CPP* has a *certId* attribute that is generic enough that it might clash with a *certId* attribute found in a collaborating party's *CPP*:

3028

3029

3030

```

<tp:Certificate
tp:certId="EncryptionCert"><ds:KeyInfo/></tp:Certificate>

```

3031 To prevent reassignment of this ID (and its associated IDREFs) in a CPA, a better choice
3032 of *certId* in Company A's CPP would be:

3033

```
3034 <tp:Certificate  
3035 tp:certId="CompanyA_EncryptionCert"><ds:KeyInfo/></tp:Certificate>
```

3036

3037 **8.119.12 Modifying Parameters of the Process-Specification Document Based on** 3038 **Information in the CPA**

3039 A *Process-Specification* document contains a number of parameters, expressed as XML
3040 attributes. An example is the security attributes that are counterparts of the attributes of the *CPA*
3041 ***BusinessProcessCharacteristics*** element. The values of these attributes can be considered to be
3042 default values or recommendations. When a *CPA* is created, the *Parties* **MAY-might** decide to
3043 accept the recommendations in the *Process-Specification* or they MAY agree on values of these
3044 parameters that better reflect their needs.

3045

3046 When a *CPA* is used to configure a run-time system, choices specified in the *CPA* MUST always
3047 assume precedence over choices specified in the referenced *Process-Specification* document. In
3048 particular, all choices expressed in a *CPA*'s ***BusinessProcessCharacteristics*** and ***Packaging***
3049 elements MUST be implemented as agreed to by the *Parties*. These choices SHALL override
3050 the default values expressed in the *Process-Specification* document. The process of installing the
3051 information from the *CPA* and *Process-Specification* document MUST verify that all of the
3052 resulting choices are mutually consistent and MUST signal an error if they are not.

3053

3054 NOTE: There are several ways of overriding the information in the *Process-*
3055 *Specification* document by information from the *CPA*. For example:

3056

- 3057 • The CPA composition tool can create a separate copy of the *Process-Specification*
3058 document. The tool can then directly modify the *Process-Specification* document
3059 with information from the *CPA*. One advantage of this method is that the override
3060 process is performed entirely by the *CPA* composition tool. A second advantage is
3061 that with a separate copy of the *Process-Specification* document associated with the
3062 particular *CPA*, there is no exposure to modifications of the *Process-Specification*
3063 document between the time that the *CPA* is created and the time it is installed in the
3064 *Parties'* systems.
- 3065 • A *CPA* installation tool can dynamically override parameters in the *Process-*
3066 *Specification* document using information from the corresponding parameters in the
3067 *CPA* at the time the *CPA* and *Process-Specification* document are installed in the
3068 *Parties'* systems. This eliminates the need to create a separate copy of the *Process-*
3069 *Specification* document.
- 3070 • Other possible methods might be based on XSLT transformations of the parameter
3071 information in the *CPA* and/or the *Process-Specification* document.

3072 910 References

3073 Some references listed below specify functions for which specific XML definitions are provided
3074 in the *CPP* and *CPA*. Other specifications are referred to in this specification in the sense that
3075 they are represented by keywords for which the *Parties* to the *CPA* MAY obtain plug-ins or
3076 write custom support software but do not require specific XML element sets in the *CPP* and
3077 *CPA*.

3078
3079 In a few cases, the only available specification for a function is a proprietary specification.
3080 These are indicated by notes within the citations below.

3081
3082 [ccOVER] ebXML Core Components and Business Process Document Overview,
3083 <http://www.ebxml.org>

3084
3085 [DIGENV] Digital Envelope, RSA Laboratories, <http://www.rsasecurity.com/rsalabs/>. NOTE:
3086 At this time, the only available specification for digital envelope appears to be the RSA
3087 Laboratories specification.

3088
3089 [ebBPSS] ebXML Business Process Specification Schema, <http://www.ebxml.org>.

3090
3091
3092 [ebMS] ebXML Message Service Specification, <http://www.ebxml.org>.

3093
3094 [ebRS] ebXML Registry Services Specification, <http://www.ebxml.org>.

3095
3096 ~~[ebTA] ebXML Technical Architecture Specification, <http://www.ebxml.org>.~~

3097
3098 [HTTP] Hypertext Transfer Protocol, Internet Engineering Task Force RFC2616.

3099
3100 [IPSEC] IP Security Document Roadmap, Internet Engineering Task Force RFC 2411.

3101
3102 [ISO6523] Structure for the Identification of Organizations and Organization Parts, International
3103 Standards Organization ISO-6523.

3104
3105 [MIME] MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying
3106 and Describing the Format of Internet *Message* Bodies. Internet Engineering Task Force RFC
3107 1521.

3108
3109 [RFC822] Standard for the Format of ARPA Internet Text Messages, Internet Engineering Task
3110 Force RFC 822.

3111
3112 [RFC959] File Transfer Protocol (FTP), Internet Engineering Task Force RFC 959.

3113
3114 [RFC1123] Requirements for Internet Hosts -- Application and Support, R. Braden, Internet
3115 Engineering Task Force, October 1989.

- 3116
3117 [RFC1579] Firewall-Friendly FTP, S. Bellovin, Internet Engineering Task Force, February 1994.
3118
3119 [RFC2015] MIME Security with Pretty Good Privacy, M. Elkins, Internet Engineering Task
3120 Force, RFC 2015.
3121
3122 [RFC2119] Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering
3123 Task Force RFC 2119.
3124
3125 [RFC2251] Lightweight Directory Access Protocol (v3); Mark Wahl, Tim Howes, Steve Kille.
3126
3127 [RFC2396] Uniform Resource Identifiers (URI): Generic Syntax; T. Berners-Lee, R. Fielding, L.
3128 Masinter - August 1998.
3129
3130 [S/MIME] S/MIME Version 3 Message Specification, Internet Engineering Task Force RFC
3131 2633.
3132
3133 [S2ML] Security Services Markup Language, <http://s2ml.org/>.
3134
3135 [SMTP] Simple Mail Transfer Protocol, Internet Engineering Task Force RFC 821.
3136
3137 [SSL] Secure Sockets Layer, Netscape Communications Corp. <http://developer.netscape.com>.
3138 NOTE: At this time, it appears that the Netscape specification is the only available specification
3139 of SSL. Work is in progress in IETF on "Transport Layer Security", which is intended as a
3140 replacement for SSL.
3141
3142 [X12] ANSI X12 Standard for Electronic Data Interchange, X12 Standard Release
3143 4050, December 2001
3144
3145 [XAML] Transaction Authority Markup Language, <http://xaml.org/>.
3146
3147 [XLINK] XML Linking Language, <http://www.w3.org/TR/xlink/>.
3148
3149 [XML] Extensible Markup Language (XML), World Wide Web Consortium,
3150 <http://www.w3.org>.
3151
3152 [XMLC14N] Canonical XML, Ver. 1.0, <http://www.w3.org/TR/XML-C14N/>.
3153
3154 [XMLDSIG] XML Signature Syntax and Processing, Worldwide Web Consortium,
3155 <http://www.w3.org/TR/xmlsig-core/>.
3156
3157 [XMLNS] Namespaces in XML, T. Bray, D. Hollander, and A. Layman, Jan. 1999,
3158 <http://www.w3.org/TR/REC-xml-names/>.
3159
3160 [XMLSCHEMA-1] XML Schema Part 1: Structures, <http://www.w3.org/TR/xmlschema-1/>.
3161

- 3162 [XMLSCHEMA-2] XML Schema Part 2: Datatypes,
3163 <http://www.w3.org/TR/xmlschema-2/>.
3164
3165 [XPOINTER] XML Pointer Language, ver. 1.0, <http://www.w3.org/TR/xptr>.

3166 ~~1011~~ Conformance

3167 In order to conform to this specification, an implementation:

- 3168 a) SHALL support all the functional and interface requirements defined in this specification,
3169 b) SHALL NOT specify any requirements that would contradict or cause non-conformance
3170 to this specification.

3171

3172 A conforming implementation SHALL satisfy the conformance requirements of the applicable
3173 parts of this specification.

3174

3175 An implementation of a tool or service that creates or maintains ebXML *CPP* or *CPA* instance
3176 documents SHALL be determined to be conformant by validation of the *CPP* or *CPA* instance
3177 documents, created or modified by said tool or service, against the XML
3178 Schema[XMLSCHEMA-1] definition of the *CPP* or *CPA* in Appendix D and available from

3179

3180 http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-1_1.xsd

3181

3182 by using two or more validating XML Schema parsers that conform to the W3C XML Schema
3183 specifications[XMLSCHEMA-1,XMLSCHEMA-2].

3184

3185 The objective of conformance testing is to determine whether an implementation being tested
3186 conforms to the requirements stated in this specification. Conformance testing enables vendors to
3187 implement compatible and interoperable systems. Implementations and applications SHALL be
3188 tested using available test suites to verify their conformance to this specification.

3189

3190 Publicly available test suites from vendor neutral organizations such as OASIS and the U.S.A.
3191 National Institute of Science and Technology (NIST) SHOULD be used to verify the
3192 conformance of implementations, applications, and components claiming conformance to this
3193 specification. Open-source reference implementations ~~MAY~~might be available to allow vendors
3194 to test their products for interface compatibility, conformance, and interoperability.

3195

3196 ~~1112~~ 1112 Disclaimer

3197 The views and specification expressed in this document are those of the authors and are not
3198 necessarily those of their employers. The authors and their employers specifically disclaim
3199 responsibility for any problems arising from correct or incorrect implementation or use of this
3200 design.

3201 [1213](#) Contact Information

3202 Martin W. Sachs (~~Team Leader~~)
3203 IBM T. J. Watson Research Center
3204 P.O.B. 704
3205 Yorktown Hts, NY 10598
3206 USA
3207 Phone: 914-784-7287
3208 email: mwsachs@us.ibm.com
3209

3210 Chris Ferris
3211 XML Technology Development
3212 Sun Microsystems, Inc
3213 One Network Drive
3214 Burlington, Ma 01824-0903
3215 USA
3216 Phone: 781-442-3063
3217 email: chris.ferris@east.sun.com
3218

3219 Dale W. Moberg ([Chair](#))
3220 Cyclone Commerce
3221 17767 North Perimeter Dr., Suite 103
3222 Scottsdale, AZ 85255
3223 USA
3224 Phone: 480-627-1800
3225 email: dmoberg@columbus.rr.com
3226

3227 Tony Weida ([Coordinating Editor](#))
3228 ~~Edifees~~
3229 ~~2310 130th Ave. NE, Suite 100~~
3230 ~~Bellevue, WA 98005~~
3231 [535 West 110th St., #4J](#)
3232 [New York, NY 10025](#)
3233 USA
3234 Phone: 212-678-5265
3235 email: TonyW@edifees.comrweida@hotmail.com

3236 **Notices**

3237
3238 OASIS takes no position regarding the validity or scope of any intellectual property or other
3239 rights that might be claimed to pertain to the implementation or use of the technology described
3240 in this document or the extent to which any license under such rights might or might not be
3241 available; neither does it represent that it has made any effort to identify any such rights.
3242 Information on OASIS's procedures with respect to rights in OASIS specifications can be found
3243 at the OASIS website. Copies of claims of rights made available for publication and any
3244 assurances of licenses to be made available, or the result of an attempt made to obtain a general
3245 license or permission for the use of such proprietary rights by implementors or users of this
3246 specification, can be obtained from the OASIS Executive Director.

3247
3248 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
3249 applications, or other proprietary rights which may cover technology that may be required to
3250 implement this specification. Please address the information to the OASIS Executive Director.

3251
3252 Copyright (C) The Organization for the Advancement of Structured Information Standards
3253 [OASIS] 2001. All Rights Reserved.

3254
3255 This document and translations of it may be copied and furnished to others, and derivative works
3256 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
3257 published and distributed, in whole or in part, without restriction of any kind, provided that the
3258 above copyright notice and this paragraph are included on all such copies and derivative works.
3259 However, this document itself may not be modified in any way, such as by removing the
3260 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
3261 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
3262 Property Rights document must be followed, or as required to translate it into languages other
3263 than English.

3264
3265 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
3266 successors or assigns.

3267
3268 This document and the information contained herein is provided on an "AS IS" basis and OASIS
3269 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT
3270 LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN
3271 WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
3272 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

3273
3274

3275 **Copyright Statement**

3276
3277 Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved.

3278
3279 This document and translations of it MAY be copied and furnished to others, and derivative
3280 works that comment on or otherwise explain it or assist in its implementation MAY be prepared,
3281 copied, published and distributed, in whole or in part, without restriction of any kind, provided
3282 that the above copyright notice and this paragraph are included on all such copies and derivative
3283 works. However, this document itself MAY not be modified in any way, such as by removing
3284 the copyright notice or references to ebXML, UN/CEFACT, or OASIS, except as required to
3285 translate it into languages other than English.

3286
3287 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
3288 successors or assigns.

3289
3290 This document and the information contained herein is provided on an "AS IS" basis and
3291 ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
3292 NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN
3293 WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
3294 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

3295

3296 **Appendix A** Example of CPP Document (Non-Normative)

3297 This example includes two CPPs that are used to form the CPA in Appendix B. They are
3298 available as ASCII files at

3299 http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-example-companyA-1_1.xml

3300 http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-example-companyB-1_1.xml

3301

3302 See draft-cpp-example-companyA-012.xml and draft-cpp-example-companyB-012.xml in zip
3303 package for now.

3304

3305

3306 **Appendix B** Example of CPA Document (Non-Normative)

3307 The example in this appendix is to be parsed with an XML Schema parser. The schema is
3308 available as an ASCII file at

3309 http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-1_1.xsd

3310

3311 The example that can be parsed with the XSD is available at:

3312 http://www.oasis-open.org/committees/ebxml-cppa/schema/cpa-example-1_1.xml

3313

3314 See draft-cpa-example-012.xml in the zip package for now.

3315

3316

3317

3318 **Appendix C Business Process Specification Corresponding**
 3319 **to Complete CPP/CPA Definition (Non-Normative)**

3320 This Business Process Specification referenced by the CPPs and CPA in Appendix A and
 3321 Appendix B are reproduced here. This document is available as an ASCII file at:

3322 http://www.oasis-open.org/committees/ebxml-cppa/schema/bpss-example-1_1.xml
 3323

```

3324 <?xml version="1.0" encoding="UTF-8"?>
3325 <ProcessSpecification xmlns="http://www.ebxml.org/BusinessProcess"
3326 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3327 xsi:schemaLocation="http://www.ebxml.org/BusinessProcess ebBPSS.xsd"
3328 name="PIP3A4RequestPurchaseOrder" uuid="bpid:RosettaNet:3A4$2.0</"
3329 version="R02.00">
3330   <Documentation>This PIP enables a buyer to issue a purchase order and
3331   obtain a quick response from the provider that acknowledges which of the
3332   purchase order product line items are accepted, rejected, or
3333   pending</Documentation>
3334   <!--Purchase order Request Document-->
3335   <BusinessDocument name="Purchase Order Request"
3336   nameID="Pip3A4PurchaseOrderRequest" specificationLocation="%SYSTEM
3337   /XMLPIPVALIDATION/3A4/PurchaseOrderRequest.xsd">
3338     <Documentation>The document is an XSD file that specifies the
3339     rules for creating the XML document for the business action of requesting a
3340     purchase order</Documentation>
3341   </BusinessDocument>
3342   <BusinessDocument name="Purchase Order Confirmation"
3343   nameID="Pip3A4PurchaseOrderConfirmation" specificationLocation="%SYSTEM
3344   /XMLPIPVALIDATION/3A4/PurchaseOrderConfirmation.xsd">
3345     <Documentation>The document is an XSD file that specifies the
3346     rules for creating the XML document for the business action of making a
3347     purchase order confirmation</Documentation>
3348   </BusinessDocument>
3349   <BusinessTransaction name="Request Purchase Order"
3350   nameID="RequestPurchaseOrder_BT">
3351     <RequestingBusinessActivity name="Purchase Order Request Action"
3352     nameID="PurchaseOrderRequestAction" isAuthorizationRequired="true"
3353     isIntelligibleCheckRequired="true" isNonRepudiationReceiptRequired="true"
3354     isNonRepudiationRequired="true" timeToAcknowledgeReceipt="P0Y0M0DT2H0M0S">
3355       <DocumentEnvelope businessDocument="Purchase Order Request"
3356       businessDocumentIDRef="Pip3A4PurchaseOrderRequest" isAuthenticated="true"
3357       isConfidential="true" isTamperProof="true"/>
3358     </RequestingBusinessActivity>
3359     <RespondingBusinessActivity name="Purchase Order Confirmation
3360     Action" nameID="PurchaseOrderConfirmationAction"
3361     isAuthorizationRequired="true" isIntelligibleCheckRequired="true"
3362     isNonRepudiationReceiptRequired="false" isNonRepudiationRequired="true"
3363     timeToAcknowledgeReceipt="P0Y0M0DT2H0M0S">
3364       <DocumentEnvelope businessDocument="Purchase Order
3365       Confirmation" businessDocumentIDRef="Pip3A4PurchaseOrderConfirmation"
3366       isAuthenticated="true" isConfidential="true" isPositiveResponse="true"
3367       isTamperProof="true"/>
3368     </RespondingBusinessActivity>
3369   </BusinessTransaction>

```

```
3370         <BinaryCollaboration name="Request Purchase Order"
3371 nameID="RequestPurchaseOrder_BC">
3372             <InitiatingRole name="Buyer" nameID="Buyer"/>
3373             <RespondingRole name="Seller" nameID="Seller"/>
3374             <BusinessTransactionActivity name="Request Purchase Order"
3375 nameID="RequestPurchaseOrder_BTA" businessTransaction="Request Purchase
3376 Order" businessTransactionIDRef="RequestPurchaseOrder_BT"
3377 fromAuthorizedRole="Buyer" fromAuthorizedRoleIDRef="Buyer"
3378 toAuthorizedRole="Seller" toAuthorizedRoleIDRef="Seller"
3379 isLegallyBinding="true" timeToPerform="P0Y0M0DT24H0M0S" isConcurrent="false"/>
3380         </BinaryCollaboration>
3381 </ProcessSpecification>
3382
```


3383 **Appendix D** W3C XML Schema Document Corresponding to
3384 Complete CPP and CPA Definition (Normative)

3385 This XML Schema document is available as an ASCII file at:

3386 http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-1_1.xsd

3387

3388 To provide for extensibility, many of the *CPP* and *CPA* schema elements allow OPTIONAL
3389 namespace qualified wildcard elements. These are defined in the form

3390

```
3391 <any namespace="not ##targetNamespace not ##ds"  
3392     processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
```

3393

3394 in the elements CollaborationProtocolProfile and CollaborationProtocolAgreement, and in the
3395 form

3396

```
3397 <any namespace="##other"  
3398     processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
```

3399

3400 in the elements PartyInfo, SimplePart, Packaging, PartyRef, CollaborationRole,
3401 ProcessSpecification, Certificate, SecurityDetails, DeliveryChannel, Transport, DocExchange,
3402 OverrideMshActionBinding, and ActionContext. (The second form cannot be used on
3403 CollaborationProtocolProfile and CollaborationProtocolAgreement because their content models
3404 would otherwise be ambiguous, due to the ds:Signature element which can occur 0 to 3 times.)

3405

3406 See draft-cpp-cpa-012.xsd in the zip package for now.

3407

3408

3409 **Appendix E** Formats of Information in the CPP and CPA
3410 (Normative)

3411 This section defines format information that is not defined by the [XML] specification and is not
3412 defined in the descriptions of specific elements.

3413

3414 Formats of Character Strings

3415

3416 **Protocol and Version Elements**

3417

3418 Values of *Protocol*, *Version*, and similar elements are flexible. In general, any protocol and
3419 version for which the support software is available to both *Parties* to a *CPA* MAY be selected as
3420 long as the choice does not require changes to the ~~DTD~~-~~or~~-schema and therefore a change to this
3421 specification.

3422

3423 NOTE: A possible implementation MAY be based on the use of plug-ins or exits to
3424 support the values of these elements.

3425

3426 **Alphanumeric Strings**

3427

3428 Alphanumeric strings not further defined in this section follow these rules unless otherwise
3429 stated in the description of an individual element:

3430

- 3431 • Values of elements are case insensitive unless otherwise stated.
- 3432 • Strings which represent file or directory names are case sensitive to ensure that they are
3433 acceptable to both UNIX and Windows systems.

3434

3435 **Numeric Strings**

3436

3437 A numeric string is a signed or unsigned decimal integer in the range imposed by a 32-bit binary
3438 number, i.e. -2,147,483,648 to +2,417,483,647. Negative numbers MAY or MAY not be
3439 permitted in particular elements.

3440 **Appendix F** Composing a CPA from Two CPPs (Non- 3441 Normative)

3442 3443 Overview and Limitations

3444
3445 In this appendix, we discuss the tasks involved in *CPA* formation from *CPPs*. The detailed
3446 procedures for *CPA* formation are currently left for implementers. Therefore, no normative
3447 specification is provided for algorithms for *CPA* formation. In this initial section, we provide
3448 some background on *CPA* formation tasks.

3449
3450 There are three basic reasons why we prefer to provide information about the component tasks
3451 involved in *CPA* formation rather than attempt to provide an algorithm for *CPA* formation:

- 3452
3453 1. The precise informational inputs to the *CPA* formation procedure vary.
- 3454 2. There exist at least two distinct approaches to *CPA* formation. One useful approach for
3455 certain situations involves basing *CPA* formation from a *CPA* template; the other approach
3456 involves composition from *CPPs*.
- 3457 3. The conditions for output of a given *CPA* given two *CPPs* can involve different levels and
3458 extents of interoperability. In other words, when an optimal solution that satisfies every level
3459 of requirement and every other additional constraint does not exist, a *Party* **MAY-might**
3460 propose a *CPA* that satisfies enough of the requirements for “a good enough”
3461 implementation. User input **MAY-can** be solicited to determine what is a good enough
3462 implementation, and so **MAY-might** be as varied as there are user configuration options to
3463 express preferences. In practice, compromises **MAY-could** be made on security, reliable
3464 messaging, levels of signals and acknowledgements, and other matters in order to find some
3465 acceptable means of doing *Business*.

3466
3467 Each of these reasons is elaborated in greater detail in the following sections.

3468 3469 Variability in Inputs

3470
3471 User preferences provide one source of variability in the inputs to the *CPA* formation process.
3472 Let us suppose in this section that each of the *Parties* has made its *CPP* available to potential
3473 collaborators. Normally one *Party* will have a desired *Business Collaboration* (defined in a
3474 *Process-Specification* document) to implement with its intended collaborator. So the information
3475 inputs will normally involve a user preference about intended *Business Collaboration* in addition
3476 to just the *CPPs*.

3477
3478 A *CPA* formation tool **MAY-might** have access to local user information not advertised in the
3479 *CPP* that **MAY-could** contribute to the *CPA* that is formed. A user **MAY-might** have chosen to
3480 only advertise those system capabilities that reflect nondeprecated capabilities. For example, a
3481 user **MAY-might** only advertise HTTP and omit FTP, even when capable of using FTP. The
3482 reason for omitting FTP might be concerns about the scalability of managing user accounts,

3483 directories, and passwords for FTP sessions. Despite not advertising an FTP capability,
3484 configuration software MAY-might use tacit knowledge about its own FTP capability to form a
3485 *CPA* with an intended collaborator who happens to have only an FTP capability for
3486 implementing a desired *Business Collaboration*. In other words, *Business* interests MAYcould,
3487 in this case, override the deprecation policy. Both tacit knowledge and detailed preference
3488 information account for variability in inputs into the *CPA* formation process.

3489

3490 Different Approaches

3491

3492 When a *CPA* is formed from a *CPA* template, it is typically because the capabilities of one of the
3493 *Parties* are limited, and already tacitly known. For example, if a *CPA* template were implicitly
3494 presented to a Web browser for use in an implementation using browser based forms capabilities,
3495 then the template maker can assume that the other *Party* has suitable web capabilities (or is about
3496 to download them). Therefore, all that really needs to be done is to supply *PartyRef*, *Certificate*,
3497 and similar items for substitution into a *CPA* template. The *CPA* template will already have all
3498 the capabilities of both *Parties* specified at the various levels, and will have placeholders for
3499 values to be supplied by one of the *Partners*. A simple form might be adequate to gather the
3500 needed information and produce a *CPA*.

3501

3502 Variable Output "Satisficing" Policies

3503

3504 A *CPA* can support a fully interoperable configuration in which agreement has been reached on
3505 all technical levels needed for *Business Collaboration*. In such a case, matches in capabilities
3506 will have been found in all relevant technical levels.

3507

3508 However, there can be interoperable configurations agreed to in a *CPA* in which not all aspects
3509 of a *Business Collaboration* match. Gaps MAY-could exist in packaging, security, signaling,
3510 reliable messaging and other areas and yet the systems can still transport the *Business* data, and
3511 special means can be employed to handle the exceptions. In such situations, a *CPA* MAY-might
3512 reflect configured policies or expressly solicited user permission to ignore some shortcomings in
3513 configurations. A system might not be capable of responding in a *Business Collaboration* so as
3514 to support a specified ability to supply non-repudiation of receipt, but might still be acceptable
3515 for *Business* reasons. A system might not be able to handle all the processing needed to support,
3516 for example, SOAP with Attachments and yet still be able to treat the multipart according to
3517 "multipart/mixed" handling and allow *Business Collaboration* to take place. In fact, short of a
3518 failure to be able to transport data and a failure to be able to provide data relevant to the *Business*
3519 *Collaboration*, there are few features that might not be temporarily or indefinitely compromised
3520 about, given overriding *Business* interests. This situation of "partial interoperability" is to be
3521 expected to persist for some time, and so interferes with formulating a "clean" algorithm for
3522 deciding on what is sufficient for interoperability.

3523

3524 In summary, the previous considerations indicate that at the present it is at best premature to seek
3525 a simple algorithm for *CPA* formation from *CPPs*. It is to be expected that as capability
3526 characterization and exchange becomes a more refined subject, that advances will be made in
3527 characterizing *CPA* formation and negotiation.

3528

3529 Despite it being too soon to propose a simple algorithm for *CPA* formation that covers all the
3530 above variations, it is currently possible to enumerate the basic tasks involved in matching
3531 capabilities within *CPPs*. This information might assist the software implementer in designing a
3532 partially automated and partially interactive software system useful for configuring *Business*
3533 *Collaboration* so as to arrive at satisfactorily complete levels of interoperability. To understand
3534 the context for characterizing the constituent tasks, the general perspective on *CPPs* and *CPAs*
3535 needs to be briefly recalled.

3536

3537 CPA Formation Component Tasks

3538

3539 Technically viewed, a *CPA* provides "bindings" between *Business-Collaboration* specifications
3540 (as defined in the *Process-Specification* document) and those services and protocols that are used
3541 to implement these specifications. The implementation takes place at several levels and involves
3542 varied services at these levels. A *CPA* that arrives at a fully interoperable binding of a *Business*
3543 *Collaboration* to its implementing services and protocols can be thought of as arriving at
3544 interoperable, application-to-application integration. *CPAs* **MAY-might** fall short of this goal and
3545 still be useful and acceptable to the collaborating *Parties*. Certainly, if no matching data-
3546 transport capabilities can be discovered, a *CPA* would not provide much in the way of
3547 interoperable *Business-to-Business* integration. Likewise, partial *CPAs* will leave significant
3548 system work to be done before a completely satisfactory application-to-application integration is
3549 realized. Even so, partial integration **MAY-could** be sufficient to allow collaboration, and to
3550 enjoy payoffs from increased levels of automation.

3551

3552 In practice, the *CPA* formation process **MAY-might** produce a complete *CPA*, a failure result, a
3553 gap list that drives a dialog with the user, or perhaps even a *CPA* that implements partial
3554 interoperability "good enough" for the *Business* collaborators. Because both matching
3555 capabilities and interoperability can be matters of degree, the constituent tasks are finding the
3556 matches in capabilities at different levels and for different services. We next proceed to
3557 characterize many of these constituent tasks.

3558

3559

3560 CPA Formation from *CPPs*: Enumeration of Tasks

3561

3562 To simplify discussion, assume in the following that we are viewing the tasks faced by a
3563 software agent when:

- 3564 1. an intended collaborator is known and the collaborator's *CPP* has been retrieved,
- 3565 2. the *Business Collaboration* between us and our intended collaborator has been selected,
- 3566 3. the specific role that our software agent is to play in the *Business Collaboration* is
3567 known, and
- 3568 4. the capabilities that are to be advertised in our *CPP* are known.

3569

3570 For vividness, we will suppose that our example agent wishes to play the role of supplier and
3571 seeks to find one of its current customers to begin a Purchase Order *Business Collaboration* in
3572 which the intended player plays a complementary role. For simplicity, we assume that the
3573 information about capabilities is restricted to what is available in our agent's *CPP* and in the

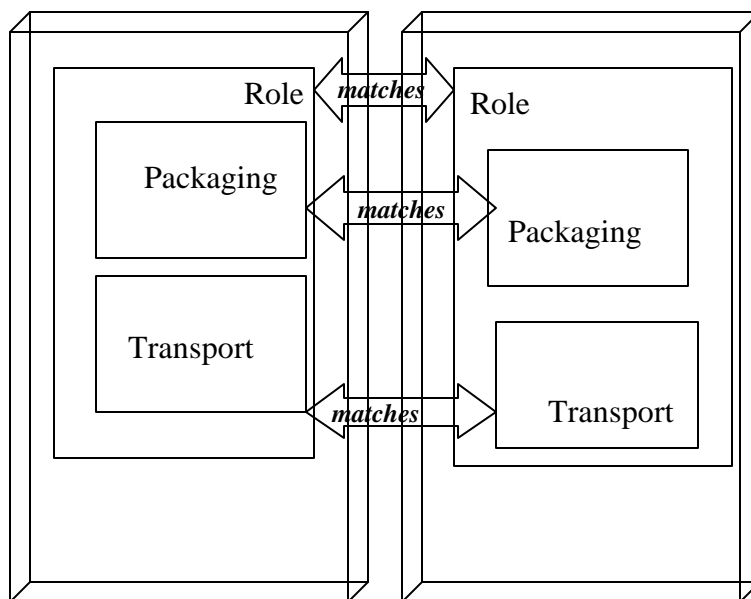
3574 *CPP* of its intended collaborator.

3575
3576 In general, the constituent tasks consist of finding "matches" between our capabilities and our
3577 intended collaborator's at the various levels of the protocol stacks and with respect to the
3578 services supplied at these various levels.

3579
3580 Figure 6 illustrates the basic tasks informing a *CPA* from two *CPPs*: matching roles, matching
3581 packaging, and matching transport.

3582

Figure 6: Basic Tasks in Forming a CPA



3583
3584 The first task to be considered is certainly the most basic: finding that our intended collaborator
3585 and ourselves have complementary role capabilities.

3586
3587

3588 Matching Roles

3589
3590 Our agent has its role already selected in the *Business Collaboration*. So it now begins to check
3591 the **Role** elements in its collaborator's *CPP*. The first element to examine is the **PartyInfo**
3592 element that contains a subtree of elements called **CollaborationRole**. This set is searched to
3593 discover a role that complements the role of our agent within the *Business Collaboration* that we
3594 have chosen. For simple binary collaboration cases, it is typically sufficient to find that our
3595 intended collaborator's **CollaborationRole** set contains **ProcessSpecification** elements that we

3596 intend to implement and where the role is not identical to our role. For more general
 3597 collaborations, we would need to know the list of roles available within the process, and keep
 3598 track that for each of the collaborators, the roles chosen instantiate those that have been specified
 3599 within the *Process-Specification* document. Collaborations involving more than two roles are not
 3600 discussed further.

3601
 3602

3603 Matching Transport

3604

3605 We now have available a list of candidate *CollaborationRole* elements with the desired
 3606 *ProcessSpecification* element (Purchase Ordering) and where our intended collaborator plays the
 3607 buyer role. For simplicity, let us suppose just one *CollaborationRole* element meets these
 3608 conditions within each of the relevant *CPPs* and not discuss iterating over lists. (Within these
 3609 remarks, where repetition is possible, we will frame the discussion by assuming that just one
 3610 element is present.)

3611

3612 Matching transport first means matching the *SendingProtocol* capabilities of our intended
 3613 collaborator with the *ReceivingProtocol* capabilities found on our side. Perusal of the *CPP DTD*
 3614 ~~⊕~~ Schema will reveal that the *ServiceBinding* element provides the doorway to the relevant
 3615 information from each side's *CollaborationRole* element with the *channelId* attribute. This
 3616 *channelId* attribute's value allows us to find *DeliveryChannels* within each *CPP*. The
 3617 *DeliveryChannel* has a *transportId* attribute that allows us to find the relevant *Transport*
 3618 subtrees.

3619

3620 For example, suppose that our intended buyer has a *Transport* entry:

3621

```
3622 <tp:Transport tp:transportId="buyerid001">
3623   <tp:SendingProtocol>HTTP</tp:SendingProtocol>
3624   <tp:ReceivingProtocol>HTTP</tp:ReceivingProtocol>
3625   <tp:Endpoint tp:uri="https://www.buyername.com/po-response"
3626     tp:type="allPurpose"/>
3627   <tp:TransportSecurity>
3628     <tp:Protocol tp:version="1.0">TLS</tp:Protocol>
3629     <tp:CertificateRef
3630 tp:certId=certid001">BuyerName</tp:CertificateRef>
3631   </tp:TransportSecurity>
3632 </tp:Transport>
```

3633

3634 and our seller has a *Transport* entry:

3635

```
3636 <tp:Transport tp:transportId="sellid001">
3637   <tp:SendingProtocol>HTTP</tp:SendingProtocol>
3638   <tp:ReceivingProtocol>HTTP</tp:ReceivingProtocol>
3639   <tp:Endpoint tp:uri="https://www.sellername.com/pos_here"
3640     tp:type="allPurpose"/>
3641   <tp:TransportSecurity>
3642     <tp:Protocol tp:version="1.0">TLS</Protocol>
3643     <tp:CertificateRef
3644 tp:certId="certid002">Sellername</tp:CertificateRef>
```

3645 </tp:TransportSecurity>
3646 </tp:Transport>

3647

3648 A transport match for requests involves finding the initiator role or buyer has a *SendingProtocol*
3649 that matches one of our *ReceivingProtocols*. So here, "HTTP" provides a match. A transport
3650 match for responses involves finding the responder role or seller has a *SendingProtocol* that
3651 matches one of the buyer's *ReceivingProtocols*. So in the above example, "HTTP" again
3652 provides a match. When such matches exist, we then have discovered an interoperable solution at
3653 the transport level. If not, no *CPA* will be available, and a high-priority gap has been identified
3654 that will need to be remedied by whatever exception handling procedures are in place.

3655

3656

3657 Matching Transport Security

3658

3659 Matches in transport security, such as in the above, will reflect agreement in versions and values
3660 of protocols. Software can supply some knowledge here so that if one side has SSL-3 and the
3661 other TLS-1, it can guess that security is available by means of a fallback of TLS to SSL.

3662

3663

3664 Matching Document Packaging

3665

3666 Probably one of the most complex matching problems arises when it comes to finding whether
3667 there are matches in document-packaging capabilities. Here both security and other MIME
3668 handling capabilities can combine to create complexity for appraising whether full
3669 interoperability can be attained.

3670

3671 Access to the information needed for undertaking this task is found under the *ServiceBinding*
3672 elements, and again we suppose that each side has just one *ServiceBinding* element. However,
3673 we will initially suppose that two *Packaging* elements are available to consider under each role.
3674 Several quite different ways of thinking about the matching task are available, and several
3675 methods for the tasks ~~MAY~~-might be performed when assessing whether a good enough match
3676 exists.

3677

3678 To continue our previous purchase-ordering example, we recall that the packaging is the
3679 particular combination of body parts, XML instances (*Headers* and payloads), and security
3680 encapsulations used in assembling the *Message* from its data sources. Both requests and
3681 responses will have packaging. The most complete specification of packaging, which ~~MAY~~
3682 might not always be needed, would consist of:

3683

- 3684 1. The buyer asserting what packaging it can generate for its purchase order, and what
3685 packaging it can parse for its purchase order response *Messages*.
- 3686 2. The seller asserting what packaging it can generate for its purchase order responses and
3687 what packaging it can parse for received purchase orders.

3688

3689 Matching by structural comparison would then involve comparing the packaging details of the
3690 purchase orders generated by the seller with the purchase orders parsable by the buyer. The

3691 comparison would seek to establish that the MIME types of the *SimplePart* elements of
 3692 corresponding subtrees match and would then proceed to check that the *CompositeList* matched
 3693 in MIME types and in sequence of composition.

3694

3695 For example, if each *CPP* contained the packaging subtrees below, and under the appropriate
 3696 *ServiceBindings*, then there would be a straightforward match by structural comparison:

3697

```

3698     <tp:Packaging tp:id="I1001">
3699         <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>
3700         <tp:SimplePart tp:id="P1" tp:mimetype="text/xml"/>
3701         <tp:NamespaceSupported
3702 tp:location="http://schemas.xmlsoap.org/soap/envelope/"
3703 tp:version="1.1">http://schemas.xmlsoap.org/soap/envelope</tp:NamespaceSupported>
3704     </tp:Packaging>
3705     <tp:Packaging id="I2001">
3706         <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>
3707         <tp:SimplePart tp:id="P11" tp:mimetype="text/xml"/>
3708         <tp:SimplePart tp:id="P12" tp:mimetype="application/xml"/>
3709         <tp:CompositeList>
3710             <tp:Composite tp:mimetype="multipart/related" tp:id="P13">
3711                 <tp:Constituent tp:idref="P11"/>
3712                 <tp:Constituent tp:idref="P12"/>
3713             </tp:Composite>
3714         </tp:CompositeList>
3715     </tp:Packaging>
  
```

3732

3733 However, it is to be expected that over time it will become possible only to assert what
 3734 packaging is *generated* within each *ServiceBinding* for the requester and responder roles. This
 3735 simplification assumes that each side has knowledge of what MIME types it handles correctly,
 3736 what encapsulations it handles correctly, and what composition modes it handles correctly. By
 3737 scanning the packaging specifications against its lists of internal capabilities, it can then look up
 3738 whether other side's generated packaging scheme is one it can process and accept it under those
 3739 conditions. Knowing what generated packaging style was produced by the other side could
 3740 enable the software agent to propose a packaging scheme using only the MIME types and
 3741 packaging styles used in the incoming *Message*. Such a packaging scheme would be likely to be
 3742 acceptable to the other side when included within a proposed *CPA*. Over time, and as proposal

3743 and negotiation conventions get established, it is to be expected that the methods used for
3744 determining a match in packaging capabilities will move away from structural comparison to
3745 simpler methods, using more economical representations. For example, parsing capabilities
3746 might eventually be captured by using a compact description of the accepting grammar for the
3747 packaging and content labelling schemes that can be parsed and for which semantic handlers are
3748 available.

3749

3750 Matching Document-Level Security

3751

3752 Although the matching task for document-level security is a subtask of the Packaging-matching
3753 task, it is useful to discuss some specifics tied to the three major document-level security
3754 approaches found in [S/MIME], OpenPGP[RFC2015], and XMLDsig[XMLDSIG].

3755

3756 XMLDsig matching capability can be inferred from document-matching capabilities when the
3757 use of ebXML *Message Service*[ebMS] packaging is present. However, there are other sources
3758 that **SHOULD** be checked to confirm this match. A *SimplePart* element can have a
3759 *NameSpaceSupported* element. XMLDsig capability **SHOULD** be found there. Likewise, a
3760 detailed check on this match **SHOULD** examine the information under the *NonRepudiation*
3761 element and similar elements under the ebXMLBinding element to check for compatibility in
3762 hash functions and algorithms.

3763

3764 The existence of several radically different approaches to document-level security, together with
3765 the fact that it is unusual at present for a given *Party* to commit to more than one form of such
3766 security, means that there can be basic failures to match security frameworks. Therefore, there
3767 might be no match in capabilities that supports full interoperability at all levels. For the moment,
3768 we assume that document-level security matches will require both sides able to handle the same
3769 security composites (multipart/signed using S/MIME, for example.)

3770

3771 However, suppose that there are matches at the transport and transport layer security levels, but
3772 that the two sides have failures at the document-security layer because one side makes use of
3773 PGP signatures while the other uses S/MIME. Does this mean that no *CPA* can be proposed?
3774 That is not necessarily the case.

3775

3776 Both S/MIME and OpenPGP permit signatures to be packaged within "multipart/signed"
3777 composites. In such a case, it **MAY-might** be possible to extract the data and arrive at a partial
3778 implementation that falls short with respect to non-repudiation. While neither side could check
3779 the other's signatures, it might still be possible to have confidential document transmission and
3780 transport-level authentication for the *Business* data. Eventually *CPA*-formation software **MAY**
3781 **might** be created that is able to identify these exceptional situations and "salvage" a proposed
3782 *CPA* with downgraded security features. Whether the other side would accept such a proposed
3783 *CPA* would, naturally, involve what their preferences are with respect to initiating a *Business*
3784 *Collaboration* and sacrificing some security features. *CPA*-formation software **MAY-might**
3785 eventually be capable of these adaptations, but it is to be expected that human assistance will be
3786 needed for such situations in the near term.

3787

3788 Of course, an implementation **MAY-could** simply decide to terminate looking for a *CPA* when a
Collaboration-Protocol Profile and Agreement Specification

3789 match fails in any crucial factor for an interoperable implementation. At the very least, the users
3790 SHOULD be warned that the only CPAs that can be proposed will be missing security or other
3791 normally desirable features or features specified by the *Business Collaboration*.
3792

3793 Other Considerations

3794 Though preferences among multiple capabilities are indicated by the document order in which
3795 they are listed, it is possible that ties might occur. At present, these ties are left to be resolved by
3796 a negotiation process not discussed here.
3797

3798 **Appendix G** Correspondence Between CPA and ebXML
 3799 Messaging Parameters (Normative)

3800 The following table shows the correspondence between elements used in the ebXML Messaging
 3801 Service message header and their counterparts in the CPA.
 3802
 3803

Message Header Element / Attribute	Corresponding CPA Element / Attribute
PartyId element	PartyId element
Role element	Role element
CPAId element	cpaid attribute in CollaborationProtocolAgreement element
ConversationId element	No equivalent; SHOULD be generated by software above the Message Service Interface (MSI)
Service element	Service element
Action element	action attribute in ThisPartyActionBinding element (same as action attribute in OtherPartyActionBinding element)
TimeToLive	Computed as the sum of Timestamp (in message header) + PersistDuration (under DocExchange/ebXMLReceiverBinding/ReliableMessaging)
MessageId	No equivalent; generated by the MSH per message
Timestamp	No equivalent; generated by the MSH per message
RefToMessageId	No equivalent; usually passed in by the application where applicable
SyncReply	syncReplyMode attribute in MessagingCharacteristics element; the SyncReply element is included if and only if the syncReplyMode attribute is not "none"
DuplicateElimination element	duplicateElimination attribute in MessagingCharacteristics element; the DuplicateElimination element is included if the duplicateElimination attribute under MessagingCharacteristics is set to "always", or if it is set to "perMessage" and the application indicates to the MSH that duplicate elimination is desired
Manifest element	Packaging element; each Reference element under Manifest SHOULD correspond to a SimplePart that is referenced from one of the CompositeList elements under Packaging
xlink:role attribute in Reference element	xlink:role attribute in SimplePart element
AckRequested element	ackRequested attribute in MessagingCharacteristics element; an AckRequested element is included in the SOAP Header if the ackRequested attribute is set to "always"; if it is set to "perMessage", input passed to the MSI is to be used to determine if an AckRequested element needs to be

	included; likewise, the signed attribute under AckRequested will be appropriately set based on the ackSignatureRequested attribute and possibly determined by input passed to the MSI
MessageOrder element	messageOrderSemantics attribute in ReliableMessaging element; the MessageOrder element will be present if the AckRequested element is present, and if the messageOrderSemantics attribute in the ReliableMessaging element is set to "Guaranteed"
ds:Signature element	ds:Signature will be present in the SOAP Header if the nonRepudiationOfOrigin attribute in the BusinessProcessCharacteristics element is set to "true"; the relevant parameters for constructing the signature can be obtained from the SenderNonRepudiation and ReceiverNonRepudiation elements

3804
3805
3806
3807
3808

The following table shows the implicit parameters employed by the ebXML Messaging Service that are not **explicitly** included in the message header and how those parameters can be obtained from the CPA.

Implicit Messaging Parameters	Corresponding CPA Element / Attribute
Retries (not in Message Header) but used to govern Reliable Messaging behavior in sender	Retries element (under ReliableMessaging element)
RetryInterval (not in Message Header) but used to govern Reliable Messaging behavior in sender	RetryInterval element (under ReliableMessaging element)
PersistDuration (not in Message Header) but used to govern Reliable Messaging behavior in receiver	PersistDuration element (under ReliableMessaging element)
Endpoint (not in Message Header) but used for sending SOAP message	Endpoint element (under TransportReceiver); the type of message being sent MUST be passed in to the MSI; an appropriate endpoint can then be selected from among the Endpoints included under the TransportReceiver element
Use Service & Action to determine the corresponding DeliveryChannel	DeliveryChannel
Use ReceiverDigitalEnvelope to determine the encryption algorithm and key	ReceiverDigitalEnvelope
Use SenderNonRepudiation to determine signing certificate(s) and ReceiverNonRepudiation to determine the trust anchors and security policy to apply to the signing certificate	SenderNonRepudiation and ReceiverNonRepudiation

Use Packaging to determine how payload containers ought to be encapsulated. Also use Packaging to determine how an individual SimplePart ought to be extracted and validated against its schema	Packaging
Use TransportClientSecurity and TransportServerSecurity to determine certificates to be used by server and client for authentication purposes	TransportClientSecurity and TransportServerSecurity
Use the DeliveryChannel identified by defaultMshChannelId for standalone MSH level messages like Acknowledgment, Error, StatusRequest, StatusResponse, Ping, Pong, unless overridden by OverrideMshActionBinding	defaultMshChannelId attribute in PartyInfo element, and OverrideMshActionBinding

3809 **Appendix H** Glossary of Terms

3810

Term	Definition
AGREEMENT	An arrangement between two partner types that specifies in advance the conditions under which they will trade (terms of shipment, terms of payment, collaboration protocols, etc.) An agreement does not imply specific economic commitments.
APPLICATION	Software that may implements a Service by processing one or more of the Messages in the Document Exchanges associated with the Service.
AUTHORISATION	A right or a permission that is granted to a system entity to access a system resource.
BUSINESS ACTIVITY	A business activity is used to represent the state of the business process of one of the partners. For instance the requester is either in the state of sending the request, in the state of waiting for the response, or in the state of receiving.
BUSINESS COLLABORATION	An activity conducted between two or more parties for the purpose of achieving a specified outcome.
BUSINESS DOCUMENT	The set of information components that are interchanged as part of a business activity.
BUSINESS PARTNER	An entity that engages in business transactions with another business partner(s).
BUSINESS PROCESS	The means by which one or more activities are accomplished in operating business practices.
BUSINESS PROCESS SPECIFICATION SCHEMA	Defines the necessary set of elements to specify run-time aspects and configuration parameters to drive the partners' systems used in the collaboration. The goal of the BP Specification Schema is to provide the bridge between the eBusiness process modeling and specification of eBusiness software components.

BUSINESS TRANSACTION	A business transaction is a logical unit of business conducted by two or more parties that generates a computable success or failure state. The community, the partners, and the process, are all in a definable, and self-reliant state prior to the business transaction, and in a new definable, and self-reliant state after the business transaction. In other words if you are still 'waiting' for your business partner's response or reaction, the business transaction has not completed.
CLIENT	TBD
COLLABORATION	Two or more parties working together under a defined set of rules.
COLLABORATION PROTOCOL	The protocol that defines for a Collaborative Process: 1. The sequence, dependencies and semantics of the Documents that are exchanged between Parties in order to carry out that Collaborative Process, and 2. The Messaging Capabilities used when sending documents between those Parties. Note that a Collaborative Process may-can have more than one Collaboration Protocol by which it may can be implemented.
COLLABORATION PROTOCOL AGREEMENT (CPA)	Information agreed between two (or more) Parties that identifies or describes the specific Collaboration Protocol that they have agreed to use. A CPA indicates what the involved Parties “will” do when carrying out a Collaborative Process. A CPA must-be is representable by a Document.
COLLABORATION PROTOCOL PROFILE (CPP)	Information about a Party that can be used to describe one or more Collaborative Processes and associated Collaborative Protocols that the Party supports. A CPP indicates what a Party “can” do in order to carry out a Collaborative Process. A CPP must-be is representable by a Document. While logically, a CPP is a single document, in practice, the CPP may-might be a set of linked documents that express various aspects of the capabilities. A CPP is not an agreement. It represents the capabilities of a Party.
COLLABORATIVE PROCESS	A shared process by which two Parties work together in order to carry out a process. The Collaborative Process may-can be defined by an ebXML Collaboration Model.

CONFORMANCE	Fulfillment of a product, process or service of all requirements specified; adherence of an implementation to the requirements of one or more specific standards or technical specifications.
DIGITAL SIGNATURE	A digital code that can be attached to an electronically transmitted message that uniquely identifies the sender
DOCUMENT	A Document is any data that can be represented in a digital form.
DOCUMENT EXCHANGE	An exchange of documents between two parties.
ENCRYPTION	Cryptographic transformation of data (called "plaintext") into a form (called "ciphertext") that conceals the data's original meaning to prevent it from being known or used. If the transformation is reversible, the corresponding reversal process is called "decryption", which is a transformation that restores encrypted state.data to its original state.
EXTENSIBLE MARKUP LANGUAGE	XML is designed to enable the exchange of information (data) between different applications and data sources on the World Wide Web and has been standardized by the W3C.
IMPLEMENTATION	An implementation is the realization of a specification. It can be a software product, system or program.
MESSAGE	The movement of a document from one party to another.
MESSAGE HEADER	A specification of the structure and composition of the information necessary for an ebXML Messaging Service to successfully generate or process an ebXML compliant message.
MESSAGING CAPABILITIES	The set of capabilities that support exchange of Documents between Parties. Examples are the communication protocol and its parameters, security definitions, and general properties of ending and receiving messages.

MESSAGING SERVICE	A framework that enables interoperable, secure and reliable exchange of Messages between Trading Partners.
PACKAGE	A general-purpose mechanism for organizing elements into groups. Packages may can be nested within other packages.
PARTY	A Party is an entity such as a company, department, organisation or individual that can generate, send, receive or relay Documents.
PARTY DISCOVERY PROCESS	A Collaborative Process by which one Party can discover CPP information about other Parties.
PAYLOAD	A section of data/information that is not part of the ebXML wrapping.
PAYLOAD CONTAINER	An optional container used to envelope the real payload of an ebXML message. If a payload is present, the payload container must consists of a MIME header portion (the ebXML Payload Envelope) and a content portion (the payload itself).
PAYLOAD ENVELOPE	The specific MIME headers that are associated with a MIME part.
<u>RECEIVER</u>	<u>TBD</u>
REGISTRY	A mechanism whereby relevant repository items and metadata about them can be registered such that a pointer to their location, and all their metadata, can be retrieved as a result of a query.
<u>REQUESTER</u>	<u>TBD</u>
<u>RESPONDER</u>	<u>TBD</u>
ROLE	The named specific behavior of an entity participating in a particular context. A role may could be static (e.g., an association end) or dynamic (e.g., a collaboration role).
SECURITY POLICY	A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources.
<u>SENDER</u>	<u>TBD</u>

<u>SERVER</u>	<u>TBD</u>
UNIQUE IDENTIFIER	The abstract concept of utilizing a standard mechanism and process for assigning a sequence of alphanumeric codes to ebXML Registry items, including: Core Components, Aggregate Information Entities, and Business Processes
UNIVERSALLY UNIQUE IDENTIFIER (UUID)	An identifier that is unique across both space and time, with respect to the space of all UUIDs. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network.

3811

3812