Collaboration-Protocol Profile and Agreement Specification
Version 1.0

ebXML Trading-Partners Team

10 May 2001

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.ebxml.org/specs/ebCCP.pdf

*Latest version***:**

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

33   2   ebXML Participants

34   The authors wish to recognize the following for their significant participation to the development
35   of this document.
36
37
38   David Burdett, CommerceOne
39   Tim Chiou, United World Chinese Commercial Bank
40   Chris Ferris, Sun
41   Scott Hinkelman, IBM
42   Maryann Hondo, IBM
43   Sam Hunting, ECOM XML
44   John Ibbotson, IBM
45   Kenji Itoh, JASTPRO
46   Ravi Kacker, eXcelon Corp.
47   Thomas Limanek, iPlanet
48   Daniel Ling, VCHEQ
49   Henry Lowe, OMG
50   Dale Moberg, Cyclone Commerce
51   Duane Nickull, XMLGlobal Technologies
52   Stefano Pogliani, Sun
53   Rebecca Reed, Mercator
54   Karsten Riemer, Sun
55   Marty Sachs, IBM
56   Yukinori Saito, ECOM
57   Tony Weida, Edifecs
58

59 # 3 Table of Contents

146   4   Introduction

147

148   **4.1 Summary of Contents of Document**

149

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

157

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

165

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

168

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

172

173   This specification is organized as follows:
174   • Section 5 defines the objectives of this specification.
175   • Section 6 provides a system overview.
176   • Section 7 contains the definition of the *CPP*, identifying the structure and all
177   necessary fields.
178   • Section 8 contains the definition of the *CPA*.
179   • The appendices include examples of XML *CPP* and *CPA* documents (non-
180   normative), the DTD (normative), an XML Schema document equivalent to the DTD
181   (normative), formats of information in the *CPP* and *CPA* (normative)*,* and composing
182   a *CPA* from two *CPPs* (non-normative).

183   **4.2 Document Conventions**

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

186

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

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

189

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

192

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

196

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

206

4.3 Use of XML Schema

208   The schema of the *CPP* and *CPA* is based on the Candidate-Recommendation version of the
209   XML Schema specification[XMLSCHEMA-1,XMLSCHEMA-2]. When XML Schema
210   advances to Recommendation status, some changes will be needed in this specification and its
211   schema. The changes are indicated by XML comments in the current schema document in
212   Appendix D

213

4.4 Version of the Specification

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

218

4.5 Definitions

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

221

4.6 Audience

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

227

4.7 Assumptions

229   It is expected that the reader has an understanding of [XML] and is familiar with the concepts of

230   electronic *Business* (eBusiness).
231

232   **4.8 Related Documents**

233   Related documents include ebXML Specifications on the following topics:
234   • ebXML Technical Architecture Specification[ebTA]
235   • ebXML *Message* Service Specification[ebMS]
236   • ebXML Business Process Specification Schema[ebBPSS]
237   • ebXML Glossary [ebGLOSS]
238   • ebXML Core Component and Business Document Overview[ccOVER]
239   • ebXML Registry Services Specification[ebRS]
240
241   See Section 9 for the complete list of references.
242

243 5 Design Objectives

244 The objective of this specification is to ensure interoperability between two *Parties* even though
245 they MAY procure application software and run-time support software from different vendors.
246 The *CPP* defines a *Party's Message*-exchange capabilities and the *Business Collaborations* that
247 it supports. The *CPA* defines the way two *Parties* will interact in performing the chosen *Business*
248 *Collaboration*. Both *Parties* SHALL use identical copies of the *CPA* to configure their run-time
249 systems. This assures that they are compatibly configured to exchange *Messages* whether or not
250 they have obtained their run-time systems from the same vendor. The configuration process
251 MAY be automated by means of a suitable tool that reads the *CPA* and performs the
252 configuration process.
253
254 In addition to supporting direct interaction between two *Parties,* this specification MAY also be
255 used to support interaction between two *Parties* through an intermediary such as a portal or
256 broker. In this initial version of this specification, this MAY be accomplished by creating a *CPA*
257 between each *Party* and the intermediary in addition to the *CPA* between the two *Parties*. The
258 functionality needed for the interaction between a *Party* and the intermediary is described in the
259 *CPA* between the *Party* and the intermediary. The functionality needed for the interaction
260 between the two *Parties* is described in the *CPA* between the two *Parties.*
261
262 It is an objective of this specification that a *CPA* SHALL be capable of being composed by
263 intersecting the respective *CPPs* of the *Parties* involved. The resulting *CPA* SHALL contain
264 only those elements that are in common, or compatible, between the two *Parties*. Variable
265 quantities, such as number of retries of errors, are then negotiated between the two *Parties*. The
266 design of the *CPP* and *CPA* schemata facilitates this composition/negotiation process. However,
267 the composition and negotiation processes themselves are outside the scope of this specification.
268 Appendix F contains a non-normative discussion of this subject.
269
270 It is a further objective of this specification to facilitate migration of both traditional EDI-based
271 applications and other legacy applications to platforms based on the ebXML specifications. In
272 particular, the *CPP* and *CPA* are components of the migration of applications based on the X12
273 838 Trading-Partner Profile to more automated means of setting up *Business* relationships and
274 doing *Business* under them.

6 System Overview

6.1 What This Specification Does

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

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

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

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

The document that defines the interactions between two *Parties* is a *Process-Specification* document that MAY conform to the ebXML Business Process Specification Schema[ebBPSS]. The *CPP* and *CPA* include references to this *Process-Specification* document. The *Process-Specification* document MAY be stored in a repository such as the ebXML Registry. See NOTE about alternative *Business-Collaboration* descriptions in section 7.5.4.

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

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

The *CPP* describes the capabilities of an individual *Party*. A *CPA* describes the capabilites that two *Parties* have agreed to use to perform a particular *Business Collaboration*. These *CPAs* define the "information technology terms and conditions" that enable *Business* documents to be electronically interchanged between *Parties*. The information content of a *CPA* is similar to the information-technology specifications sometimes included in Electronic Data Interchange (EDI) *Trading Partner Agreements (TPAs).* However, these *CPAs* are not paper documents. Rather, they are electronic documents that can be processed by computers at the *Parties'* sites in order to

319 set up and then execute the desired *Business* information exchanges. The "legal" terms and
320 conditions of a *Business* agreement are outside the scope of this specification and therefore are
321 not included in the *CPP* and *CPA*.

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

Repository

CPP(A)

<PartyInfo PartyId="N01">
 <ProcessSpecification xlink:href="http://

Process Specification(A1)

Business Collaboration

<PartyInfo PartyId="N02">
 <ProcessSpecification xlink:href="http://

Process Specification(A2)

Business Collaboration
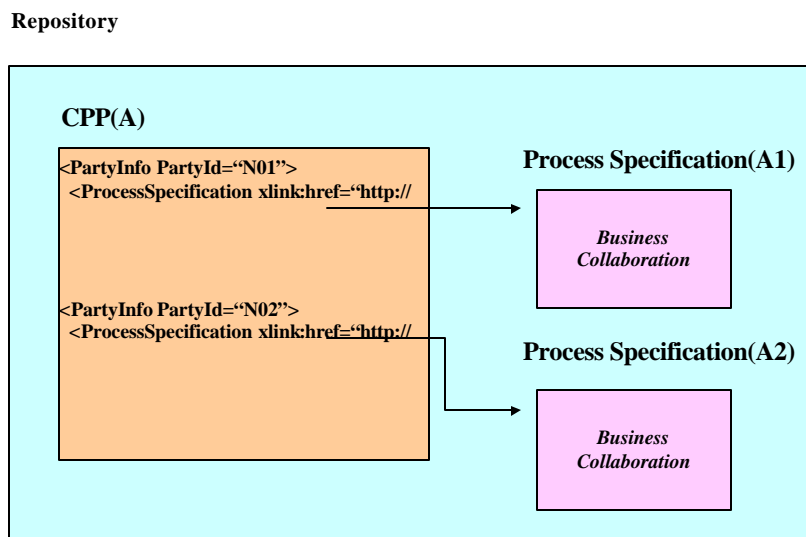
322
323 An enterprise MAY choose to represent itself as multiple *Parties.* For example, it might
324 represent a central office supply procurement organization and a manufacturing supplies
325 procurement organization as separate *Parties.* The enterprise MAY then construct a *CPP* that
326 includes all of its units that are represented as separate *Parties.* In the *CPP*, each of those units
327 would be represented by a separate ***PartyInfo*** element.
328
329 In general, the *Parties* to a *CPA* can have both client and server characteristics. A client requests
330 services and a server provides services to the *Party* requesting services. In some applications,
331 one *Party* only requests services and one *Party* only provides services. These applications have
332 some resemblance to traditional client-server applications. In other applications, each *Party*
333 MAY request services of the other. In that case, the relationship between the two *Parties* can be
334 described as a peer-peer relationship rather than a client-server relationship.
335

336 **6.2 Forming a CPA from Two CPPs**

337 This section summarizes the process of discovering a *Party* to do *Business* with and forming a
338 *CPA* from the two *Parties' CPP*s. In general, this section is an overview of a possible procedure
339 and is not to be considered a normative specification. See Appendix F "Composing a CPA from
340 Two CPPs (Non-Normative)" for more information.
341

342 Figure 2 illustrates forming a *CPP*. *Party* A tabulates the information to be placed in a repository
343 for the discovery process, constructs a *CPP* that contains this information, and enters it into an
344 ebXML Registry or similar repository along with additional information about the *Party*. The
345 additional information might include a description of the *Businesses* that the *Party* engages in.
346 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)

CPP

Party A — Describe → What *Business* capabilities it can perform when conducting a *Business Collaboration* with other parties — Build →

Party's **information**
- *Party* **name**
- **contact info**
Transport Protocol
Transport Security Protocol
Messaging Protocol
Link to Process-
Specification document
Time out/Retry
-etc.
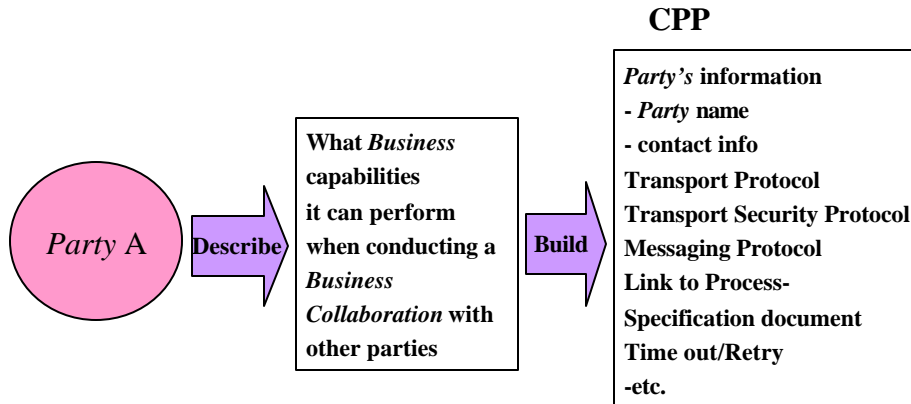
347 repository's discovery services.
348
349 In figure 3, *Party* A and *Party* B use their *CPP*s to jointly construct a single copy of a *CPA* by
350 calculating the intersection of the information in their *CPP*s. The resulting *CPA* defines how the
351 two *Parties* will behave in performing their *Business Collaboration*.

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

CPA

CPP For *Party* A	**negotiate** 1	*CPA* **ID** *Party's* **information** - *Party* A - *Party* B **Transport Protocol** **Transport Security** **DocExchange Protocol** **Link to Process-** **Specification Doc.** **Retry** **-etc.**

negotiate 2 → *CPP* For *Party* B

Agreed *CPA* ← 3 **Agree-ment on *CPA* has arrived.**

3 → Agreed *CPA* **Agree-ment on *CPA* has arrived.**

4 Start Business activities with each other
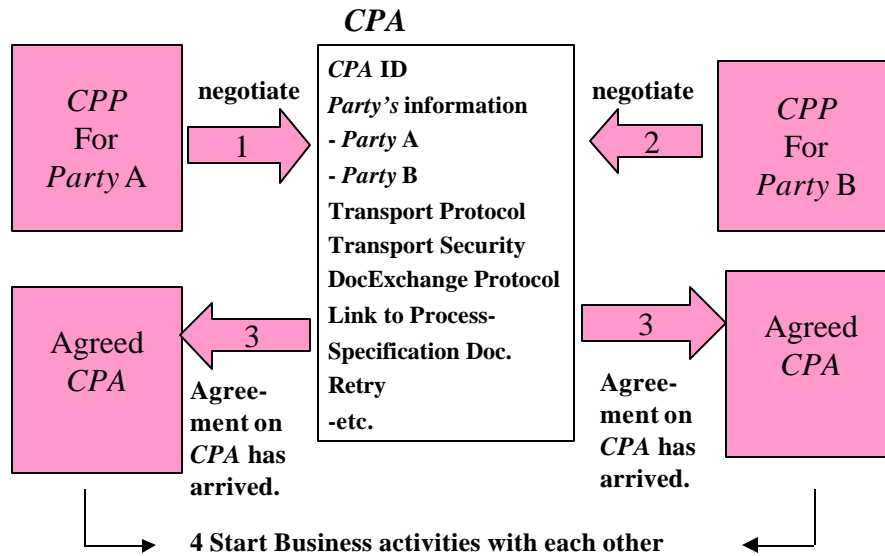
352
353 Figure 4 illustrates the entire process. The steps are listed at the left. The end of the process is
354 that the two *Parties* configure their systems from identical copies of the agreed *CPA* and they are

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*.

Party A (Seller,Server)

5.
(Exe. Code)   (Document)
CPA(A,B)   *CPA*(A,B)

6.   **4.**   **3.**

CPA(A,B)   *CPA*(A,B)
(Exe. Code)   (Document)
5.

Party B (Buyer,Server)

Registry

CPP(A) **1.**
CPP(B) **1.**
2.
CPP(X)
CPP(Y)
CPP(Z)
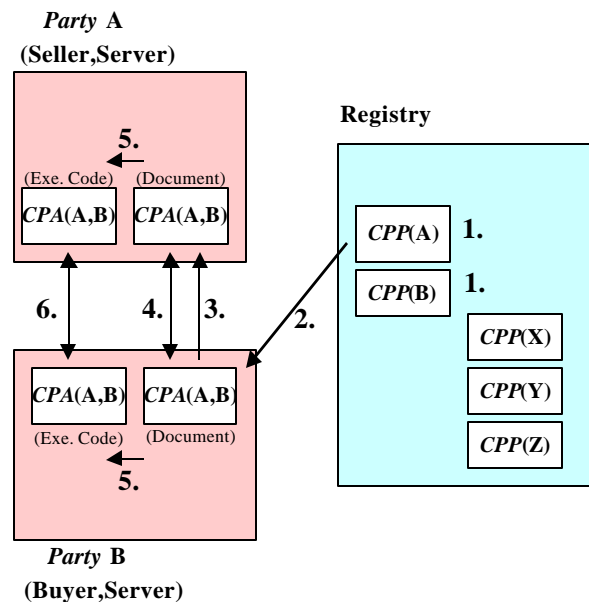
355   then ready to do *Business.*

356

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

361

362 6.3 How the CPA Works

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

370

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

377

378   The *CPA* and the *Process-Specification* document that it references define a conversation
379   between the two *Parties*. The conversation represents a single unit of *Business* as defined by the
380   *Binary-Collaboration* component of the *Process-Specification* document. The conversation
381   consists of one or more *Business Transactions*, each of which is a request *Message* from one
382   *Party* and zero or one response *Message* from the other *Party.* The *Process-Specification*
383   document defines, among other things, the request and response *Message*s for each *Business*
384   *Transaction* and the order in which the *Business Transactions* are REQUIRED to occur. See
385   [ebBPSS] for a detailed explanation.

386

387   The *CPA* MAY actually reference more than one *Process-Specification* document. When a *CPA*
388   references more than one *Process-Specification* document, each *Process-Specification* document
389   defines a distinct type of conversation. Any one conversation involves only a single *Process-*
390   *Specification* document.

391

392   A new conversation is started each time a new unit of *Business* is started. The *Business*
393   *Collaboration* also determines when the conversation ends. From the viewpoint of a *CPA*
394   between *Party* A and *Party* B, the conversation starts at *Party* A when *Party* A sends the first
395   request *Message* to *Party* B. At *Party* B, the conversation starts when it receives the first request
396   of the unit of *Business* from *Party* A. A conversation ends when the *Parties* have completed the
397   unit of *Business*.

398

399   NOTE: The run-time system SHOULD provide an interface by which the *Business*
400   application can request initiation and ending of conversations.
401

6.4 Where the CPA May Be Implemented

403 Conceptually, a *Business*-to-*Business* (B2B) server at each *Party's* site implements the CPA and
404 Process-Specification document. The B2B server includes the run-time software, i.e. the
405 middleware that supports communication with the other *Party*, execution of the functions
406 specified in the *CPA*, interfacing to each *Party's* back-end processes, and logging the interactions
407 between the *Parties* for purposes such as audit and recovery. The middleware might support the
408 concept of a long-running conversation as the embodiment of a single unit of *Business* between
409 the *Parties*. To configure the two *Parties'* systems for *Business* to *Business* operations, the
410 information in the copy of the *CPA* and *Process-Specification* documents at each *Party's* site is
411 installed in the run-time system. The static information MAY be recorded in a local database and
412 other information in the *CPA* and *Process-Specification* document MAY be used in generating or
413 customizing the necessary code to support the *CPA*.
414
415   NOTE: It is possible to provide a graphic *CPP/CPA*-authoring tool that understands both
416   the semantics of the *CPP/CPA* and the XML syntax. Equally important, the definitions in
417   this specification make it feasible to automatically generate, at each *Party's* site, the code
418   needed to execute the *CPA*, enforce its rules, and interface with the *Party's* back-end
419   processes.
420

6.5 Definition and Scope

422
423 This specification defines and explains the contents of the *CPP* and *CPA* XML documents. Its
424 scope is limited to these definitions. It does not define how to compose a *CPA* from two *CPPs*
425 nor does it define anything related to run-time support for the *CPP* and *CPA*. It does include
426 some non-normative suggestions and recommendations regarding run-time support where these
427 notes serve to clarify the *CPP* and *CPA* definitions. See section 10 for a discussion of
428 conformance to this specification.
429
430   NOTE: This specification is limited to defining the contents of the *CPP* and *CPA*, and it is
431   possible to be conformant with it merely by producing a *CPP* or *CPA* document that
432   conforms to the DTD and XML Schema documents defined herein. It is, however, important
433   to understand that the value of this specification lies in its enabling a run-time system that
434   supports electronic commerce between two *Parties* under the guidance of the information in
435   the *CPA*.

7 CPP Definition

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

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

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

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

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

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

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

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

7.1 Globally-Unique Identifier of CPP Instance Document

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

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

7.2 SchemaLocation Attribute

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

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

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

An example of the use of the schemaLocation attribute follows:

```
    <CollaborationProtocolAgreement
        xmlns="http://www.ebxml.org/namespaces/tradePartner"
        xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
        xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
            http://ebxml.org/project_teams/trade_partner/cpp-cpa-
10.xsd"
        ...
        >
        ...
    </CollaborationProtocolAgreement>
```

524   **7.3 CPP Structure**

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

```
527
528   <CollaborationProtocolProfile
529         xmlns="http://www.ebxml.org/namespaces/tradePartner"
530         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
531         xmlns:xlink="http://www.w3.org/1999/xlink"
532         version="1.1">
533         <PartyInfo>  <!--one or more-->
534          ...
535         </PartyInfo>
536         <Packaging id="ID"> <!--one or more-->
537             ...
538         <Packaging>
539         <ds:Signature>  <!--zero or one-->
540         ...
541         </ds:Signature>
542         <Comment>text</Comment> <!--zero or more-->
543         </CollaborationProtocolProfile>
544
```

545   **7.4 CollaborationProtocolProfile element**

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

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

549   - The default namespace: xmlns="http://www.ebxml.org/namespaces/tradePartner",
550   - XML Digital Signature namespace:
551   xmlns:ds="http://www.w3.org/2000/09/xmldsig#",
552   - and the XLINK namespace: xmlns:xlink="http://www.w3.org/1999/xlink".

553
554   In addition, the *CollaborationProtocolProfile* element contains an IMPLIED *version* attribute
555   that indicates the version of the *CPP*. Its purpose is to provide versioning capabilities for
556   instances of an enterprise's *CPP*. The value of the version attribute SHOULD be a string
557   representation of a numeric value such as "1.0" or "2.3". The value of the version string
558   SHOULD be changed with each change made to the *CPP* document after it has been published.

559
560   NOTE: The method of assigning the version-identifier value is left to the implementation.

561
562   The *CollaborationProtocolProfile* element SHALL consist of the following child elements:
563   - One or more REQUIRED *PartyInfo* elements that identify the organization (or parts
564   of the organization) whose capabilities are described by the *CPP*,
565   - One REQUIRED *Packaging* element,
566   - Zero or one *ds:Signature* elements that contain the digital signature that signs the
567   *CPP* document,
568   - Zero or more *Comment* elements.
569

570 A *CPP* document MAY be digitally signed so as to provide for a means of ensuring that the
571 document has not been altered (integrity) and to provide for a means of authenticating the author
572 of the document. A digitally signed *CPP* SHALL be signed using techno logy that conforms to
573 the joint W3C/IETF XML Digital Signature specification[XMLDSIG].
574

575 **7.5 PartyInfo Element**

576 The *PartyInfo* element identifies the organization whose capabilities are described in this *CPP*
577 and includes all the details about this *Party.* More than one *PartyInfo* element MAY be
578 provided in a *CPP* if the organization chooses to represent itself as subdivisions with different
579 characteristics. Each of the subelements of *PartyInfo* is discussed later. The overall structure of
580 the *PartyInfo* element is as follows:
581

```
582  <PartyInfo>
583        <PartyId type="...">  <!--one or more-->
584            ...
585        </PartyId>
586        <PartyRef xlink:type="..", xlink:href="..."/>
587        <CollaborationRole>   <!--one or more-->
588            ...
589        </CollaborationRole>
590        <Certificate>  <!--one or more-->
591            ...
592        </Certificate>
593        <DeliveryChannel>  <!--one or more-->
594        ...
595        </DeliveryChannel>
596        <Transport>  <!--one or more-->
597        ...
598        </Transport>
599        <DocExchange>  <!--one or more-->
600        ...
601        </DocExchange>
602  </PartyInfo>
```
603

604 The *PartyInfo* element consists of the following child elements:
605   • One or more REQUIRED *PartyId* elements that provide a logical identifier for the
606   organization.
607   • A REQUIRED *PartyRef* element that provides a pointer to more information about
608   the *Party.*
609   • One or more REQUIRED *CollaborationRole* elements that identify the roles that this
610   *Party* can play in the context of a *Process Specification*.
611   • One or more REQUIRED *Certificate* elements that identify the certificates used by
612   this *Party* in security functions.
613   • One or more REQUIRED *DeliveryChannel* elements that define the characteristics of
614   each delivery channel that the *Party* can use to receive *Messages.* It includes both the
615   transport level (e.g. HTTP) and the messaging protocol (e.g. ebXML *Message*
616   Service).
617   • One or more REQUIRED *Transport* elements that define the characteristics of the
618   transport protocol(s) that the *Party* can support to receive *Messages*.

619   • One or more REQUIRED *DocExchange* elements that define the *Message*-exchange
620   characteristics, such as the *Message*-exchange protocol, that the *Party* can support.
621

7.5.1 PartyId element

623   The REQUIRED *PartyId* element provides a logical identifier that MAY be used to logically
624   identify the *Party*. Additional *PartyId* elements MAY be present under the same *PartyInfo*
625   element so as to provide for alternative logical identifiers for the *Party*. If the *Party* has
626   preferences as to which logical identifier is used, the *PartyId* elements SHOULD be listed in
627   order of preference starting with the most-preferred identifier.
628

629   In a *CPP* that contains multiple *PartyInfo* elements, different *PartyInfo* elements MAY contain
630   *PartyId* elements that define different logical identifiers. This permits a large organization, for
631   example, to have different identifiers for different purposes.
632

633   The value of the *PartyId* element is any string that provides a unique identifier. The identifier
634   MAY be any identifier that is understood by both *Parties* to a *CPA*. Typically, the identifier
635   would be listed in a well-known directory such as DUNS or in any naming system specified by
636   [ISO6523].
637

638   The *PartyId* element has a single IMPLIED attribute: *type* that has a string value.
639

640   If the *type* attribute is present, then it provides a scope or namespace for the content of the
641   *PartyId* element.
642

643   If the *type* attribute is not present, the content of the *PartyId* element MUST be a URI that
644   conforms to [RFC2396]. It is RECOMMENDED that the value of the *type* attribute be a URN
645   that defines a namespace for the value of the *PartyId* element. Typically, the URN would be
646   registered as a well-known directory of organization identifiers.
647

648   The following example illustrates two URI references.
649

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

652

653   The first example is the URN for the *Party's* DUNS number, assuming that Dun and Bradstreet
654   has registered a URN for DUNS numbers with the Internet Assigned Numbers Authority
655   (IANA). The last field is the DUNS number of the organization.
656

657   The second example shows an arbitrary URN. This might be a URN that the *Party* has
658   registered with IANA to identify itself directly.
659

7.5.2 PartyRef element

661

662   The *PartyRef* element provides a link, in the form of a URI, to additional information about the
663   *Party*. Typically, this would be the URL from which the information can be obtained. The

Collaboration-Protocol Profile Tend Agreement Specification   **Page 19 of 90**

664 information might be at the *Party's* web site or in a publicly accessible repository such as an
665 ebXML Registry, a UDDI repository, or an LDAP directory. Information available at that URI
666 MAY include contact names, addresses, and phone numbers, and perhaps more information
667 about the *Business Collaborations* that the *Party* supports. This information MAY be in the form
668 of an ebXML Core Component[ccOVER]. It is not within the scope of this specification to
669 define the content or format of the information at that URI.
670
671 The *PartyRef* element is an [XLINK] simple link. It has the following attributes:
672   • a REQUIRED *xlink:type* attribute,
673   • a REQUIRED *xlink:href* attribute,
674   • an IMPLIED *type* attribute.
675
676 **7.5.2.1 xlink:type attribute**
677 The REQUIRED *xlink:type* attribute SHALL have a FIXED value of "simple". This identifies
678 the element as being an [XLINK] simple link.
679
680 **7.5.2.2 xlink:href attribute**
681 The REQUIRED *xlink:href* attribute SHALL have a value that is a URI that conforms to
682 [RFC2396] and identifies the location of the external information about the *Party.*
683
684 **7.5.2.3 type attribute**
685 The value of the IMPLIED *type* attribute identifies the document type of the external information
686 about the *Party.* It MUST be a URI that defines the namespace associated with the information
687 about the *Party.* If the *type* attribute is omitted, the external information about the *Party* MUST
688 be an HTML web page.
689
690 An example of the *PartyRef* element is:
691
692   ```
       <PartyRef xlink:type="simple"
693   xlink:href="http://example2.com/ourInfo.xml"
694   type="uri-reference"/>
       ```

695  **7.5.3  CollaborationRole element**

696  The *CollaborationRole* element associates a *Party* with a specific role in the *Business*
697  *Collaboration* that is defined in the *Process-Specification* document[ebBPSS].  Generally, the
698  *Process Specification* is defined in terms of roles such as "buyer" and "seller".  The association
699  between a specific *Party* and the role(s) it is capable of fulfilling within the context of a *Process*
700  *Specification* is defined in both the *CPP* and *CPA* documents.  In a *CPP*, the *CollaborationRole*
701  element identifies which role the *Party* is capable of playing in each *Process Specification*
702  documents referenced by the *CPP*. An example of the *CollaborationRole* element is:
703
704  ```
<CollaborationRole id="N11" >
705      <ProcessSpecification name="BuySell" version="1.0">
706       ...
707      </ProcessSpecification>
708      <Role name="buyer" xlink:href="..."/>
709      <CertificateRef certId = "N03"/>
710        <!-- primary binding with "preferred" DeliveryChannel -->
```

```
711        <ServiceBinding name="some process" channelId="N02"  packageId="N06">
712            <!-- override "default" deliveryChannel  for selected message(s)-->
713            <Override action="OrderAck" channelId="N05" packageId="N09"
714                xlink:type="simple"
715                xlink:href="..."/>
716        </ServiceBinding>
717        <!-- the first alternate binding -->
718        <ServiceBinding channelId="N04" packageId="N06">
719            <Override action="OrderAck" channelId="N05" packageId="N09"
720        xlink:type="simple"
721                xlink:href="..."/>
722        </ServiceBinding>
723 </CollaborationRole>
```

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

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

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

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

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

> NOTE: An implementation MAY provide the capability of dynamically assigning delivery channels on a per *Message* basis during performance of the *Business*

Collaboration-Protocol Profile Tand Agreement Specification   **Page 21 of 90**

760   *Collaboration.* The delivery channel selected would be chosen, based on present
761   conditions, from those identified by *ServiceBinding* elements that refer to the *Business*
762   *Collaboration* that is sending the *Message*. If more than one delivery channel is
763   applicable, the one referred to by the highest-preference *ServiceBinding* element is used.
764

765 The *CollaborationRole* element has the following attribute:
766   • a REQUIRED *id* attribute.
767

768 **7.5.3.1 id attribute**
769 The REQUIRED *id* attribute is an [XML] ID attribute by which this *CollaborationRole* element
770 can be referenced from elsewhere in the *CPP* document.
771

772 **7.5.3.2 CertificateRef element**
773 The EMPTY *CertificateRef* element contains an IMPLIED IDREF attribute, *certId,* which
774 identifies the certificate to be used by referring to the *Certificate* element (under *PartyInfo*) that
775 has the matching ID attribute value.
776

777 **7.5.3.3 certId attribute**
778 The IMPLIED *certId* attribute is an [XML] IDREF that associates the *CollaborationRole* with a
779 *Certificate* with a matching ID attribute*.*
780

781   NOTE: This *certId* attribute relates to the authorizing role in the *Process Specification*
782   while the certificates identified in the delivery-channel description relate to *Message*
783   exchanges.
784

785 **7.5.4 ProcessSpecification element**

786 The *ProcessSpecification* element provides the link to the *Process-Specification* document that
787 defines the interactions between the two *Parties.* It is RECOMMENDED that this *Business-*
788 *Collaboration* description be prepared in accord with the ebXML Business Process Specification
789 Schema[ebBPSS]. The *Process-Specification* document MAY be kept in an ebXML Registry.
790

791   NOTE: A *Party* MAY describe the *Business Collaboration* using any desired alternative
792   to the ebXML Business Process Specification Schema. When an alternative *Business-*
793   *Collaboration* description is used, the *Parties* to a *CPA* MUST agree on how to interpret
794   the *Business-Collaboration* description and how to interpret the elements in the *CPA* that
795   reference information in the *Business-Collaboration* description. The affected elements
796   in the *CPA* are the *Role* element, the *Override* element, and some attributes of the
797   *Characteristics* element.
798

799 The syntax of the *ProcessSpecification* element is:
800

```
801 <ProcessSpecification
802      name="BuySell"
803      version="1.0"
804      xlink:type="simple"
805      xlink:href="http://www.ebxml.org/services/purchasing.xml"
806      <ds:Reference ds:URI="http://www.ebxml.org/services/purchasing.xml">
```

Collaboration-Protocol Profile Tend Agreement Specification   **Page 22 of 90**

```
807              <ds:Transforms>
808                    <ds:Transform
809              ds:Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
810              </ds:Transforms>
811              <ds:DigestMethod
812                    ds:Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1">
813                    String
814              </ds:DigestMethod>
815              <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
816        </ds:Reference>
817 </ProcessSpecification>
818
819
```

820 The ***ProcessSpecification*** element has a single REQUIRED child element, ***ds:Reference,*** and the
821 following attributes:
822   - a REQUIRED ***name*** attribute, with type ID,
823   - a REQUIRED ***version*** attribute,
824   - a FIXED ***xlink:type*** attribute,
825   - a REQUIRED ***xlink:href*** attribute.

826

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

834

835 **7.5.4.1 name attribute**
836 The ***ProcessSpecification*** element MUST include a REQUIRED ***name*** attribute: an [XML] ID
837 that MAY be used to refer to this element from elsewhere within the *CPP* document.

838

839 **7.5.4.2 version attribute**
840 The ***ProcessSpecification*** element includes a REQUIRED ***version*** attribute to identify the
841 version of the *Process-Specification* document identified by the ***xlink:href*** attribute (and also
842 identified by the ***ds:Reference*** element, if any).

843

844 **7.5.4.3 xlink:type attribute**
845 The ***xlink:type*** attribute has a FIXED value of "simple". This identifies the element as being an
846 [XLINK] simple link.

847

848 **7.5.4.4 xlink:href attribute**
849 The REQUIRED ***xlink:href*** attribute SHALL have a value that identifies the *Process-*
850 *Specification* document and is a URI that conforms to [RFC2396].

851

852 **7.5.4.5 ds:Reference element**
853 The ***ds:Reference*** element identifies the same *Process-Specification* document as the enclosing
854 ***ProcessSpecification*** element's ***xlink:href*** attribute and additionally provides for verification that

855   the *Process-Specification* document has not changed since the *CPP* was created.
856

857   NOTE: *Parties* MAY test the validity of the *CPP* or *CPA* at any time. The following
858   validity tests MAY be of particular interest:
859

860   •   test of the validity of a *CPP* and the referenced *Process-Specification* documents at
861   the time composition of a *CPA* begins in case they have changed since they were
862   created,
863   •   test of the validity of a *CPA* and the referenced *Process-Specification* documents at
864   the time a *CPA* is installed into a *Party's* system,
865   •   test of the validity of a *CPA* at intervals after the *CPA* has been installed into a *Party's*
866   system. The *CPA* and the referenced *Process-Specification* documents MAY be
867   processed by an installation tool into a form suited to the particular middleware.
868   Therefore, alterations to the *CPA* and the referenced *Process-Specification* documents
869   do not necessarily affect ongoing run-time operations. Such alterations might not be
870   detected until it becomes necessary to reinstall the *CPA* and the referenced *Process-*
871   *Specification* documents.
872

873   The syntax and semantics of the ***ds:Reference*** element and its child elements are defined in the
874   XML Digital Signature specification[XMLDSIG]. As an alternative to the string value of the
875   ***ds:DigestMethod***, shown in the above example, the child element, ***ds:HMACOutputLength***,
876   with a string value, MAY be used.
877

878   According to [XMLDSIG], a ***ds:Reference*** element can have a ***ds:Transforms*** child element,
879   which in turn has an ordered list of one or more ***ds:Transform*** child elements to specify a
880   sequence of transforms. However, this specification currently REQUIRES the Canonical
881   XML[XMLC14N] transform and forbids other transforms. Therefore, the following additional
882   requirements apply to a ***ds:Reference*** element within a ***ProcessSpecification*** element:
883

884   •   The ***ds:Reference*** element MUST have a ***ds:Transforms*** child element.
885   •   That ***ds:Transforms*** element MUST have exactly one ***ds:Transform*** child element.
886   •   That ***ds:Transform*** element MUST specify the Canonical XML[XMLC14N]
887   transform via the following REQUIRED value for its REQUIRED ***ds:Algorithm***
888   attribute: http://www.w3.org/TR/2000/CR-xml-c14n-20001026
889

890   Note that implementation of Canonical XML is REQUIRED by the XML Digital Signature
891   specification[XMLDSIG].
892

893   A ***ds:Reference*** element in a ***ProcessSpecification*** element has implications for *CPP* validity:
894

895   •   A *CPP* MUST be considered invalid if any ***ds:Reference*** element within a
896   ***ProcessSpecification*** element fails reference validation as defined by the XML Digital
897   Signature specification[XMLDSIG].
898

899   •   A *CPP* MUST be considered invalid if any ***ds:Reference*** within it cannot be
900   dereferenced.

901
902   Other validity implications of such ***ds:Reference*** elements are specified in the description of the
903   ***ds:Signature*** element.
904
905   NOTE: The XML Digital Signature specification[XMLDSIG] states "The signature
906   application MAY rely upon the identification (URI) and Transforms provided by the
907   signer in the Reference element, or it MAY obtain the content through other means such
908   as a local cache" (emphases on MAY added). However, it is RECOMMENDED that
909   ebXML *CPP*/*CPA* implementations not make use such cached results when signing or
910   validating.
911
912   NOTE: It is recognized that the XML Digital Signature specification[XMLDSIG]
913   provides for signing an XML document together with externally referenced documents.
914   In cases where a *CPP* or *CPA* document is in fact suitably signed, that facility could also
915   be used to ensure that the referenced *Process-Specification* documents are unchanged.
916   However, this specification does not currently mandate that a *CPP* or *CPA* be signed.
917
918   NOTE: If the *Parties* to a *CPA* wish to customize a previously existing *Process-*
919   *Specification* document, they MAY copy the existing document, modify it, and cause
920   their *CPA* to reference the modified copy. It is recognized that for reasons of clarity,
921   brevity, or historical record, the parties might prefer to reference a previously existing
922   *Process-Specification* document in its original form and accompany that reference with a
923   specification of the agreed modifications. Therefore, *CPP* usage of the ***ds:Reference***
924   element's ***ds:Transforms*** subelement within a ***ProcessSpecification*** element might be
925   expanded in the future to allow other transforms as specified in the XML Digital
926   Signature specification[XMLDSIG]. For example, modifications to the original
927   document could then be expressed as XSLT transforms. After applying any transforms,
928   it would be necessary to validate the transformed document against the ebXML Business
929   Process Specification Schema[ebBPSS].
930

7.5.5 Role element

931

932   The REQUIRED ***Role*** element identifies which role in the *Process Specification* the *Party* is
933   capable of supporting via the ***ServiceBinding*** element(s) siblings within this ***CollaborationRole***
934   element.
935
936   The ***Role*** element has the following attributes:
937   • a REQUIRED ***name*** attribute,
938   • a FIXED ***xlink:type*** attribute,
939   • a REQUIRED ***xlink:href*** attribute.
940

7.5.5.1 name attribute

941

942   The REQUIRED ***name*** attribute is a string that gives a name to the ***Role***. Its value is taken from
943   one of the following sources in the *Process Specification*[ebBPSS] that is referenced by the
944   ***ProcessSpecification*** element depending upon which element is the "root" (highest order) of the
945   process referenced:
946   • ***name*** attribute of a ***BinaryCollaboration***/***initiatingRole*** element,

947   • *name* attribute of a ***BinaryCollaboration***/***respondingRole*** element,
948   • *fromAuthorizedRole* attribute of a ***BusinessTransactionActivity*** element,
949   • *toAuthorizedRole* attribute of a ***BusinessTransactionActivity*** element,
950   • *fromAuthorizedRole* attribute of a ***CollaborationActivity*** element,
951   • *toAuthorizedRole* attribute of a ***CollaborationActivity*** element,
952   • *name* attribute of the ***business-partner-role*** element.
953
954   See NOTE in section 7.5.4 regarding alternative *Business-Collaboration* descriptions.
955
956   **7.5.5.2 xlink:type attribute**
957   The *xlink:type* attribute has a FIXED value of "simple". This identifies the element as being an
958   [XLINK] simple link.
959
960   **7.5.5.3 xlink:href attribute**
961   The REQUIRED *xlink:href* attribute SHALL have a value that is a URI that conforms to
962   [RFC2396]. It identifies the location of the element or attribute within the *Process-Specification*
963   document that defines the role in the context of the *Business Collaboration.* An example is:
964
965   Xlink:href="http://www.ebxml.org/processes/purchasing#N05
966
967   Where "N05" is the value of the ID attribute of the element in the Process-Specification
968   document that defines the role name.
969
970   **7.5.6 ServiceBinding element**

971   The *ServiceBinding* element identifies a default *DeliveryChannel* element for all of the *Message*
972   traffic that is to be sent to the *Party* within the context of the identified *Process-Specification*
973   document. An example of the *ServiceBinding* element is:
974

```
975     <ServiceBinding channelId="X03" packageId="N06">
976                 <Service type="string">serviceName</Service>
977                 <Override action="OrderAck"
978                     channelId="X04"
979                     packageId="N09"
980                     xlink:type="simple"
981                     xlink"href="..."/>  <!--zero or more-->
982         </ServiceBinding>
```

983
984   The *ServiceBinding* element SHALL have one child *Service* element and zero or more *Override*
985   child elements.
986
987   The *ServiceBinding* element has the following attributes:
988   • a REQUIRED *channelId* attribute,
989   • a REQUIRED *packageId* attribute.
990
991   **7.5.6.1 channelId attribute**
992   The REQUIRED *channelId* attribute is an [XML] IDREF that identifies the *DeliveryChannel*
993   that SHALL provide a default technical binding for all of the *Message* traffic that is received for

Collaboration-Protocol Profile Tand Agreement Specification   **Page 26 of 90**

994   the *Process Specification* that is referenced by the ***ProcessSpecification*** element.
995

7.5.6.2 packageId attribute

997   The REQUIRED ***packageId*** attribute is an [XML] IDREF that identifies the ***Packaging*** element
998   that SHALL be used with the ***ServiceBinding*** element.
999

1000   **7.5.7 Service element**

1001   The value of the ***Service*** element is a string that SHALL be used as the value of the ***Service***
1002   element in the ebXML *Message Header*[ebMS] or a similar element in the *Message Header* of
1003   an alternative *message* service. The ***Service*** element has an IMPLIED ***type*** attribute.
1004

1005   If the *Process-Specification* document is defined by the ebXML Business Process Specification
1006   Schema[ebBPSS], then the value of the ***Service*** element is an overall identifier for the set of
1007   *Business Transactions* associated with the authorized role corresponding to the role identified in
1008   the parent ***CollaborationRole*** element.
1009

1010   NOTE: The purpose of the ***Service*** element is only to provide routing information for the
1011   ebXML *Message Header*. The ***CollaborationRole*** element and its child elements identify
1012   the information in the ***ProcessSpecification*** document that is relevant to the *CPP* or *CPA*.
1013

1014   **7.5.7.1 type attribute**

1015   If the ***type*** attribute is present, it indicates that the *Parties* sending and receiving the *Message*
1016   know, by some other means, how to interpret the value of the ***Service*** element. The two *Parties*
1017   MAY use the value of the ***type*** attribute to assist the interpretation.
1018

1019   If the ***type*** attribute is not present, the value of the ***Service*** element MUST be a URI[RFC2396].
1020

1021   **7.5.8 Override element**

1022   The ***Override*** element provides a *Party* with the ability to map, or bind, a different
1023   ***DeliveryChannel*** to *Messages* of a selected *Business Transaction* that are to be received by the
1024   *Party* within the context of the parent ***ServiceBinding*** element.
1025

1026   Each ***Override*** element SHALL specify a different ***DeliveryChannel*** for selected *Messages* that
1027   are to be received by the *Party* in the context of the *Process Specification* that is associated with
1028   the parent ***ServiceBinding*** element. The ***Override*** element has the following attributes:

1029   • a REQUIRED ***action*** attribute,
1030   • a REQUIRED ***channelId*** attribute,
1031   • a REQUIRED ***packageId*** attribute,
1032   • an IMPLIED ***xlink:href*** attribute,
1033   • a FIXED ***xlink:type*** attribute.
1034

1035   Under a given ***ServiceBinding*** element, there SHALL be only one ***Override*** element whose
1036   ***action*** attribute has a given value.
1037

1038   NOTE: It is possible that when a *CPA* is composed from two *CPPs*, a delivery channel in
1039   one *CPP* might have an *Override* element that will not be compatible with the other *Party*.
1040   This incompatibility MUST be resolved either by negotiation or by reverting to a compatible
1041   default delivery channel.

1042

1043   **7.5.8.1 action attribute**

1044   The value of the REQUIRED *action* attribute is a string that identifies the *Business Transaction*
1045   that is to be associated with the **DeliveryChannel** that is identified by the **channelId** attribute. If
1046   the *Process-Specification* document is defined by the ebXML Business Process Specification
1047   Schema[ebBPSS], the value of the *action* attribute MUST match the value of the *name* attribute
1048   of the desired **BusinessTransaction** element in the *Process-Specification* document that is
1049   referenced by the **ProcessSpecification** element.

1050

1051   See NOTE in section 7.5.4 regarding alternative *Business-Collaboration* descriptions.

1052

1053   **7.5.8.2 channelId attribute**

1054   The REQUIRED *channelId* attribute is an [XML] IDREF that identifies the **DeliveryChannel**
1055   element that is to be associated with the *Message* that is identified by the **action** attribute.

1056

1057   **7.5.8.3 packageId attribute**

1058   The REQUIRED *packageId* attribute is an [XML] IDREF that identifies the **Packaging** element
1059   that is to be associated with the *Message* that is identified by the **action** attribute*.

1060

1061   **7.3.7.4 xlink:href attribute**

1062   The IMPLIED *xlink:href* attribute MAY be present. If present, it SHALL provide an absolute
1063   [XPOINTER] URI expression that specifically identifies the **BusinessTransaction** element
1064   within the associated *Process-Specification* document[ebBPSS] that is identified by the
1065   **ProcessSpecification** element.

1066

1067   **7.3.7.5 xlink:type attribute**

1068   The IMPLIED *xlink:type* attribute has a FIXED value of "simple". This identifies the element as
1069   being an [XLINK] simple link.

1070

1071   **7.5.9 Certificate element**

1072   The *Certificate* element defines certificate information for use in this *CPP*. One or more
1073   *Certificate* elements MAY be provided for use in the various security functions in the *CPP*. An
1074   example of the *Certificate* element is:

1075

```
1076        <Certificate certId = "N03">
1077            <ds:KeyInfo>. . .</ds:KeyInfo>
1078        </Certificate>
```

1079

1080   The *Certificate* element has a single REQUIRED attribute: *certId*. The *Certificate* element has a
1081   single child element: *ds:KeyInfo*.

1082

1083 **7.5.9.1 certId attribute**
1084 The REQUIRED *certId* attribute is an ID attribute. Its is referred to in a *CertificateRef* element,
1085 using an IDREF attribute, where a certificate is specified elsewhere in the *CPP*. For example:
1086
1087   `<CertificateRef certId = "N03"/>`
1088
1089 **7.5.9.2 ds:KeyInfo element**
1090 The *ds:KeyInfo* element define s the certificate information. The content of this element and any
1091 subelements are defined by the XML Digital Signature specification[XMLDSIG].
1092
1093   NOTE: Software for creation of *CPPs* and *CPAs* MAY recognize the *ds:KeyInfo* element
1094   and insert the subelement structure necessary to define the certificate.
1095
1096 **7.5.10 DeliveryChannel element**

1097 A delivery channel is a combination of a *Transport* element and a *DocExchange* element that
1098 describes the *Party's Message*-receiving characteristics. The *CPP* SHALL contain one or more
1099 *DeliveryChannel* elements, one or more *Transport* elements, and one or more *DocExchange*
1100 elements. Each delivery channel MAY refer to any combination of a *DocExchange* element and
1101 a *Transport* element. The same *DocExchange* element or the same *Transport* element MAY be
1102 referred to by more than one delivery channel. Two delivery channels MAY use the same
1103 transport protocol and the same document-exchange protocol and differ only in details such as

Figure 5: Three Delivery Channels
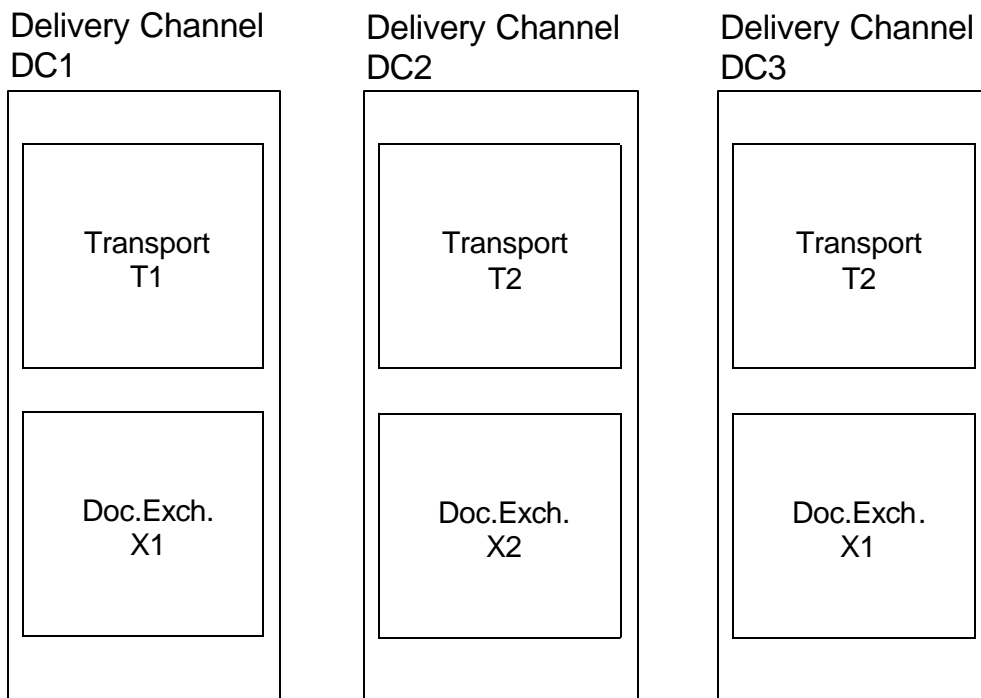
1104 communication addresses or security definitions. Figure 5 illustrates three delivery channels.
1105

1106   The delivery channels have ID attributes with values "DC1", "DC2", and "DC3". Each delivery
1107   channel contains one transport definition and one document-exchange definition. Each transport
1108   definition and each document-exchange definition also has a name as shown in the figure. Note
1109   that delivery-channel DC3 illustrates that a delivery channel MAY refer to the same transport
1110   definition and document-exchange definition used by other delivery channels but a different
1111   combination. In this case delivery-channel DC3 is a combination of transport definition T2 (also
1112   referred to by delivery-channel DC2) and document-exchange definition X1 (also referred to by
1113   delivery-channel DC1).

1114

1115   A specific delivery channel SHALL be associated with each *ServiceBinding* element or
1116   *Override* element (*action* attribute). Following is the delivery-channel syntax.

1117

```
1118        <DeliveryChannel channelId="N04" transportId="N05" docExchangeId="N06">
1119             <Characteristics
1120                  syncReplyMode = "responseOnly"
1121                  nonrepudiationOfOrigin = "true"
1122                  nonrepudiationOfReceipt = "true"
1123                  secureTransport = "true"
1124                  confidentiality = "true"
1125                  authenticated = "true"
1126                  authorized = "true"/>
1127        </DeliveryChannel>
```

1128

1129   Each *DeliveryChannel* element identifies one *Transport* element and one *DocExchange* element
1130   that make up a single delivery channel definition.

1131

1132   The *DeliveryChannel* element has the following attributes:
1133   • a REQUIRED *channelId* attribute,
1134   • a REQUIRED *transportId* attribute,
1135   • a REQUIRED *docExchangeId* attribute.

1136

1137   The *DeliveryChannel* element has one REQUIRED child element, *Characteristics.*

1138

1139 7.5.10.1 channelId attribute
1140   The *channelId* attribute is an [XML] ID attribute that uniquely identifies the *DeliveryChannel*
1141   element for reference, using IDREF attributes, from other parts of the *CPP* or *CPA*.

1142

1143 7.5.10.2 transportId attribute
1144   The *transportId* attribute is an [XML] IDREF that identifies the *Transport* element that defines
1145   the transport characteristics of the delivery channel. It MUST have a value that is equal to the
1146   value of a *transportId* attribute of a *Transport* element elsewhere within the *CPP* document.

1147

1148 7.5.10.3 docExchangeId attribute
1149   The *docExchangeId* attribute is an [XML] IDREF that identifies the *DocExchange* element that
1150   defines the document-exchange characteristics of the delivery channel. It MUST have a value
1151   that is equal to the value of a *docExchangeId* attribute of a *DocExchange* element elsewhere
1152   within the *CPP* document.

1153

1154 **7.5.11 Characteristics element**

1155 The ***Characteristics*** element describes the security characteristics and other attributes of the
1156 delivery channel. The attributes of the ***Characteristics*** element, except ***syncReplyMode***, MAY be
1157 used to override the values of the corresponding attributes in the *Process-Specification*
1158 document.
1159
1160 See NOTE in section 7.5.4 regarding alternative *Business-Collaboration* descriptions.
1161
1162 The ***Characteristics*** element has the following attributes:
1163   • An IMPLIED ***syncReplyMode*** attribute,
1164   • an IMPLIED ***nonrepudiationOfOrigin*** attribute,
1165   • an IMPLIED ***nonrepudiationOfReceipt*** attribute,
1166   • an IMPLIED ***secureTransport*** attribute,
1167   • an IMPLIED ***confidentiality*** attribute,
1168   • an IMPLIED ***authenticated*** attribute,
1169   • an IMPLIED ***authorized*** attribute.
1170
1171 **7.5.11.1 syncReplyMode attribute**
1172 The ***syncReplyMode*** attribute is an enumeration comprised of the following possible values:
1173   • "signalsOnly"
1174   • "responseOnly"
1175   • "signalsAndResponse"
1176   • "none"
1177
1178 This attribute, when present, indicates what the receiving application expects in a response when
1179 bound to a synchronous communication protocol such as HTTP. The value of "signalsOnly"
1180 indicates that the response returned (on the HTTP 200 response in the case of HTTP) will only
1181 include one or more *Business* signals as defined in the *Process Specification* document[ebBPSS],
1182 but not a *Business*-response *Message*. The value of "responseOnly" indicates that only the
1183 *Business*-response *Message* will be returned. The value of "signalsAndResponse" indicates that
1184 the application will return the *Business*-response *Message* in addition to one or more *Business*
1185 signals. The value of "none", which is the implied default value in the absence of the
1186 ***syncReplyMode*** attribute, indicates that neither the *Business*-response *Message* nor any *Business*
1187 signals will be returned synchronously. In this case, the *Business*-response *Message* and any
1188 *Business* signals will be returned as separate asynchronous responses.
1189
1190 The ebXML *Message* Service's ***syncReply*** attribute is set to a value of "true" whenever the
1191 ***syncReplyMode*** attribute has a value other than "none".
1192
1193 If the delivery channel identifies a transport protocol that has no synchronous capabilities (such
1194 as SMTP) and the ***Characteristics*** element has a ***syncReplyMode*** attribute with a value other
1195 than "none", a response SHALL contain the same content as if the transport protocol did support
1196 synchronous responses.
1197

1198 **7.5.11.2 nonrepudiationOfOrigin attribute**

1199 The *nonrepudiationOfOrigin* attribute is a Boolean with possible values of "true" and "false".
1200 If the value is "true" then the delivery channel REQUIRES the *Message* to be digitally signed by
1201 the certificate of the *Party* that sent the *Message*.

1202

1203 **7.5.11.3 nonrepudiationOfReceipt attribute**

1204 The *nonrepudiationOfReceipt* attribute is a Boolean with possible values of "true" and "false".
1205 If the value is "true" then the delivery channel REQUIRES that the *Message* be acknowledged by
1206 a digitally signed *Message*, signed by the certificate of the *Party* that received the *Message*, that
1207 includes the digest of the *Message* being acknowledged.

1208

1209 **7.5.11.4 secureTransport attribute**

1210 The *secureTransport* attribute is a Boolean with possible values of "true" and "false". If the
1211 value is "true" then it indicates that the delivery channel uses a secure transport protocol such as
1212 [SSL] or [IPSEC].

1213

1214 **7.5.11.5 confidentiality attribute**

1215 The *confidentiality* attribute is a Boolean with possible values of "true" and "false". If the value
1216 is "true" then it indicates that the delivery channel REQUIRES that the *Message* be encrypted in
1217 a persistent manner. It MUST be encrypted above the level of the transport and delivered,
1218 encrypted, to the application.

1219

1220 **7.5.11.6 authenticated attribute**

1221 The *authenticated* attribute is a Boolean with possible values of "true" and "false". If the value
1222 is "true" then it indicates that the delivery channel REQUIRES that the sender of the *Message* be
1223 authenticated before delivery to the application.

1224

1225 **7.5.11.7 authorized attribute**

1226 The *authorized* attribute is a Boolean with possible of values of "true" and "false". If the value
1227 is "true" then it indicates that the delivery channel REQUIRES that the sender of the *Message* be
1228 authorized before delivery to the application.

1229

1230 **7.5.12 Transport element**

1231 The *Transport* element of the *CPP* defines the *Party's* capabilities with regard to communication
1232 protocol, encoding, and transport security information.

1233

1234 The overall structure of the *Transport* element is as follows:

1235

```
1236        <Transport transportId = "N05">
1237             <!--protocols are HTTP, SMTP, and FTP-->
1238             <SendingProtocol version = "1.1">HTTP</SendingProtocol>
1239                  <!--one or more SendingProtocol elements-->
1240             <ReceivingProtocol version = "1.1">HTTP</ReceivingProtocol>
1241             <!--one or more endpoints-->
1242             <Endpoint uri="http://example.com/servlet/ebxmlhandler"
1243                  type = "request"/>
1244             <TransportSecurity>  <!--0 or 1 times-->
```

Collaboration-Protocol Profile Tend Agreement Specification   **Page 32 of 90**

```
1245                    <Protocol version = "3.0">SSL</Protocol>
1246                    <CertificateRef certId = "N03"/>
1247            </TransportSecurity>
1248        </Transport>
1249
```

1250 **7.5.12.1 transportId attribute**

1251 The *Transport* element has a single REQUIRED *transportId* attribute, of type [XML] ID, that
1252 provides a unique identifier for each *Transport* element, which SHALL be referred to by the
1253 *transportId* IDREF attribute in a *DeliveryChannel* element elsewhere within the *CPP* or *CPA*
1254 document.

1255

1256 **7.5.12.2 Synchronous Responses**

1257

1258 One distinguishing characteristic of transport protocols is whether a given transport protocol
1259 supports synchronous replies. See section 7.5.11.1 for a discussion of synchronous replies.

1260

1261 **7.5.13 Transport protocol**

1262 Supported communication protocols are HTTP, SMTP, and FTP. The *CPP* MAY specify as
1263 many protocols as the *Party* is capable of supporting.

1264

1265   NOTE: It is the aim of this specification to enable support for any transport capable of
1266   carrying MIME content using the vocabulary defined herein.

1267

1268 **7.5.13.1 SendingProtocol element**

1269 The *SendingProtocol* element identifies the protocol that a *Party* can, or will, use to send
1270 *Business* data to its intended collaborator. The IMPLIED *version* attribute identifies the specific
1271 version of the protocol. For example, suppose that within a *CPP*, a *Transport* element,
1272 containing *SendingProtocol* elements whose values are SMTP and HTTP, is referenced within a
1273 *DeliveryChannel* element. Suppose, further, that this *DeliveryChannel* element is referenced for
1274 the role of Seller within a purchase-ordering process. Then the party is asserting that it can send
1275 purchase orders by either SMTP or HTTP. In a *CPP*, the *SendingProtocol* element MAY appear
1276 one or more times under each *Transport* element. In a *CPA*, the *SendingProtocol* element
1277 SHALL appear once.

1278

1279 **7.5.13.2 ReceivingProtocol element**

1280 The *ReceivingProtocol* element identifies the protocol by which a *Party* can receive its *Business*
1281 data from the other *Party*. The IMPLIED *version* attribute identifies the specific version of the
1282 protocol. For example, suppose that within a *CPP*, a *Transport* element is referenced within a
1283 *DeliveryChannel* element containing a *ReceivingProtocol* element whose value is HTTP.
1284 Suppose further that this *DeliveryChannel* element is referenced for the role of seller within a
1285 purchase ordering *Business Collaboration*. Then the party is asserting that it can receive *Business*
1286 responses to purchase orders over HTTP.

1287

1288 Within a *CPA*, the *SendingProtocol* and *ReceivingProtocol* elements serve to indicate the actual
1289 agreement upon what transports will be used for the complementary roles of the collaborators.
1290 For example, continuing the earlier examples, the seller in a purchase-order *Business*

1291   *Collaboration* could specify its receiving protocol to be SMTP and its sending protocol to be
1292   HTTP. These collaborator capabilities would match the buyer capabilities indicated in the *CPP*.
1293   These matches support an interoperable transport agreement where the buyer would send
1294   purchase orders by SMTP and where the responses to purchase orders (acknowledgements,
1295   cancellations, or change requests, for example) would be sent by the seller to the buyer using
1296   HTTP.
1297
1298   To fully describe receiving transport capabilities, the receiving-protocol information needs to be
1299   combined with URLs that provide the endpoints (see below).
1300
1301   NOTE: Though the URL scheme gives information about the protocol used, an explicit
1302   *ReceivingProtocol* element remains useful for future extensibility to protocols all of
1303   whose endpoints are identified by the same URL schemes, such as distinct transport
1304   protocols that all make use of HTTP endpoints. Likewise, both URL schemes of HTTP://
1305   and HTTPS:// can be regarded as the same receiving protocol since HTTPS is HTTP with
1306   [SSL] for the transport-security protocol. Therefore, the *ReceivingProtocol* element is
1307   separated from the endpoints, which are, themselves, needed to provide essential
1308   information needed for connections.
1309

7.5.14 Endpoint element

1311   The REQUIRED *uri* attribute of the *Endpoint* element specifies the *Party's* communication
1312   addressing information associated with the *ReceiveProtocol* element. One or more *Endpoint*
1313   elements SHALL be provided for each *Transport* element in order to provide different addresses
1314   for different purposes. The value of the *uri* attribute is a URI that contains the electronic address
1315   of the *Party* in the form REQUIRED for the selected protocol. The value of the *uri* attribute
1316   SHALL conform to the syntax for expressing URIs as defined in [RFC2396].
1317
1318   The *type* attribute identifies the purpose of this endpoint. The value of *type* is an enumeration;
1319   permissible values are "login", "request", "response", "error", and "allPurpose". There can be, at
1320   most, one of each. The *type* attribute MAY be omitted. If it is omitted, its value defaults to
1321   "allPurpose". The "login" endpoint MAY be used for the address for the initial *Message* between
1322   the two *Parties*. The "request" and "response" endpoints are used for request and response
1323   *Messages*, respectively. The "error" endpoint MAY be used as the address for error *Messages*
1324   issued by the messaging service. If no "error" endpoint is defined, these error *Messages* SHALL
1325   be sent to the "response" address, if defined, or to the "allPurpose" endpoint. To enable error
1326   *Messages* to be received, each *Transport* element SHALL contain at least one endpoint of type
1327   "error", "response", or "allPurpose".
1328

7.5.15 Transport protocols

1330   In the following sections, we discuss the specific details of each supported transport protocol.
1331

7.5.15.1 HTTP

1333   HTTP is Hypertext Transfer Protocol[HTTP]. For HTTP, the address is a URI that SHALL
1334   conform to [RFC2396]. Depending on the application, there MAY be one or more endpoints,
1335   whose use is determined by the application.

1336
1337   Following is an example of an HTTP endpoint:

1338
1339   <Endpoint uri="http://example.com/servlet/ebxmlhandler"
1340   type = "request"/>

1341
1342   The "request" and "response" endpoints MAY be dynamically overridden for a particular
1343   request or asynchronous response by application-specified URIs exchanged in *Business*
1344   documents exchanged under the *CPA*.

1345
1346   For a synchronous response, the "response" endpoint is ignored if present. A synchronous
1347   response is always returned on the existing connection, i.e. to the URI that is identified as the
1348   source of the connection.

1349
1350   **7.5.15.2 SMTP**
1351   SMTP is Simple Mail Transfer Protocol[SMTP]. For use with this standard, Multipurpose
1352   Internet Mail Extensions[MIME] MUST be supported. The MIME media type used by the
1353   SMTP transport layer is "Application" with a sub-type of "octet-stream".

1354
1355   For SMTP, the communication address is the fully qualified mail address of the destination *Party*
1356   as defined by [RFC822]. Following is an example of an SMTP endpoint:

1357
1358   <Endpoint uri="mailto:ebxmlhandler@example.com"
1359   type = "request"/>

1360
1361   SMTP with MIME automatically encodes or decodes the document as required, on each link in
1362   the path, and presents the decoded document to the destination document-exchange function.

1363
1364   NOTE: The SMTP mail transfer agent encodes binary data (i.e. data that are not 7-bit
1365   ASCII) unless it is aware that the upper level (mail user agent) has already encoded the
1366   data.

1367
1368   NOTE: SMTP by itself (without any authentication or encryption) is subject to denial of
1369   service and masquerading by unknown *Parties*. It is strongly suggested that those *Parties*
1370   who choose SMTP as their transport layer also choose a suitable means of encryption and
1371   authentication either in the document-exchange layer or in the transport layer such as
1372   [S/MIME].

1373
1374   NOTE: SMTP is an asynchronous protocol that does not guarantee a particular quality of
1375   service. A transport-layer acknowledgment (i.e. an SMTP acknowledgment) to the
1376   receipt of a mail *Message* constitutes an assertion on the part of the SMTP server that it
1377   knows how to deliver the mail *Message* and will attempt to do so at some point in the
1378   future. However, the *Message* is not hardened and might never be delivered to the
1379   recipient. Furthermore, the sender will see a transport-layer acknowledgment only from
1380   the nearest node. If the *Message* passes through intermediate nodes, SMTP does not
1381   provide an end-to-end acknowledgment. Therefore receipt of an SMTP
1382   acknowledgement does not guarantee that the *Message* will be delivered to the

1383   application and failure to receive an SMTP acknowledgment is not evidence that the
1384   *Message* was not delivered. It is recommended that the reliable-messaging protocol in
1385   the ebXML *Message* Service be used with SMTP.

1386

1387   **7.5.15.3 FTP**
1388   FTP is File Transfer Protocol[RFC959].

1389

1390   Since a delivery channel specifies receive characteristics, each *Party* sends a *Message* using FTP
1391   PUT. The endpoint specifies the user id and input directory path (for PUTs to this *Party*). An
1392   example of an FTP endpoint is:

1393

```
1394            <Endpoint uri="ftp://userid@server.foo.com"
1395                  type = "request"/>
```

1396

1397   Since FTP must be compatible across all implementations, the FTP for ebXML will use the
1398   minimum sets of commands and parameters available for FTP as specified in [RFC959], section
1399   5.1, and modified in [RFC1123], section 4.1.2.13. The mode SHALL be stream only and the
1400   type MUST be either ASCII Non-print (AN), Image (I) (binary), or Local 8 (L 8) (binary
1401   between 8-bit machines and machines with 36 bit words – for an 8-bit machine Local 8 is the
1402   same as Image).

1403

1404   Stream mode closes the data connection upon end of file. The server side FTP MUST set control
1405   to "PASV" before each transfer command to obtain a unique port pair if there are multiple third
1406   party sessions.

1407

1408   NOTE: [RFC 959] states that User-FTP SHOULD send a PORT command to assign a
1409   non-default data port before each transfer command is issued to allow multiple transfers
1410   during a single FTP because of the long delay after a TCP connection is closed until its
1411   socket pair can be reused.

1412

1413   NOTE: The format of the 227 reply to a PASV command is not well-standardized and an
1414   FTP client may assume that the parentheses indicated in [RFC959] will be present when
1415   in some cases they are not. If the User-FTP program doesn't scan the reply for the first
1416   digit of host and port numbers, the result will be that the User-FTP might point at the
1417   wrong host. In the response, the h1, h2, h3, h4 is the IP address of the server host and the
1418   p1, p2 is a non-default data transfer port that PASV has assigned.

1419

1420   NOTE: As a recommendation for firewall transparency, [RFC1579] proposes that the
1421   client sends a PASV command, allowing the server to do a passive TCP open on some
1422   random port, and inform the client of the port number. The client can then do an active
1423   open to establish the connection.

1424

1425   NOTE: Since STREAM mode closes the data connection upon end of file, the receiving
1426   FTP may assume abnormal disconnect if a 226 or 250 control code hasn't been received
1427   from the sending machine.

1428

1429   NOTE: [RFC1579] also makes the observation that it might be worthwhile to enhance the

1430   FTP protocol to have the client send a new command APSV (all passive) at startup that
1431   would allow a server that implements this option to always perform a passive open. A
1432   new reply code 151 would be issued in response to all file transfer requests not preceded
1433   by a PORT or PASV command; this *Message* would contain the port number to use for
1434   that transfer. A PORT command could still be sent to a server that had previously
1435   received APSV; that would override the default behavior for the next transfer operation,
1436   thus permitting third-party transfers.

1437

7.5.16 Transport security

1438

1439   The *TransportSecurity* element provides the *Party's* security specifications, associated with the
1440   *ReceivingProtocol* element, for the transport layer of the *CPP*. It MAY be omitted if transport
1441   security will not be used for any *CPAs* composed from this *CPP*. Unless otherwise specified
1442   below, transport security applies to *Messages* in both directions.

1443

1444   Following is the syntax:

1445

```
1446        <TransportSecurity>
1447            <Protocol version = "3.0">SSL</Protocol>
1448            <CertificateRef certId = "N03"/> <!--zero or one-->
1449        </TransportSecurity>
```

1450

1451   The *TransportSecurity* element contains two REQUIRED child elements, *Protocol* and
1452   *CertificateRef.*

1453

7.5.16.1 Protocol element

1454

1455   The value of the *Protocol* element can identify any transport security protocol that the *Party* is
1456   prepared to support. The IMPLIED *version* attribute identifies the version of the specified
1457   protocol.

1458

1459   The specific security properties depend on the services provided by the identified protocol. For
1460   example, SSL performs certificate-based encryption and certificate-based authentication.

1461

1462   Whether authentication is bidirectional or just from *Message* sender to *Message* recipient
1463   depends on the selected transport-security protocol.

1464

7.5.16.2 CertificateRef element

1465

1466   The EMPTY *CertificateRef* element contains an IMPLIED IDREF attribute, *certId* that
1467   identifies the certificate to be used by referring to the *Certificate* element (under *PartyInfo*) that
1468   has the matching ID attribute value. The *CertificateRef* element MUST be present if the
1469   transport-security protocol uses certificates. It MAY be omitted otherwise (e.g. if authentication
1470   is by password).

1471

7.5.16.3 Specifics for HTTP

1472

1473   For encryption with HTTP, the protocol is SSL[SSL] (Secure Socket Layer) Version 3.0, which
1474   uses public-key encryption.

1475

1476 **7.6 DocExchange Element**

1477 The *DocExchange* element provides information that the *Parties* must agree on regarding
1478 exchange of documents between them. This information includes the messaging service
1479 properties (e.g. ebXML *Message* Service[ebMS]).
1480

1481 Following is the structure of the *DocExchange* element of the *CPP*. Subsequent sections
1482 describe each child element in greater detail.
1483

```
1484        <DocExchange docExchangeId = "N06">
1485            <ebXMLBinding version = "0.92">
1486                <ReliableMessaging>  <!--cardinality 0 or 1-->
1487                ...
1488                </ReliableMessaging>
1489                <NonRepudiation>  <!--cardinality 0 or 1-->
1490                    ...
1491                </NonRepudiation>
1492                <DigitalEnvelope>  <!--cardinality 0 or 1-->
1493                    ...
1494                </DigitalEnvelope>
1495                <NamespaceSupported> <!-- 1 or more -->
1496                    ...
1497                </NamespaceSupported>
1498            </ebXMLBinding>
1499        </DocExchange>
```
1500

1501 The *DocExchange* element of the *CPP* defines the properties of the messaging service to be
1502 used with *CPAs* composed from the *CPP*.
1503

1504 The *DocExchange* element is comprised of a single *ebXMLBinding* child element.
1505

1506   NOTE: The document-exchange section can be extended to messaging services other
1507   than the ebXML *Message* service by adding additional *xxxBinding* elements and their
1508   child elements that describe the other services, where *xxx* is replaced by the name of the
1509   additional binding. An example is *XPBinding*, which might define support for the future
1510   XML Protocol specification.
1511

1512 **7.6.1 docExchangeId attribute**

1513 The *DocExchange* element has a single IMPLIED *docExchangeId* attribute that is an [XML] ID
1514 that provides a unique identifier that MAY be referenced from elsewhere within the *CPP*
1515 document.
1516

1517 **7.6.2 ebXMLBinding element**

1518 The *ebXMLBinding* element describes properties specific to the ebXML *Message*
1519 Service[ebMS]. The *ebXMLBinding* element is comprised of the following child elements:
1520   • zero or one *ReliableMessaging* element which specifies the characteristics of reliable
1521   messaging,
1522   • zero or one *NonRepudiation* element which specifies the requirements for signing the

Collaboration-Protocol Profile Tend Agreement Specification   **Page 38 of 90**

1523   *Message*,
1524   • zero or one *DigitalEnvelope* element which specifies the requirements for encryption
1525   by the digital-envelope[DIGENV] method,
1526   • zero or more *NamespaceSupported* elements that identify any namespace extensions
1527   supported by the messaging service implementation.
1528

1529   7.6.3 version attribute

1530   The *ebXMLBinding* element has a single REQUIRED *version* attribute that identifies the
1531   version of the ebXML *Message* Service specification being used.
1532

1533   7.6.4 ReliableMessaging element

1534   The *ReliableMessaging* element specifies the properties of reliable ebXML *Message* exchange.
1535   The default that applies if the *ReliableMessaging* element is omitted is "BestEffort". See
1536   Section 7.6.4.1. The following is the element structure:
1537

```
1538          <ReliableMessaging deliverySemantics="OnceAndOnlyOnce"
1539                      idempotency="false"
1540                      messageOrderSemantics="Guaranteed">
1541         <!--The triplet of elements Retries, RetryInterval, and
1542             PersistDuration has cardinality 0 or 1-->
1543            <Retries>5</Retries>
1544            <RetryInterval>60</RetryInterval> <!--time in seconds-->
1545            <PersistDuration>30S</PersistDuration>
1546          </ReliableMessaging>
```

1547
1548   The *ReliableMessaging* element is comprised of the following child elements. These elements
1549   have cardinality 0 or 1. They MUST either be all present or all absent.
1550   • a *Retries* element,
1551   • a *RetryInterval* element,
1552   • a *PersistDuration* element.
1553
1554   The *ReliableMessaging* element has attributes as follows:
1555   • a REQUIRED *deliverySemantics* attribute,
1556   • a REQUIRED *idempotency* attribute,
1557   • an IMPLIED *messageOrderSemantics* attribute.
1558

1559   7.6.4.1 deliverySemantics attribute

1560   The *deliverySemantics* attribute of the *ReliableMessaging* element specifies the degree of
1561   reliability of *Message* delivery. This attribute is an enumeration of possible values that consist
1562   of:
1563   • "OnceAndOnlyOnce",
1564   • "BestEffort".
1565
1566   A value of "OnceAndOnlyOnce" specifies that a *Message* must be delivered exactly once.
1567   "BestEffort" specifies that reliable-messaging semantics are not to be used.
1568

7.6.4.2 idempotency attribute

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

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

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

7.6.4.3 messageOrderSemantics attribute

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

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

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

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

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

7.6.4.4 Retries and RetryInterval elements

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

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

1615

7.6.4.5 PersistDuration element

1616

1617 The value of the ***PersistDuration*** element is the minimum length of time, expressed as an XML
1618 Schema[XMLSCHEMA-2] timeDuration, that data from a *Message* that is sent reliably is kept in
1619 *Persistent Storage* by an ebXML *Message*-Service implementation that receives that *Message*.

1620

7.6.5 NonRepudiation element

1621

1622 Non-repudiation both proves who sent a *Message* and prevents later repudiation of the contents
1623 of the *Message*. Non-repudiation is based on signing the *Message* using XML Digital
1624 Signature[XMLDSIG]. The element structure is as follows:

1625

```
1626        <NonRepudiation>
1627            <Protocol version="2000/10/31">http://www.w3.org/2000/09/xmldsig#
1628            </Protocol>
1629            <HashFunction>sha1</HashFunction>
1630            <SignatureAlgorithm>rsa</SignatureAlgorithm>
1631            <CertificateRef certId = "N03"/>
1632        </NonRepudiation>
```

1633

1634 If the ***NonRepudiation*** element is omitted, the *Messages* are not digitally signed.

1635

1636 Security at the document-exchange level applies to all *Messages* in both directions for *Business*
1637 *Transaction*s for which security is enabled.

1638

1639 The ***NonRepudiation*** element is comprised of the following child elements:
1640 • a REQUIRED ***Protocol*** element,
1641 • a REQUIRED ***HashFunction*** (e.g. SHA1, MD5) element,
1642 • a REQUIRED ***SignatureAlgorithm*** element,
1643 • a REQUIRED ***Certificate*** element.

1644

7.6.5.1 Protocol element

1645

1646 The REQUIRED ***Protocol*** element identifies the technology that will be used to digitally sign a
1647 *Message*. It has a single IMPLIED ***version*** attribute whose value is is a string that identifies the
1648 version of the specified technology. An example of the ***Protocol*** element follows:

1649

```
1650        <Protocol version="2000/10/31">http://www.w3.org/2000/09/xmldsig#
1651        </Protocol>
```

1652

7.6.5.2 HashFunction element

1653

1654 The REQUIRED ***HashFunction*** element identifies the algorithm that is used to compute the
1655 digest of the *Message* being signed.

1656

7.6.5.3 SignatureAlgorithm element

1657

1658 The REQUIRED ***SignatureAlgorithm*** element identifies the algorithm that is used to compute
1659 the value of the digital signature.

1660

7.6.5.4 CertificateRef element

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

7.6.6 DigitalEnvelope element

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

```
<DigitalEnvelope>
        <Protocol version = "2.0">S/MIME</Protocol>
        <EncryptionAlgorithm>rsa</EncryptionAlgorithm>
        <CertificateRef certId = "N03"/>
</DigitalEnvelope>
```

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

7.6.6.1 Protocol element

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

7.6.6.2 EncryptionAlgorithm element

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

7.6.6.3 CertificateRef element

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

7.6.7 NamespaceSupported element

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

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

7.7 Packaging element

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

Collaboration-Protocol Profile Tend Agreement Specification   **Page 42 of 90**

1706 capabilities are typically the capabilities or agreements described in this subtree. The ***Packaging***
1707 element provides information about MIME content types, XML namespaces, security
1708 parameters, and MIME structure of the data that is exchanged between *Parties*.

1709

1710 Following is an example of the ***Packaging*** element:

1711

```
1712        <Packaging id="id">
1713        <!--The Packaging triple MAY appear one or more times-->
1714               <ProcessingCapabilities parse="..."  generate="..."/>
1715               <SimplePart
1716                     id="id" mimetype="type"/> <!--one or more-->
1717                     <NamespaceSupported location = "" version="">
1718                      URI
1719                     </NamespaceSupported> <!--zero or more-->
1720                <!--The child of CompositeList is an enumeration of either
1721                Composite or Encapsulation.  The enumeration MAY appear one
1722                or more time, with the two elements intermixed-->
1723                <CompositeList>
1724                     <Composite mimetype="type"
1725                           id="name"
1726                           mimeparameters="parameter">
1727                           <Constituent idref="name"/>
1728                     </Composite>
1729                     <Encapsulation mimetype="type" id="name">
1730                           <Constituent idref="name"/>
1731                     </Encapsulation>
1732                </CompositeList>
1733        </Packaging>
```

1734

1735 See "Matching Packaging" in Appendix F   for a more specific example.

1736

1737 The ***Packaging*** element has one attribute; the REQUIRED ***id*** attribute, with type ID. It is
1738 referred to in the ***ServiceBinding*** element and in the ***Override*** element, by using the IDREF
1739 attribute, ***packageId.***

1740

1741 The child elements of the ***Packaging*** element are ***ProcessingCapabilities, SimplePart****,* and
1742 ***CompositeList.*** This set of elements MAY appear one or more times as a child of each
1743 ***Packaging*** element in a *CPP* and SHALL appear once as a child of each ***Packaging*** element in a
1744 *CPA*.

1745

1746 **7.7.1 ProcessingCapabilities element**

1747 The ***ProcessingCapabilities*** element has two REQUIRED attributes with Boolean values of
1748 either "true" or "false". The attributes are ***parse*** and ***generate.*** Normally, these attributes will
1749 both have values of "true" to indicate that the packaging constructs specified in the other child
1750 elements can be both produced as well as processed at the software *Message* service layer.
1751 At least one of the ***generate*** or ***parse*** attributes MUST be true.

1752

1753 **7.7.2 SimplePart element**

1754 The ***SimplePart*** element provides a repeatable list of the constituent parts, primarily identified by

Collaboration-Protocol Profile Tand Agreement Specification   **Page 43 of 90**

1755 the MIME content-type value. The *SimplePart* element has two REQUIRED attributes: *id* and
1756 *mimetype*. The *id* attribute, type ID, provides the value that will be used later to reference this
1757 *Message* part when specifying how the parts are packaged into composites, if composite
1758 packaging is present. The *mimetype* attribute provides the actual value of content-type for the
1759 simple *Message* part being specified.
1760

7.7.3 SimplePart element

1762 The *SimplePart* element can have zero or more *NamespaceSupported* elements. Each of these
1763 identifies any namespace extensions supported for the XML packaged in the parent simple body
1764 part. Examples include Security Services Markup Language[S2ML] and Transaction Authority
1765 Markup Language[XAML]. For example, support for the S2ML namespace would be defined as
1766 follows:
1767
1768   `<NamespaceSupported location = "http://www.s2ml.org/s2ml.xsd"`
1769   `version = "0.8">http://www.s2ml.org/s2ml</NamespaceSupported>`
1770

7.7.4 CompositeList element

1772 The final child element of *Packaging* is *CompositeList*, which is a container for the specific way
1773 in which the simple parts are combined into groups (MIME multiparts) or encapsulated within
1774 security-related MIME content-types. The *CompositeList* element MAY be omitted from
1775 *Packaging* when no security encapsulations or composite multiparts are used. When the
1776 *CompositeList* element is present, the content model for the *CompositeList* element is a
1777 repeatable sequence of choices of *Composite* or *Encapsulation* elements. The *Composite* and
1778 *Encapsulation* elements MAY appear intermixed as desired.
1779
1780 The sequence in which the choices are presented is important because, given the recursive
1781 character of MIME packaging, composites or encapsulations MAY include previously
1782 mentioned composites (or rarely, encapsulations) in addition to the *Message* parts characterized
1783 within the *SimplePart* subtree. Therefore, the "top-level" packaging will be described last in the
1784 sequence.
1785
1786 The *Composite* element has the following attributes:
1787   • a REQUIRED *mimetype* attribute,
1788   • a REQUIRED *id* attribute,
1789   • an IMPLIED *mimeparameters* attribute.
1790
1791 The *mimetype* attribute provides the value of the MIME content-type for this *Message* part, and
1792 this will be some MIME composite type, such as "multipart/related" or "multipart/signed". The
1793 *id* attribute, type ID, provides a way to refer to this composite if it needs to be mentioned as a
1794 constituent of some later element in the sequence. The *mimeparameters* attribute provides the
1795 values of any significant MIME parameter (such as "type=application/vnd.eb+xml") that is
1796 needed to understand the processing demands of the content-type.
1797
1798 The *Composite* element has one child element, *Constituent.*
1799

1800 The *Constituent* element has one REQUIRED attribute, *idref,* type IDREF, and has an EMPTY
1801 content model. The *idref* attribute has as its value the value of the *id* attribute of a previous
1802 *Composite, Encapsulation*, or *SimplePart* element. The purpose of this sequence of
1803 *Constituents* is to indicate both the contents and the order of what is packaged within the current
1804 *Composite* or *Encapsulation*.
1805
1806 The *Encapsulation* element is typically used to indicate the use of MIME security mechanisms,
1807 such as [S/MIME] or Open-PGP[RFC2015]. A security body part can encapsulate a MIME part
1808 that has been previously characaterized. For convenience, all such security structures are under
1809 the *Encapsulation* element, even when technically speaking the data is not "inside" the body
1810 part. (In other words, the so-called clear-signed or detached signature structures possible with
1811 MIME multipart/signed are for simplicity found under the *Encapsulation* element.)
1812
1813 The *Encapsulation* element has the following attributes:
1814   • a REQUIRED *mimetype* attribute,
1815   • a REQUIRED *id* attribute,
1816   • an IMPLIED *mimeparameters* attribute.
1817
1818 The *mimetype* attribute provides the value of the MIME content-type for this *Message* part, such
1819 as "application/pkcs7-mime". The *id* attribute, type ID, provides a way to refer to this
1820 encapsulation if it needs to be mentioned as a constituent of some later element in the sequence.
1821 The *mimeparameters* attribute provides the values of any significant MIME parameter(s)
1822 needed to understand the processing demands of the content-type.
1823
1824 Both the *Encapsulation* element and the *Composite* element have child elements consisting of a
1825 *Constituent* element or of a repeatable sequence of *Constituent* elements, respectively.
1826

7.8 ds:Signature element

1828 The *CPP* MAY be digitally signed using technology that conforms with the XML Digital
1829 Signature specification[XMLDSIG]. The *ds:Signature* element is the root of a subtree of
1830 elements that MAY be used for signing the *CPP*. The syntax is:
1831
1832   <ds:Signature>...</ds:Signature>
1833
1834 The content of this element and any subelements are defined by the XML Digital Signature
1835 specification. See Section 8.7 for a detailed discussion. The following additional constraints on
1836 *ds:Signature* are imposed:
1837
1838   • A *CPP* MUST be considered invalid if any *ds:Signature* element fails core validation as
1839   defined by the XML Digital Signature specification[XMLDSIG].
1840
1841   • Whenever a *CPP* is signed, each *ds:Reference* element within a *ProcessSpecification*
1842   element MUST pass reference validation and each *ds:Signature* element MUST pass
1843   core validation.
1844

1845   NOTE: In case a *CPP* is unsigned, software MAY nonetheless validate the ***ds:Reference***
1846   elements within ***ProcessSpecification*** elements and report any exceptions.
1847
1848   NOTE: Software for creation of *CPPs* and *CPAs* MAY recognize ***ds:Signature*** and
1849   automatically insert the element structure necessary to define signing of the *CPP* and *CPA*.
1850   Signature creation itself is a cryptographic process that is outside the scope of this
1851   specification.
1852
1853   NOTE: See non-normative note in Section 7.5.4.5 for a discussion of times at which validity
1854   tests MAY be made.
1855

1856   7.9 Comment Element

1857   The ***CollaborationProtocolProfile*** element MAY contain zero or more ***Comment*** elements. The
1858   ***Comment*** element is a textual note that MAY be added to serve any purpose the author desires.
1859   The language of the ***Comment*** is identified by a REQUIRED ***xml:lang*** attribute. The ***xml:lang***
1860   attribute MUST comply with the rules for identifying languages specified in [XML]. If multiple
1861   ***Comment*** elements are present, each MAY have a different ***xml:lang*** attribute value. An
1862   example of a ***Comment*** element follows:
1863
1864   `<Comment xml:lang="en-gb">yadda yadda, blah blah</Comment>`
1865
1866   When a *CPA* is composed from two *CPPs*, all ***Comment*** elements from both *CPPs* SHALL be
1867   included in the *CPA* unless the two *Parties* agree otherwise.

1868 # 8 CPA Definition

1869 A *Collaboration-Protocol Agreement (CPA)* defines the capabilities that two *Parties* must agree
1870 upon to enable them to engage in electronic *Business* for the purposes of the particular *CPA*. This
1871 section defines and discusses the details of the *CPA*. The discussion is illustrated with some
1872 XML fragments.
1873
1874 Most of the XML elements in this section are described in detail in section 7, "CPP Definition".
1875 In general, this section does not repeat that information. The discussions in this section are
1876 limited to those elements that are not in the *CPP* or for which additional discussion is required in
1877 the *CPA* context. See also Appendix C and Appendix D for the DTD and XML Schema,
1878 respectively, and Appendix B for an example of a *CPA* document.
1879

1880 ## 8.1 CPA Structure

1881 Following is the overall structure of the *CPA:*
1882

```
1883 <CollaborationProtocolAgreement
1884       xmlns="http://www.ebxml.org/namespaces/tradePartner"
1885       xmlns:bpm="http://www.ebxml.org/namespaces/businessProcess"
1886       xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
1887       xmlns:xlink = "http://www.w3.org/1999/xlink"
1888       cpaid="YoursAndMyCPA"
1889       version="1.2">
1890       <Status value = "proposed"/>
1891       <Start>1988-04-07T18:39:09</Start>
1892       <End>1990-04-07T18:40:00</End>
1893       <!--ConversationConstraints MAY appear 0 or 1 times-->
1894       <ConversationConstraints invocationLimit = "100"
1895             concurrentConversations = "4"/>
1896       <PartyInfo>
1897             …
1898       </PartyInfo>
1899       <PartyInfo>
1900             …
1901       </PartyInfo>
1902       <Packaging id="N20"> <!--one or more-->
1903             ...
1904       </Packaging>
1905       <!--ds:signature MAY appear 0 or more times-->
1906       <ds:Signature>any combination of text and elements
1907       </ds:Signature>
1908       <Comment xml:lang="en-gb">any text</Comment> <!--zero or more-->
1909 </CollaborationProtocolAgreement>
```
1910

1911 ## 8.2 CollaborationProtocolAgreement Element

1912 The *CollaborationProtocolAgreement* element is the root element of a *CPA*. It has a
1913 REQUIRED *cpaid* attribute of type [XML] CDATA that supplies a unique idenfier for the

1914   document. The value of the **cpaid** attribute SHALL be assigned by one *Party* and used by both.
1915   It is RECOMMENDED that the value of the **cpaid** attribute be a URI. The value of the **cpaid**
1916   attribute MAY be used as the value of the **CPAId** element in the ebXML *Message*
1917   *Header*[ebMS] or of a similar element in a *Message Header* of an alternative messaging service.
1918
1919   NOTE: Each *Party* MAY associate a local identifier with the **cpaid** attribute.
1920
1921   In addition, the **CollaborationProtocolAgreement** element has an IMPLIED **version** attribute.
1922   This attribute indicates the version of the *CPA*. Its purpose is to provide versioning capabilities
1923   for an instance of a *CPA* as it undergoes negotiation between the two parties. The **version**
1924   attribute SHOULD also be used to provide versioning capability for a *CPA* that has been
1925   deployed and then modified. The value of the **version** attribute SHOULD be a string
1926   representation of a numeric value such as "1.0" or "2.3". The value of the version string
1927   SHOULD be changed with each change made to the *CPA* document both during negotiation and
1928   after it has been deployed.
1929
1930   NOTE: The method of assigning version identifiers is left to the implementation.
1931
1932   The **CollaborationProtocolAgreement** element has REQUIRED [XML] Namespace[XMLNS]
1933   declarations that are defined in Section 7, "CPP Definition".
1934
1935   The **CollaborationProtocolAgreement** element is comprised of the following child elements,
1936   each of which is described in greater detail in subsequent sections:
1937   • a REQUIRED **Status** element that identifies the state of the process that creates the
1938   *CPA,*
1939   • a REQUIRED **Start** element that records the date and time that the *CPA* goes into
1940   effect,
1941   • a REQUIRED **End** element that records the date and time after which the *CPA* must
1942   be renegotiated by the *Parties,*
1943   • zero or one **ConversationConstraints** element that documents certain agreements
1944   about conversation processing,
1945   • two REQUIRED **PartyInfo** elements, one for each *Party* to the *CPA,*
1946   • one or more **ds:Signature** elements that provide signing of the *CPA* using the XML
1947   Digital Signature[XMLDSIG] standard.
1948

8.3 Status Element

1950   The **Status** element records the state of the composition/negotiation process that creates the *CPA*.
1951   An example of the **Status** element follows:
1952
1953   ```
        <Status value = "proposed"/>
        ```
1954
1955   The Status element has a REQUIRED **value** attribute that records the current state of
1956   composition of the *CPA*. This attribute is an enumeration comprised of the following possible
1957   values:
1958        • "proposed", meaning that the *CPA* is still being negotiated by the *Parties,*

1959     •    "agreed", meaning that the contents of the *CPA* have been agreed to by both *Parties,*

1960     •    "signed", meaning that the *CPA* has been "signed" by the *Parties.* This "signing"
1961         MAY take the form of a digital signature that is described in section 8.7 below.

1962

1963      NOTE: The **Status** element MAY be used by a *CPA* composition and negotiation tool to
1964      assist it in the process of building a *CPA*.

1965

1966 ## 8.4 CPA Lifetime

1967 The lifetime of the *CPA* is given by the **Start** and **End** elements.  The syntax is:

1968

1969      `<Start>1988-04-07T18:39:09</Start>`
1970      `<End>1990-04-07T18:40:00</End>`

1971

1972 ### 8.4.1 Start element

1973 The **Start** element specifies the starting date and time of the *CPA*. The **Start** element SHALL be
1974 a string value that conforms to the content model of a canonical timeInstant as defined in the
1975 XML Schema Datatypes Specification[XMLSCHEMA-2].  For example, to indicate 1:20 pm
1976 UTC (Coordinated Universal Time) on May 31, 1999, a **Start** element would have the following
1977 value:

1978

1979      `1999-05-31T13:20:00Z`

1980

1981 The **Start** element SHALL be represented as Coordinated Universal Time (UTC).

1982

1983 ### 8.4.2 End element

1984 The **End** element specifies the ending date and time of the *CPA*. The **End** element SHALL be a
1985 string value that conforms to the content model of a canonical timeInstant as defined in the XML
1986 Schema Datatypes Specification[XMLSCHEMA-2].  For example, to indicate 1:20 pm UTC
1987 (Coordinated Universal Time) on May 31, 1999, an **End** element would have the following
1988 value:

1989

1990      `1999-05-31T13:20:00Z`

1991

1992 The **End** element SHALL be represented as Coordinated Universal Time (UTC).

1993

1994 When the end of the *CPA's* lifetime is reached, any *Business Transaction*s that are still in
1995 progress SHALL be allowed to complete and no new *Business Transactions* SHALL be started.
1996 When all in-progress *Business Transactions* on each conversation are completed, the
1997 *Conversation* shall be terminated whether or not it was completed.

1998

1999      NOTE: It should be understood that if a *Business* application defines a conversation as
2000      consisting of multiple *Business Transactions*, such a conversation MAY be terminated
2001      with no error indication when the end of the lifetime is reached. The run-time system
2002      could provide an error indication to the application.

**Collaboration-Protocol Profile Tand Agreement Specification**                     **Page 49 of 90**

2003
2004        NOTE: It should be understood that it MAY not be feasible to wait for outstanding
2005        conversations to terminate before ending the *CPA* since there is no limit on how long a
2006        conversation MAY last.

2007
2008        NOTE:  The run-time system SHOULD return an error indication to both *Parties* when a
2009        new *Business Transaction* is started under this *CPA* after the date and time specified in
2010        the **End** element.

2011

2012    **8.5 ConversationConstraints Element**

2013

2014    The ***ConversationConstraints*** element places limits on the number of conversations under the
2015    *CPA*. An example of this element follows:

2016
2017            <ConversationConstraints invocationLimit = "100"
2018                                    concurrentConversations = "4"/>

2019
2020    The  ***ConversationConstraints*** element has the following attributes:
2021        • an IMPLIED ***invocationLimit*** attribute,
2022        • an IMPLIED ***concurrentConversations*** attribute.

2023

2024    **8.5.1 invocationLimit attribute**

2025    The ***invocationLimit*** attribute defines the maximum number of conversations that can be
2026    processed under the *CPA*.  When this number has been reached, the *CPA* is terminated and must
2027    be renegotiated. If no value is specified, there is no upper limit on the number of conversations
2028    and the lifetime of the *CPA* is controlled solely by the **End** element.

2029
2030        NOTE: The ***invocationLimit*** attribute sets a limit on the number of units of *Business* that
2031        can be performed under the *CPA*. It is a *Business* parameter, not a performance
2032        parameter.

2033

2034    **8.5.2 concurrentConversations attribute**

2035    The ***concurrentConversations*** attribute defines the maximum number of conversations that can
2036    be in process under this *CPA* at the same time. If no value is specified, processing of concurrent
2037    conversations is strictly a local matter.

2038
2039        NOTE: The ***concurrentConversations*** attribute provides a parameter for the *Parties* to use
2040        when it is necessary to limit the number of conversations that can be concurrently processed
2041        under a particular *CPA*. For example, the back-end process might only support a limited
2042        number of concurrent conversations. If a request for a new conversation is received when
2043        the maximum number of conversations allowed under this *CPA* is already in process, an
2044        implementation MAY reject the new conversation or MAY enqueue the request until an
2045        existing conversation ends. If no value is given for ***concurrentConversations***, how to handle
2046        a request for a new conversation for which there is no capacity is a local implementation

**Collaboration-Protocol Profile Tend Agreement Specification**                    **Page 50 of 90**

2047    matter.
2048

## 8.6 PartyInfo Element

2050    The general characteristics of the *PartyInfo* element are discussed in section 7.5.
2051

2052    The *CPA* SHALL have one *PartyInfo* element for each *Party* to the *CPA*.  The *PartyInfo*
2053    element specifies the *Parties'* agreed terms for engaging in the *Business Collaborations* defined
2054    by the *Process-Specification* documents referenced by the *CPA*. If a *CPP* has more than one
2055    *PartyInfo* element, the appropriate *PartyInfo* element SHALL be selected from each *CPP* when
2056    composing a *CPA*.
2057

2058    In the *CPA*, there SHALL be one *PartyId* element under each *PartyInfo* element. The value of
2059    this element is the same as the value of the *PartyId* element in the ebXML *Message* Service
2060    specification[ebMS] or similar messaging service specification. One *PartyId* element SHALL be
2061    used within a *To* or *From Header* element of an ebXML *Message*.
2062

### 8.6.1 ProcessSpecification element

2064    The  *ProcessSpecification*  element  identifies the *Business Collaboration* that the two *Parties*
2065    have agreed to perform.  There MAY be one or more *ProcessSpecification* elements in a *CPA*.
2066    Each SHALL be a child element of a separate *CollaborationRole* element. See the discussion in
2067    Section 7.5.3.
2068

## 8.7 ds:Signature Element

2070    A *CPA* document MAY be digitally signed by one or more of the *Parties* as a means of ensuring
2071    its integrity as well as a means of expressing the agreement just as a corporate officer's signature
2072    would do for a paper document. If signatures are being used to digitally sign an ebXML *CPA* or
2073    *CPP* document, then it is strongly RECOMMENDED that [XMLDSIG] be used to digitally sign
2074    the document. The *ds:Signature* element is the root of a subtree of elements that MAY be used
2075    for signing the *CPP*. The syntax is:
2076

2077    ```
        <ds:Signature>...</ds:Signature>
        ```
2078

2079   The content of this element and any subelements are defined by the XML Digital Signature
2080   specification[XMLDSIG]. The following additional constraints on *ds:Signature* are imposed:
2081

2082   •   A *CPA* MUST be considered invalid if any *ds:Signature* fails core validation as defined
2083   by the XML Digital Signature specification.
2084

2085   •   Whenever a *CPA* is signed, each *ds:Reference* within a *ProcessSpecification* MUST
2086   pass reference validation and each *ds:Signature* MUST pass core validation.
2087

2088   NOTE: In case a *CPA* is unsigned, software MAY nonetheless validate the *ds:Reference*
2089   elements within *ProcessSpecification* elements and report any exceptions.
2090

2091   NOTE: Software for creation of *CPPs* and *CPAs* MAY recognize *ds:Signature* and
2092   automatically insert the element structure necessary to define signing of the *CPP* and *CPA*.
2093   Signature creation itself is a cryptographic process that is outside the scope of this
2094   specification.
2095
2096   NOTE: See non-normative note in section 7.5.4.5 for a discussion of times at which a *CPA*
2097   MAY be validated.
2098

8.7.1 Persistent Digital Signature

2100   If [XMLDSIG] is used to sign an ebXML *CPP* or *CPA,* the process defined in this section of the
2101   specification SHALL be used.
2102

8.7.1.1 Signature Generation
2104   Following are the steps to create a digital signature:
2105   1.   Create a *SignedInfo* element, a child element of *ds:Signature. SignedInfo* SHALL have
2106   child elements *SignatureMethod*, *CanonicalizationMethod*, and *Reference* as prescribed by
2107   [XMLDSIG].
2108   2.   Canonicalize and then calculate the **SignatureValue** over *SignedInfo* based on algorithms
2109   specified in *SignedInfo* as specified in [XMLDSIG].
2110   3.   Construct the *Signature* element that includes the *SignedInfo*, *KeyInfo*
2111   (RECOMMENDED), and *SignatureValue* elements as specified in [XMLDSIG].
2112   4.   Include the namespace qualified *Signature* element in the document just signed, following
2113   the last *PartyInfo* element.
2114

8.7.1.2 ds:SignedInfo element
2116   The *ds:SignedInfo* element SHALL be comprised of zero or one *ds:CanonicalizationMethod*
2117   element, the *ds:SignatureMethod* element, and one or more *ds:Reference* elements.
2118

8.7.1.3 ds:CanonicalizationMethod element
2120   The *ds:CanonicalizationMethod* element is defined as OPTIONAL in [XMLDSIG], meaning
2121   that the element need not appear in an instance of a *ds:SignedInfo* element. The default
2122   canonicalization method that is applied to the data to be signed is [XMLC14N] in the absence of
2123   a *ds:CanonicalizationMethod* element that specifies otherwise. This default SHALL also serve
2124   as the default canonicalization method for the ebXML *CPP* and *CPA* documents.
2125

8.7.1.4 ds:SignatureMethod element
2127   The *ds:SignatureMethod* element SHALL be present and SHALL have an *Algorithm* attribute.
2128   The RECOMMENDED value for the *Algorithm* attribute is:
2129
2130   http://www.w3.org/2000/09/xmldsig#dsa-sha1
2131
2132   This RECOMMENDED value SHALL be supported by all compliant ebXML *CPP* or *CPA*
2133   software implementations.
2134

8.7.1.5 ds:Reference element

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

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

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

8.7.1.6 ds:Transform element

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

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

For example:
```
<ds:Reference ds:URI="">
      <ds:Transforms>
            <ds:Transform
    ds:Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature "/>
      </ds:Transforms>
      <ds:DigestMethod
     ds:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <ds:DigestValue>...</ds:DigestValue>
</ds:Reference>
```

8.7.1.7 ds:Algorithm element

The *ds:Transform* element SHALL include a ds*Algorithm* attribute that has a value of:
```
http://www.w3.org/2000/09/xmldsig#enveloped-signature
```

> NOTE: When digitally signing a *CPA*, it is RECOMMENDED that each *Party* sign the document in accordance with the process described above. The first *Party* that signs the *CPA* will sign only the *CPA* contents, excluding their own signature. The second *Party* signs 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 sign over both signatures.

8.8 Comment element

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

8.9 Composing a CPA from Two CPPs

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

8.9.1 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.

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

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

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

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

- The CPA composition tool can create a separate copy of the Process-Specification document. The tool can then directly modify the *Process-Specification* document with information from the *CPA*. One advantage of this method is that the override process is performed entirely by the *CPA* composition tool. A second advantage is that with a separate copy of the *Process-Specification* document associated with the particular *CPA*, there is no exposure to modifications of the *Process-Specification* document between the time that the *CPA* is created and the time it is installed in the *Parties'* systems.
- A *CPA* installation tool can dynamically override parameters in the *Process-Specification* document using information from the corresponding parameters in the *CPA* at the time the *CPA* and *Process-Specification* document are installed in the *Parties'* systems. This eliminates the need to create a separate copy of the *Process-Specification* document.

2225   • Other possible methods might be based on XSLT transformations of the parameter
2226   information in the *CPA* and/or the *Process-Specification* document.

2227 9 References

2228 Some references listed below specify functions for which specific XML definitions are provided
2229 in the *CPP* and *CPA*. Other specifications are referred to in this specification in the sense that
2230 they are represented by keywords for which the *Parties* to the *CPA* MAY obtain plug-ins or
2231 write custom support software but do not require specific XML element sets in the *CPP* and
2232 *CPA*.
2233
2234 In a few cases, the only available specification for a function is a proprietary specification.
2235 These are indicated by notes within the citations below.
2236
2237 [ccOVER] ebXML Core Components and Business Process Document Overview,
2238 http://www.ebxml.org.
2239
2240 [DIGENV] Digital Envelope, RSA Laboratories, http://www.rsasecurity.com/rsalabs/. NOTE:
2241 At this time, the only available specification for digital envelope appears to be the RSA
2242 Laboratories specification.
2243
2244 [ebBPSS] ebXML Business Process Specification Schema, http://www.ebxml.org.
2245
2246 [ebGLOSS] ebXML Glossary, http://www.ebxml.org.
2247
2248 [ebMS] ebXML Message Service Specification, http://www.ebxml.org.
2249
2250 [ebRS] ebXML Registry Services Specification, http://www.ebxml.org.
2251
2252 [ebTA] ebXML Technical Architecture Specification, http://www.ebxml.org.
2253
2254 [HTTP] Hypertext Transfer Protocol, Internet Engineering Task Force RFC2616.
2255
2256 [IPSEC] IP Security Document Roadmap, Internet Engineering Task Force RFC 2411.
2257
2258 [ISO6523] Structure for the Identification of Organizations and Organization Parts, International
2259 Standards Organization ISO-6523.
2260
2261 [MIME] MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying
2262 and Describing the Format of Internet *Message* Bodies. Internet Engineering Task Force RFC
2263 1521.
2264
2265 [RFC822] Standard for the Format of ARPA Internet Text Messages, Internet Engineering Task
2266 Force RFC 822.
2267
2268 [RFC959] File Transfer Protocol (FTP), Internet Engineering Task Force RFC 959.
2269
2270 [RFC1123] Requirements for Internet Hosts -- Application and Support, R. Braden, Internet

2271   Engineering Task Force, October 1989.

2272

2273   [RFC1579] Firewall-Friendly FTP, S. Bellovin, Internet Engineering Task Force, February 1994.

2274

2275   [RFC2015] MIME Security with Pretty Good Privacy, M. Elkins, Internet Engineering Task
2276   Force, RFC 2015.

2277

2278   [RFC2119] Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering
2279   Task Force RFC 2119.

2280

2281   [RFC2396] Uniform Resource Identifiers (URI): Generic Syntax; T. Berners-Lee, R. Fielding, L.
2282   Masinter - August 1998.

2283

2284   [S/MIME] S/MIME Version 3 Message Specification, Internet Engineering Task Force RFC
2285   2633.

2286

2287   [S2ML] Security Services Markup Language, http://s2ml.org/.

2288

2289   [SMTP] Simple Mail Transfer Protocol, Internet Engineering Task Force RFC 821.

2290

2291   [SSL] Secure Sockets Layer, Netscape Communications Corp. http://developer.netscape.com.
2292   NOTE: At this time, it appears that the Netscape specification is the only available specification
2293   of SSL. Work is in progress in IETF on "Transport Layer Security", which is intended as a
2294   replacement for SSL.

2295

2296   [XAML] Transaction Authority Markup Language, http://xaml.org/.

2297

2298   [XLINK] XML Linking Language, http://www.w3.org/TR/xlink/.

2299

2300   [XML] Extensible Markup Language (XML), World Wide Web Consortium,
2301   http://www.w3.org.

2302

2303   [XMLC14N] Canonical XML, Ver. 1.0, http://www.w3.org/TR/XML-C14N/.

2304

2305   [XMLDSIG] XML Signature Syntax and Processing, Worldwide Web Consortium,
2306   http://www.w3.org/TR/xmldsig-core/.

2307

2308   [XMLNS] Namespaces in XML, T. Bray, D. Hollander, and A. Layman, Jan. 1999,
2309   http://www.w3.org/TR/REC-xml-names/.

2310

2311   [XMLSCHEMA-1] XML Schema Part 1: Structures, http://www/w3/org/TR/xmlschema-1/.

2312

2313   [XMLSCHEMA-2] XML Schema Part 2: Datatypes,
2314   http://www.w3.org/TR/xmlschema-2/.

2315

2316   [XPOINTER] XML Pointer Language, ver. 1.0, http://www.w3.org/TR/xptr.

2317   10 Conformance

2318 In order to conform to this specification, an implementation:

2319   a) SHALL support all the functional and interface requirements defined in this specification,

2320   b) SHALL NOT specify any requirements that would contradict or cause non-conformance
2321   to this specification.

2322

2323 A conforming implementation SHALL satisfy the conformance requirements of the applicable
2324 parts of this specification.

2325

2326 An implementation of a tool or service that creates or maintains ebXML *CPP* or *CPA* instance
2327 documents SHALL be determined to be conformant by validation of the *CPP* or *CPA* instance
2328 documents, created or modified by said tool or service, against the XML
2329 Schema[XMLSCHEMA-1] definition of the *CPP* or *CPA* in Appendix D and available from

2330

2331   `http://www.ebxml.org/schemas/cpp-cpa-v1_0.xsd`

2332

2333 by using two or more validating XML Schema parsers that conform to the W3C XML Schema
2334 specifications[XMLSCHEMA-1,XMLSCHEMA-2].

2335

2336 The objective of conformance testing is to determine whether an implementation being tested
2337 conforms to the requirements stated in this specification. Conformance testing enables vendors to
2338 implement compatible and interoperable systems. Implementations and applications SHALL be
2339 tested using available test suites to verify their conformance to this specification.

2340

2341 Publicly available test suites from vendor neutral organizations such as OASIS and the U.S.A.
2342 National Institute of Science and Technology (NIST) SHOULD be used to verify the
2343 conformance of implementations, applications, and components claiming conformance to this
2344 specification. Open-source reference implementations MAY be available to allow vendors to test
2345 their products for interface compatibility, conformance, and interoperability.

2346
2347
2348
2349

11 Disclaimer

2350

2351 The views and specification expressed in this document are those of the authors and are not
2352 necessarily those of their employers. The authors and their employers specifically disclaim
2353 responsibility for any problems arising from correct or incorrect implementation or use of this
2354 design.

12 Contact Information

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

Chris Ferris
XML Technology Development
Sun Microsystems, Inc
One Network Drive
Burlington, Ma 01824-0903
USA
Phone: 781-442-3063
email: chris.ferris@east.sun.com

Dale W. Moberg
Cyclone Commerce
17767 North Perimeter Dr., Suite 103
Scottsdale, AZ 85255
USA
Phone: 480-627-1800
email: dmoberg@columbus.rr.com

Tony Weida
Edifecs
2310 130th Ave. NE, Suite 100
Bellevue, WA 98005
USA
Phone: 212-678-5265
email: TonyW@edifecs.com

2388 Copyright Statement

2389
2390 Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved.

2391
2392 This document and translations of it MAY be copied and furnished to others, and derivative
2393 works that comment on or otherwise explain it or assist in its implementation MAY be prepared,
2394 copied, published and distributed, in whole or in part, without restriction of any kind, provided
2395 that the above copyright notice and this paragraph are included on all such copies and derivative
2396 works. However, this document itself MAY not be modified in any way, such as by removing
2397 the copyright notice or references to ebXML, UN/CEFACT, or OASIS, except as required to
2398 translate it into languages other than English.

2399
2400 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
2401 successors or assigns.

2402
2403 This document and the information contained herein is provided on an "AS IS" basis and
2404 ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
2405 NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN
2406 WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
2407 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
2408

2409 **Appendix A** Example of CPP Document (Non-Normative)

2410 This example is available as an ASCII file at
2411   http://ebxml.org/project_teams/trade_partner/cpp-example.xml
2412
```
2413   <?xml version="1.0" encoding="UTF-8"?>
2414   <tp:CollaborationProtocolProfile
2415         xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"
2416         xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2417         xmlns:xlink="http://www.w3.org/1999/xlink"
2418         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2419         xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
2420   http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.xsd"
2421         tp:version="1.1">
2422         <tp:PartyInfo>
2423                 <tp:PartyId tp:type="DUNS">123456789</tp:PartyId>
2424                 <tp:PartyRef tp:href="http://example.com/about.html"/>
2425                 <tp:CollaborationRole tp:id="N00">
2426                         <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
2427   xlink:type="simple" xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
2428                         <tp:Role tp:name="buyer" xlink:type="simple"
2429   xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>
2430                         <tp:CertificateRef tp:certId="N03"/>
2431                         <tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402">
2432                                 <tp:Service
2433   tp:type="uriReference">uri:example.com/services/buyerService</tp:Service>
2434
2435                                 <tp:Override tp:action="orderConfirm" tp:channelId="N07"
2436   tp:packageId="N0402" xlink:href="http://ebxml.org/processes/buySell.xml#orderConfirm"
2437   xlink:type="simple"/>
2438                         </tp:ServiceBinding>
2439                 </tp:CollaborationRole>
2440                 <tp:Certificate tp:certId="N03">
2441                         <ds:KeyInfo/>
2442                 </tp:Certificate>
2443                 <tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
2444   tp:docExchangeId="N06">
2445                         <tp:Characteristics tp:syncReplyMode="none"
2446   tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
2447   tp:secureTransport="true" tp:confidentiality="true" tp:authenticated="true"
2448   tp:authorized="false"/>
2449                 </tp:DeliveryChannel>
2450                 <tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
2451   tp:docExchangeId="N06">
2452                         <tp:Characteristics tp:syncReplyMode="none"
2453   tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
2454   tp:secureTransport="false" tp:confidentiality="true" tp:authenticated="true"
2455   tp:authorized="false"/>
2456                 </tp:DeliveryChannel>
2457                 <tp:Transport tp:transportId="N05">
2458                         <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2459                         <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
2460                         <tp:Endpoint
2461   tp:uri="https://www.example.com/servlets/ebxmlhandler" tp:type="allPurpose"/>
2462                         <tp:TransportSecurity>
2463                                 <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
2464                                 <tp:CertificateRef tp:certId="N03"/>
2465                         </tp:TransportSecurity>
2466                 </tp:Transport>
2467                 <tp:Transport tp:transportId="N08">
```

```
2468                        <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2469                        <tp:ReceivingProtocol tp:version="1.1">SMTP</tp:ReceivingProtocol>
2470                        <tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
2471    tp:type="allPurpose"/>
2472                </tp:Transport>
2473                <tp:DocExchange tp:docExchangeId="N06">
2474                        <tp:ebXMLBinding tp:version="0.98b">
2475                                <tp:ReliableMessaging
2476    tp:deliverySemantics="OnceAndOnlyOnce" tp:idempotency="true"
2477    tp:messageOrderSemantics="Guaranteed">
2478                                        <tp:Retries>5</tp:Retries>
2479                                        <tp:RetryInterval>30</tp:RetryInterval>
2480                                        <tp:PersistDuration>P1D</tp:PersistDuration>
2481                                </tp:ReliableMessaging>
2482                                <tp:NonRepudiation>
2483
2484        <tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
2485
2486        <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
2487
2488        <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
2489    sha1</tp:SignatureAlgorithm>
2490                                        <tp:CertificateRef tp:certId="N03"/>
2491                                </tp:NonRepudiation>
2492                                <tp:DigitalEnvelope>
2493                                        <tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
2494                                        <tp:EncryptionAlgorithm>DES-
2495    CBC</tp:EncryptionAlgorithm>
2496                                        <tp:CertificateRef tp:certId="N03"/>
2497                                </tp:DigitalEnvelope>
2498                        </tp:ebXMLBinding>
2499                </tp:DocExchange>
2500        </tp:PartyInfo>
2501        <tp:Packaging tp:id="N0402">
2502                <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>
2503                <tp:SimplePart tp:id="N40" tp:mimetype="text/xml">
2504                        <tp:NamespaceSupported
2505    tp:location="http://ebxml.org/project_teams/transport/messageService.xsd"
        tp:version="0.98b">http://wwNNNc5e 36 u9ng
```

Appendix B Example of CPA Document (Non-Normative)

2529

2530 The example in this appendix is to be parsed with an XML Schema parser. It is available as an
2531 ASCII file at

2532   http://ebxml.org/project_teams/trade_partner/cpa-example.xml

2533

2534 An example that can be parsed with the DTD is available at:

2535   http://ebxml.org/project_teams/trade_partner/cpa-example-dtd.xml

2536

2537   NOTE: Two separate examples of the CPA are needed because at least some existing tools

2538   require the DTD to have a `<!DOCTYPE...>` to assign the DTD and not to have a

2539   namespace qualifier.

2540

```
2541  <?xml version="1.0"?>
2542  <!-- edited with XML Spy v3.5 (http://www.xmlspy.com) by christopher ferris (sun
2543  microsystems, inc) -->
2544  <tp:CollaborationProtocolAgreement
2545        xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"
2546        xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2547        xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
2548  http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.xsd"
2549        xmlns:xlink="http://www.w3.org/1999/xlink"
2550        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2551        tp:cpaid="uri:yoursandmycpa"
2552        tp:version="1.2">
2553        <tp:Status tp:value="proposed"/>
2554        <tp:Start>2001-05-20T07:21:00Z</tp:Start>
2555        <tp:End>2002-05-20T07:21:00Z</tp:End>
2556        <tp:ConversationConstraints tp:invocationLimit="100"
2557  tp:concurrentConversations="100"/>
2558        <tp:PartyInfo>
2559              <tp:PartyId tp:type="DUNS">123456789</tp:PartyId>
2560              <tp:PartyRef xlink:href="http://example.com/about.html"/>
2561              <tp:CollaborationRole tp:id="N00">
2562                    <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
2563  xlink:type="simple" xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
2564                    <tp:Role tp:name="buyer" xlink:type="simple"
2565  xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>
2566                    <tp:CertificateRef tp:certId="N03"/>
2567                    <tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402">
2568                          <tp:Service
2569  tp:type="uriReference">uri:example.com/services/buyerService</tp:Service>
2570                          <tp:Override tp:action="orderConfirm" tp:channelId="N08"
2571  tp:packageId="N0402" xlink:href="http://ebxml.org/processes/buySell.xml#orderConfirm"
2572  xlink:type="simple"/>
2573                    </tp:ServiceBinding>
2574              </tp:CollaborationRole>
2575              <tp:Certificate tp:certId="N03">
2576                    <ds:KeyInfo/>
2577              </tp:Certificate>
2578              <tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
2579  tp:docExchangeId="N06">
2580                    <tp:Characteristics tp:syncReplyMode="none"
2581  tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
2582  tp:secureTransport="true" tp:confidentiality="true" tp:authenticated="true"
2583  tp:authorized="false"/>
2584              </tp:DeliveryChannel>
```

```
2585                    <tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
2586    tp:docExchangeId="N06">
2587                            <tp:Characteristics tp:syncReplyMode="none"
2588    tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
2589    tp:secureTransport="false" tp:confidentiality="true" tp:authenticated="true"
2590    tp:authorized="false"/>
2591                    </tp:DeliveryChannel>
2592                    <tp:Transport tp:transportId="N05">
2593                            <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2594                            <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
2595                            <tp:Endpoint
2596    tp:uri="https://www.example.com/servlets/ebxmlhandler" tp:type="allPurpose"/>
2597                            <tp:TransportSecurity>
2598                                    <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
2599                                    <tp:CertificateRef tp:certId="N03"/>
2600                            </tp:TransportSecurity>
2601                    </tp:Transport>
2602                    <tp:Transport tp:transportId="N18">
2603                            <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2604                            <tp:ReceivingProtocol tp:version="1.1">SMTP</tp:ReceivingProtocol>
2605                            <tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
2606    tp:type="allPurpose"/>
2607                    </tp:Transport>
2608                    <tp:DocExchange tp:docExchangeId="N06">
2609                            <tp:ebXMLBinding tp:version="0.98b">
2610                                    <tp:ReliableMessaging
2611    tp:deliverySemantics="OnceAndOnlyOnce" tp:idempotency="true"
2612    tp:messageOrderSemantics="Guaranteed">
2613                                            <tp:Retries>5</tp:Retries>
2614                                            <tp:RetryInterval>30</tp:RetryInterval>
2615                                            <tp:PersistDuration>P1D</tp:PersistDuration>
2616                                    </tp:ReliableMessaging>
2617                                    <tp:NonRepudiation>
2618
2619        <tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
2620
2621        <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
2622
2623        <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
2624    sha1</tp:SignatureAlgorithm>
2625                                            <tp:CertificateRef tp:certId="N03"/>
2626                                    </tp:NonRepudiation>
2627                                    <tp:DigitalEnvelope>
2628                                            <tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
2629                                            <tp:EncryptionAlgorithm>DES-
2630    CBC</tp:EncryptionAlgorithm>
2631                                            <tp:CertificateRef tp:certId="N03"/>
2632                                    </tp:DigitalEnvelope>
2633                            </tp:ebXMLBinding>
2634                    </tp:DocExchange>
2635        </tp:PartyInfo>
2636        <tp:PartyInfo>
2637                    <tp:PartyId tp:type="DUNS">987654321</tp:PartyId>
2638                    <tp:PartyRef xlink:type="simple" xlink:href="http://contrived-
2639    example.com/about.html"/>
2640                    <tp:CollaborationRole tp:id="N30">
2641                            <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
2642    xlink:type="simple" xlink:href="http://www.ebxml.org/processes/buySell.xml"/>
2643                            <tp:Role tp:name="seller" xlink:type="simple"
2644    xlink:href="http://ebxml.org/processes/buySell.xml#seller"/>
2645                            <tp:CertificateRef tp:certId="N33"/>
2646                            <tp:ServiceBinding tp:channelId="N34" tp:packageId="N0402">
2647                                    <tp:Service
```

```
2648  tp:type="uriReference">uri:example.com/services/sellerService</tp:Service>
2649                    </tp:ServiceBinding>
2650              </tp:CollaborationRole>
2651              <tp:Certificate tp:certId="N33">
2652                    <ds:KeyInfo/>
2653              </tp:Certificate>
2654              <tp:DeliveryChannel tp:channelId="N34" tp:transportId="N35"
2655  tp:docExchangeId="N36">
2656                    <tp:Characteristics tp:nonrepudiationOfOrigin="true"
2657  tp:nonrepudiationOfReceipt="false" tp:secureTransport="true" tp:confidentiality="true"
2658  tp:authenticated="true" tp:authorized="false"/>
2659              </tp:DeliveryChannel>
2660              <tp:Transport tp:transportId="N35">
2661                    <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2662                    <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
2663                    <tp:Endpoint tp:uri="https://www.contrived-
2664  example.com/servlets/ebxmlhandler" tp:type="allPurpose"/>
2665                    <tp:TransportSecurity>
2666                          <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
2667                          <tp:CertificateRef tp:certId="N33"/>
2668                    </tp:TransportSecurity>
2669              </tp:Transport>
2670              <tp:DocExchange tp:docExchangeId="N36">
2671                    <tp:ebXMLBinding tp:version="0.98b">
2672                          <tp:ReliableMessaging
2673  tp:deliverySemantics="OnceAndOnlyOnce" tp:idempotency="true"
2674  tp:messageOrderSemantics="Guaranteed">
2675                                <tp:Retries>5</tp:Retries>
2676                                <tp:RetryInterval>30</tp:RetryInterval>
2677                                <tp:PersistDuration>P1D</tp:PersistDuration>
2678                          </tp:ReliableMessaging>
2679                          <tp:NonRepudiation>
2680
2681        <tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
2682
2683        <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunction>
2684
2685        <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
2686  sha1</tp:SignatureAlgorithm>
2687                                <tp:CertificateRef tp:certId="N33"/>
2688                          </tp:NonRepudiation>
2689                          <tp:DigitalEnvelope>
2690                                <tp:Protocol tp:version="2.0">S/MIME</tp:Protocol>
2691                                <tp:EncryptionAlgorithm>DES-
2692  CBC</tp:EncryptionAlgorithm>
2693                                <tp:CertificateRef tp:certId="N33"/>
2694                          </tp:DigitalEnvelope>
2695                    </tp:ebXMLBinding>
2696              </tp:DocExchange>
2697        </tp:PartyInfo>
2698        <tp:Packaging tp:id="N0402">
2699              <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>
2700              <tp:SimplePart tp:id="N40" tp:mimetype="text/xml">
2701                    <tp:NamespaceSupported
2702  tp:location="http://ebxml.org/project_teams/transport/messageService.xsd"
2703  tp:version="0.98b">http://www.ebxml.org/namespaces/messageService</tp:NamespaceSupport
2704  ed>
2705                    <tp:NamespaceSupported
2706  tp:location="http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"
2707  tp:version="1.0">http://www.w3.org/2000/09/xmldsig</tp:NamespaceSupported>
2708              </tp:SimplePart>
2709              <tp:SimplePart tp:id="N41" tp:mimetype="text/xml">
2710                    <tp:NamespaceSupported
```

```
2711    tp:location="http://ebxml.org/processes/buysell.xsd"
2712    tp:version="1.0">http://ebxml.org/processes/buysell.xsd</tp:NamespaceSupported>
2713                    </tp:SimplePart>
2714                    <tp:CompositeList>
2715                            <tp:Composite tp:id="N42" tp:mimetype="multipart/related"
2716    tp:mimeparameters="type=text/xml;">
2717                                    <tp:Constituent tp:idref="N40"/>
2718                                    <tp:Constituent tp:idref="N41"/>
2719                            </tp:Composite>
2720                    </tp:CompositeList>
2721            </tp:Packaging>
2722            <tp:Comment xml:lang="en-us">buy/sell agreement between example.com and
2723    contrived-example.com</tp:Comment>
2724    </tp:CollaborationProtocolAgreement>
2725
2726
```

Appendix C   DTD Corresponding to Complete CPP/CPA

2727

2728   Definition (Normative)

2729   This DTD is available as an ASCII file at:

2730   http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.dtd

2731

```
2732    <?xml version="1.0" encoding="UTF-8"?>
2733    <!--Generated by XML Authority-->
2734    <!ELEMENT CollaborationProtocolAgreement (Status, Start, End,
2735    ConversationConstraints?, PartyInfo+, Packaging, ds:Signature*, Comment*)>
2736    <!ATTLIST CollaborationProtocolAgreement
2737            cpaid CDATA #IMPLIED
2738            version CDATA #IMPLIED
2739    >
2740    <!ELEMENT CollaborationProtocolProfile (PartyInfo+, Packaging, ds:Signature?,
2741    Comment*)>
2742    <!ATTLIST CollaborationProtocolProfile
2743            version CDATA #IMPLIED
2744    >
2745    <!ELEMENT ProcessSpecification (ds:Reference?)>
2746    <!ATTLIST ProcessSpecification
2747            version CDATA #REQUIRED
2748            name CDATA #REQUIRED
2749            xlink:type CDATA #FIXED "simple"
2750            xlink:href CDATA #IMPLIED
2751    >
2752    <!ELEMENT Protocol (#PCDATA)>
2753    <!ATTLIST Protocol
2754            version CDATA #IMPLIED
2755    >
2756    <!ELEMENT SendingProtocol (#PCDATA)>
2757    <!ATTLIST SendingProtocol
2758            version CDATA #IMPLIED
2759    >
2760    <!ELEMENT ReceivingProtocol (#PCDATA)>
2761    <!ATTLIST ReceivingProtocol
2762            version CDATA #IMPLIED
2763    >
2764    <!ELEMENT CollaborationRole (ProcessSpecification, Role, CertificateRef?,
2765    ServiceBinding+)>
2766    <!ATTLIST CollaborationRole
2767            id ID #IMPLIED
2768    >
2769    <!ELEMENT PartyInfo (PartyId+, PartyRef, CollaborationRole+, Certificate+,
2770    DeliveryChannel+, Transport+, DocExchange+)>
2771    <!ELEMENT PartyId (#PCDATA)>
2772    <!ATTLIST PartyId
2773            type CDATA #IMPLIED
2774    >
2775    <!ELEMENT PartyRef EMPTY>
2776    <!ATTLIST PartyRef
```

```
2777            xlink:type (simple) #IMPLIED
2778            xlink:href CDATA #IMPLIED
2779    >
2780    <!ELEMENT DeliveryChannel (Characteristics)>
2781    <!ATTLIST DeliveryChannel
2782            channelId ID #REQUIRED
2783            transportId IDREF #REQUIRED
2784            docExchangeId IDREF #REQUIRED
2785    >
2786    <!ELEMENT Transport (SendingProtocol+, ReceivingProtocol, Endpoint+,
2787    TransportSecurity?)>
2788    <!ATTLIST Transport
2789            transportId ID #REQUIRED
2790    >
2791    <!ELEMENT Endpoint EMPTY>
2792    <!ATTLIST Endpoint
2793            uri CDATA #REQUIRED
2794            type (login | request | response | error | allPurpose) "allPurpose"
2795    >
2796    <!ELEMENT Retries (#PCDATA)>
2797    <!ELEMENT RetryInterval (#PCDATA)>
2798    <!ELEMENT TransportSecurity (Protocol, CertificateRef?)>
2799    <!ELEMENT Certificate (ds:KeyInfo)>
2800    <!ATTLIST Certificate
2801            certId ID #REQUIRED
2802    >
2803    <!ELEMENT DocExchange (ebXMLBinding)>
2804    <!ATTLIST DocExchange
2805            docExchangeId ID #REQUIRED
2806    >
2807    <!ELEMENT PersistDuration (#PCDATA)>
2808    <!ATTLIST PersistDuration
2809            e-dtype NMTOKEN #FIXED "timeDuration"
2810    >
2811    <!ELEMENT ReliableMessaging (Retries, RetryInterval, PersistDuration)?>
2812    <!ATTLIST ReliableMessaging
2813            deliverySemantics (OnceAndOnlyOnce | BestEffort) #REQUIRED
2814            messageOrderSemantics (Guaranteed | NotGuaranteed) "NotGuaranteed"
2815            idempotency CDATA #REQUIRED
2816    >
2817    <!ELEMENT NonRepudiation (Protocol, HashFunction, SignatureAlgorithm, CertificateRef)>
2818    <!ELEMENT HashFunction (#PCDATA)>
2819    <!ELEMENT EncryptionAlgorithm (#PCDATA)>
2820    <!ELEMENT SignatureAlgorithm (#PCDATA)>
2821    <!ELEMENT DigitalEnvelope (Protocol, EncryptionAlgorithm, CertificateRef)>
2822    <!ELEMENT CertificateRef EMPTY>
2823    <!ATTLIST CertificateRef
2824            certId IDREF #REQUIRED
2825    >
2826    <!ELEMENT ebXMLBinding (ReliableMessaging?, NonRepudiation?, DigitalEnvelope?,
2827    NamespaceSupported*)>
2828    <!ATTLIST ebXMLBinding
2829            version CDATA #REQUIRED
```

Collaboration-Protocol Profile Tend Agreement Specification   **Page 69 of 90**

```
2830    >
2831    <!ELEMENT NamespaceSupported (#PCDATA)>
2832    <!ATTLIST NamespaceSupported
2833          location CDATA #REQUIRED
2834          version CDATA #IMPLIED
2835    >
2836    <!ELEMENT Characteristics EMPTY>
2837    <!ATTLIST Characteristics
2838          syncReplyMode (responseOnly | signalsAndResponse | signalsOnly | none) #IMPLIED
2839          nonrepudiationOfOrigin CDATA #IMPLIED
2840          nonrepudiationOfReceipt CDATA #IMPLIED
2841          secureTransport CDATA #IMPLIED
2842          confidentiality CDATA #IMPLIED
2843          authenticated CDATA #IMPLIED
2844          authorized CDATA #IMPLIED
2845    >
2846    <!ELEMENT ServiceBinding (Service, Override*)>
2847    <!ATTLIST ServiceBinding
2848          channelId IDREF #REQUIRED
2849          packageId IDREF #REQUIRED
2850    >
2851    <!ELEMENT Service (#PCDATA)>
2852    <!ATTLIST Service
2853          type CDATA #IMPLIED>
2854
2855    <!ELEMENT Status EMPTY>
2856    <!ATTLIST Status
2857          value (agreed | signed | proposed) #REQUIRED
2858    >
2859    <!ELEMENT Start (#PCDATA)>
2860    <!ELEMENT End (#PCDATA)>
2861    <!ELEMENT Type (#PCDATA)>
2862    <!ELEMENT ConversationConstraints EMPTY>
2863    <!ATTLIST ConversationConstraints
2864          invocationLimit CDATA #IMPLIED
2865          concurrentConversations CDATA #IMPLIED
2866    >
2867    <!ELEMENT Override EMPTY>
2868    <!ATTLIST Override
2869          action CDATA #REQUIRED
2870          channelId ID #REQUIRED
2871          packageId IDREF #REQUIRED
2872          xlink:href CDATA #IMPLIED
2873          xlink:type CDATA #FIXED "simple"
2874    >
2875    <!ELEMENT Role EMPTY>
2876    <!ATTLIST Role
2877          name CDATA #REQUIRED
2878          xlink:type CDATA #FIXED "simple"
2879          xlink:href CDATA #IMPLIED
2880    >
2881    <!ELEMENT Constituent EMPTY>
2882    <!ATTLIST Constituent
```

```
2883            idref CDATA #REQUIRED
2884    >
2885    <!ELEMENT ProcessingCapabilities EMPTY>
2886    <!ATTLIST ProcessingCapabilities
2887            parse CDATA #REQUIRED
2888            generate CDATA #REQUIRED
2889    >
2890    <!ELEMENT SimplePart (NamespaceSupported*)>
2891    <!ATTLIST SimplePart
2892            id ID #IMPLIED
2893            mimetype CDATA #REQUIRED
2894    >
2895    <!ELEMENT Encapsulation (Constituent)>
2896    <!ATTLIST Encapsulation
2897            id ID #IMPLIED
2898            mimetype CDATA #REQUIRED
2899            mimeparameters CDATA #IMPLIED
2900    >
2901    <!ELEMENT Composite (Constituent+)>
2902    <!ATTLIST Composite
2903            id ID #IMPLIED
2904            mimetype CDATA #REQUIRED
2905            mimeparameters CDATA #IMPLIED
2906    >
2907    <!ELEMENT CompositeList (Encapsulation | Composite)+>
2908    <!ELEMENT Packaging (ProcessingCapabilities, SimplePart+, CompositeList?)>
2909    <!ATTLIST Packaging
2910            id ID #REQUIRED
2911    >
2912    <!ELEMENT Comment (#PCDATA)>
2913    <!ATTLIST Comment
2914            xml:lang CDATA #REQUIRED
2915    >
2916    <!ELEMENT ds:Signature ANY>
2917    <!ELEMENT ds:Reference ANY>
2918    <!ELEMENT ds:KeyInfo ANY>
2919
```

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

2920
2921

2922 This XML Schema document is available as an ASCII file at:
2923   http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.xsd
2924 <?xml version="1.0" encoding="UTF-8"?>
2925 <schema targetNamespace="http://www.ebxml.org/namespaces/tradePartner"
2926 xmlns:xml="http://www.w3.org/XML/1998/namespace"
2927 xmlns="http://www.w3.org/2000/10/XMLSchema"
2928 xmlns:tns="http://www.ebxml.org/namespaces/tradePartner"
2929 xmlns:xlink="http://www.w3.org/1999/xlink"
2930 xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2931 xmlns:ds="http://www.w3.org/2000/09/xmldsig#" elementFormDefault="qualified"
2932 attributeFormDefault="unqualified" version="1.0">
2933   <import namespace="http://www.w3.org/1999/xlink"
2934 schemaLocation="http://ebxml.org/project_teams/transport/xlink.xsd"/>
2935   <import namespace="http://www.w3.org/2000/09/xmldsig#"
2936 schemaLocation="http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"/>
2937   <import namespace="http://www.w3.org/XML/1998/namespace"
2938 schemaLocation="http://ebxml.org/project_teams/transport/xml_lang.xsd"/>
2939   <attributeGroup name="pkg.grp">
2940   <attribute ref="tns:id"/>
2941   <attribute name="mimetype" type="tns:non-empty-string" use="required"/>
2942   <attribute name="mimeparameters" type="tns:non-empty-string"/>
2943   </attributeGroup>
2944   <attributeGroup name="xlink.grp">
2945   <attribute ref="xlink:type"/>
2946   <attribute ref="xlink:href"/>
2947   </attributeGroup>
2948   <element name="CollaborationProtocolAgreement">
2949   <complexType>
2950   <sequence>
2951   <element ref="tns:Status"/>
2952   <element ref="tns:Start"/>
2953   <element ref="tns:End"/>
2954   <element ref="tns:ConversationConstraints" minOccurs="0"/>
2955   <element ref="tns:PartyInfo" maxOccurs="unbounded"/>
2956   <element ref="tns:Packaging"/>
2957   <element ref="ds:Signature" minOccurs="0"
2958 maxOccurs="unbounded"/>
2959   <element ref="tns:Comment" minOccurs="0"
2960 maxOccurs="unbounded"/>
2961   </sequence>
2962   <attribute name="cpaid" type="tns:non-empty-string"/>
2963   <attribute ref="tns:version"/>
2964   <anyAttribute namespace="##targetNamespace
2965 http://www.w3.org/2000/10/XMLSchema-instance" processContents="lax"/>
2966   </complexType>
2967   </element>
2968   <element name="CollaborationProtocolProfile">
2969   <complexType>

```
2970                    <sequence>
2971                            <element ref="tns:PartyInfo" maxOccurs="unbounded"/>
2972                            <element ref="tns:Packaging"/>
2973                            <element ref="ds:Signature" minOccurs="0"/>
2974                            <element ref="tns:Comment" minOccurs="0"
2975    maxOccurs="unbounded"/>
2976                    </sequence>
2977                    <attribute ref="tns:version"/>
2978                    <anyAttribute namespace="##targetNamespace
2979    http://www.w3.org/2000/10/XMLSchema-instance" processContents="lax"/>
2980            </complexType>
2981        </element>
2982        <element name="ProcessSpecification">
2983            <complexType>
2984                    <sequence>
2985                            <element ref="ds:Reference" minOccurs="0"/>
2986                    </sequence>
2987                    <attribute ref="tns:version"/>
2988                    <attribute name="name" type="tns:non-empty-string"
2989    use="required"/>
2990                    <attributeGroup ref="tns:xlink.grp"/>
2991            </complexType>
2992        </element>
2993        <element name="Service" type="tns:service.type"/>
2994        <element name="Protocol" type="tns:protocol.type"/>
2995        <element name="SendingProtocol" type="tns:protocol.type"/>
2996        <element name="ReceivingProtocol" type="tns:protocol.type"/>
2997        <element name="CollaborationRole">
2998            <complexType>
2999                    <sequence>
3000                            <element ref="tns:ProcessSpecification"/>
3001                            <element ref="tns:Role"/>
3002                            <element ref="tns:CertificateRef" minOccurs="0"/>
3003                            <element ref="tns:ServiceBinding" maxOccurs="unbounded"/>
3004                    </sequence>
3005                    <attribute ref="tns:id"/>
3006            </complexType>
3007        </element>
3008        <element name="PartyInfo">
3009            <complexType>
3010                    <sequence>
3011                            <element ref="tns:PartyId" maxOccurs="unbounded"/>
3012                            <element ref="tns:PartyRef"/>
3013                            <element ref="tns:CollaborationRole"
3014    maxOccurs="unbounded"/>
3015                            <element ref="tns:Certificate" maxOccurs="unbounded"/>
3016                            <element ref="tns:DeliveryChannel" maxOccurs="unbounded"/>
3017                            <element ref="tns:Transport" maxOccurs="unbounded"/>
3018                            <element ref="tns:DocExchange" maxOccurs="unbounded"/>
3019                    </sequence>
3020            </complexType>
3021        </element>
3022        <element name="PartyId">
```

```
3023              <complexType>
3024                     <simpleContent>
3025                            <extension base="tns:non-empty-string">
3026                                   <attribute name="type" type="tns:non-empty-string"/>
3027                            </extension>
3028                     </simpleContent>
3029              </complexType>
3030       </element>
3031       <element name="PartyRef">
3032              <complexType>
3033                     <attributeGroup ref="tns:xlink.grp"/>
3034                     <attribute name="type" type="tns:non-empty-string"/>
3035              </complexType>
3036       </element>
3037       <element name="DeliveryChannel">
3038              <complexType>
3039                     <sequence>
3040                            <element ref="tns:Characteristics"/>
3041                     </sequence>
3042                     <attribute name="channelId" type="ID" use="required"/>
3043                     <attribute name="transportId" type="IDREF" use="required"/>
3044                     <attribute name="docExchangeId" type="IDREF" use="required"/>
3045              </complexType>
3046       </element>
3047       <element name="Transport">
3048              <complexType>
3049                     <sequence>
3050                            <element ref="tns:SendingProtocol" maxOccurs="unbounded"/>
3051                            <element ref="tns:ReceivingProtocol"/>
3052                            <element ref="tns:Endpoint" maxOccurs="unbounded"/>
3053                            <element ref="tns:TransportSecurity" minOccurs="0"/>
3054                     </sequence>
3055                     <attribute name="transportId" type="ID" use="required"/>
3056              </complexType>
3057       </element>
3058       <element name="Endpoint">
3059              <complexType>
3060                     <attribute name="uri" type="uriReference" use="required"/>
3061                     <attribute name="type" type="tns:endpointType.type" use="default"
3062  value="allPurpose"/>
3063              </complexType>
3064       </element>
3065       <element name="Retries" type="string"/>
3066       <element name="RetryInterval" type="string"/>
3067       <element name="TransportSecurity">
3068              <complexType>
3069                     <sequence>
3070                            <element ref="tns:Protocol"/>
3071                            <element ref="tns:CertificateRef" minOccurs="0"/>
3072                     </sequence>
3073              </complexType>
3074       </element>
3075       <element name="Certificate">
```

Collaboration-Protocol Profile Tend Agreement Specification   **Page 74 of 90**

```
3076                     <complexType>
3077                             <sequence>
3078                                     <element ref="ds:KeyInfo"/>
3079                             </sequence>
3080                             <attribute name="certId" type="ID" use="required"/>
3081                     </complexType>
3082             </element>
3083             <element name="DocExchange">
3084                     <complexType>
3085                             <sequence>
3086                                     <element ref="tns:ebXMLBinding"/>
3087                             </sequence>
3088                             <attribute name="docExchangeId" type="ID" use="required"/>
3089                     </complexType>
3090             </element>
3091             <element name="ReliableMessaging">
3092                     <complexType>
3093                             <sequence minOccurs="0">
3094                                     <element ref="tns:Retries"/>
3095                                     <element ref="tns:RetryInterval"/>
3096                                     <element name="PersistDuration" type="timeDuration"/>
3097                             </sequence>
3098                             <attribute name="deliverySemantics" type="tns:ds.type"
3099     use="required"/>
3100                             <attribute name="idempotency" type="boolean" use="required"/>
3101                             <attribute name="messageOrderSemantics" type="tns:mos.type"
3102     use="optional" value="NotGuaranteed"/>
3103                     </complexType>
3104                     <!-- <element name="PersistDuration" type="duration"/> -->
3105             </element>
3106             <element name="NonRepudiation">
3107                     <complexType>
3108                             <sequence>
3109                                     <element ref="tns:Protocol"/>
3110                                     <element ref="tns:HashFunction"/>
3111                                     <element ref="tns:SignatureAlgorithm"/>
3112                                     <element ref="tns:CertificateRef"/>
3113                             </sequence>
3114                     </complexType>
3115             </element>
3116             <element name="HashFunction" type="string"/>
3117             <element name="EncryptionAlgorithm" type="string"/>
3118             <element name="SignatureAlgorithm" type="string"/>
3119             <element name="DigitalEnvelope">
3120                     <complexType>
3121                             <sequence>
3122                                     <element ref="tns:Protocol"/>
3123                                     <element ref="tns:EncryptionAlgorithm"/>
3124                                     <element ref="tns:CertificateRef"/>
3125                             </sequence>
3126                     </complexType>
3127             </element>
3128             <element name="CertificateRef">
```

```
3129                    <complexType>
3130                            <attribute name="certId" type="IDREF" use="required"/>
3131                    </complexType>
3132            </element>
3133            <element name="ebXMLBinding">
3134                    <complexType>
3135                            <sequence>
3136                                    <element ref="tns:ReliableMessaging" minOccurs="0"/>
3137                                    <element ref="tns:NonRepudiation" minOccurs="0"/>
3138                                    <element ref="tns:DigitalEnvelope" minOccurs="0"/>
3139                                    <element ref="tns:NamespaceSupported" minOccurs="0"
3140     maxOccurs="unbounded"/>
3141                            </sequence>
3142                            <attribute ref="tns:version"/>
3143                    </complexType>
3144            </element>
3145            <element name="NamespaceSupported">
3146                    <complexType>
3147                            <simpleContent>
3148                                    <extension base="uriReference">
3149                                            <attribute name="location" type="uriReference"
3150     use="required"/>
3151                                            <attribute ref="tns:version"/>
3152                                    </extension>
3153                            </simpleContent>
3154                    </complexType>
3155            </element>
3156            <element name="Characteristics">
3157                    <complexType>
3158                            <attribute ref="tns:syncReplyMode"/>
3159                            <attribute name="nonrepudiationOfOrigin" type="boolean"/>
3160                            <attribute name="nonrepudiationOfReceipt" type="boolean"/>
3161                            <attribute name="secureTransport" type="boolean"/>
3162                            <attribute name="confidentiality" type="boolean"/>
3163                            <attribute name="authenticated" type="boolean"/>
3164                            <attribute name="authorized" type="boolean"/>
3165                    </complexType>
3166            </element>
3167            <element name="ServiceBinding">
3168                    <complexType>
3169                            <sequence>
3170                                    <element ref="tns:Service"/>
3171                                    <element ref="tns:Override" minOccurs="0"
3172     maxOccurs="unbounded"/>
3173                            </sequence>
3174                            <attribute name="channelId" type="IDREF" use="required"/>
3175                            <attribute name="packageId" type="IDREF" use="required"/>
3176                    </complexType>
3177                    <unique name="action.const">
3178                            <selector xpath=".//Override"/>
3179                            <field xpath="@action"/>
3180                    </unique>
3181            </element>
```

```
3182          <element name="Status">
3183                  <complexType>
3184                          <attribute name="value" type="tns:statusValue.type"
3185  use="required"/>
3186                  </complexType>
3187          </element>
3188          <element name="Start" type="timeInstant"/>
3189          <element name="End" type="timeInstant"/>
3190          <!--
3191          <element name="Start" type="dateTime"/>
3192          <element name="End" type="dateTime"/>
3193          -->
3194          <element name="Type" type="string"/>
3195          <element name="ConversationConstraints">
3196                  <complexType>
3197                          <attribute name="invocationLimit" type="int"/>
3198                          <attribute name="concurrentConversations" type="int"/>
3199                  </complexType>
3200          </element>
3201          <element name="Override">
3202                  <complexType>
3203                          <attribute name="action" type="tns:non-empty-string"
3204  use="required"/>
3205                          <attribute name="channelId" type="ID" use="required"/>
3206                          <attribute name="packageId" type="IDREF" use="required"/>
3207                          <attributeGroup ref="tns:xlink.grp"/>
3208                  </complexType>
3209          </element>
3210          <element name="Role">
3211                  <complexType>
3212                          <attribute name="name" type="tns:non-empty-string"
3213  use="required"/>
3214                          <attributeGroup ref="tns:xlink.grp"/>
3215                  </complexType>
3216          </element>
3217          <element name="Constituent">
3218                  <complexType>
3219                          <attribute ref="tns:idref"/>
3220                  </complexType>
3221          </element>
3222          <element name="Packaging">
3223                  <complexType>
3224                          <sequence>
3225                                  <element name="ProcessingCapabilities">
3226                                          <complexType>
3227                                                  <attribute name="parse" type="boolean"
3228  use="required"/>
3229                                                  <attribute name="generate" type="boolean"
3230  use="required"/>
3231                                          </complexType>
3232                                  </element>
3233                                  <element name="SimplePart" maxOccurs="unbounded">
3234                                          <complexType>
```

```
3235                                             <sequence>
3236                                                   <element ref="tns:NamespaceSupported"
3237  minOccurs="0" maxOccurs="unbounded"/>
3238                                             </sequence>
3239                                             <attributeGroup ref="tns:pkg.grp"/>
3240                                       </complexType>
3241                                 </element>
3242                                 <element name="CompositeList" minOccurs="0">
3243                                       <complexType>
3244                                             <choice maxOccurs="unbounded">
3245                                                   <element name="Encapsulation">
3246                                                         <complexType>
3247                                                               <sequence>
3248                                                                     <element
3249  ref="tns:Constituent"/>
3250                                                               </sequence>
3251                                                               <attributeGroup
3252  ref="tns:pkg.grp"/>
3253                                                         </complexType>
3254                                                   </element>
3255                                                   <element name="Composite">
3256                                                         <complexType>
3257                                                               <sequence>
3258                                                                     <element
3259  ref="tns:Constituent" maxOccurs="unbounded"/>
3260                                                               </sequence>
3261                                                               <attributeGroup
3262  ref="tns:pkg.grp"/>
3263                                                         </complexType>
3264                                                   </element>
3265                                             </choice>
3266                                       </complexType>
3267                                 </element>
3268                           </sequence>
3269                           <attribute ref="tns:id"/>
3270                     </complexType>
3271               </element>
3272               <element name="Comment">
3273                     <complexType>
3274                           <simpleContent>
3275                                 <extension base="tns:non-empty-string">
3276                                       <attribute ref="xml:lang"/>
3277                                 </extension>
3278                           </simpleContent>
3279                     </complexType>
3280               </element>
3281               <!-- COMMON -->
3282               <simpleType name="ds.type">
3283                     <restriction base="NMTOKEN">
3284                           <enumeration value="OnceAndOnlyOnce"/>
3285                           <enumeration value="BestEffort"/>
3286                     </restriction>
3287               </simpleType>
```

Collaboration-Protocol Profile Tand Agreement Specification   **Page 78 of 90**

```
3288          <simpleType name="mos.type">
3289                  <restriction base="NMTOKEN">
3290                          <enumeration value="Guaranteed"/>
3291                          <enumeration value="NotGuaranteed"/>
3292                  </restriction>
3293          </simpleType>
3294          <simpleType name="statusValue.type">
3295                  <restriction base="NMTOKEN">
3296                          <enumeration value="agreed"/>
3297                          <enumeration value="signed"/>
3298                          <enumeration value="proposed"/>
3299                  </restriction>
3300          </simpleType>
3301          <simpleType name="endpointType.type">
3302                  <restriction base="NMTOKEN">
3303                          <enumeration value="login"/>
3304                          <enumeration value="request"/>
3305                          <enumeration value="response"/>
3306                          <enumeration value="error"/>
3307                          <enumeration value="allPurpose"/>
3308                  </restriction>
3309          </simpleType>
3310          <simpleType name="non-empty-string">
3311                  <restriction base="string">
3312                          <minLength value="1"/>
3313                  </restriction>
3314          </simpleType>
3315          <simpleType name="syncReplyMode.type">
3316                  <restriction base="NMTOKEN">
3317                          <enumeration value="responseOnly"/>
3318                          <enumeration value="signalsAndResponse"/>
3319                          <enumeration value="signalsOnly"/>
3320                          <enumeration value="none"/>
3321                  </restriction>
3322          </simpleType>
3323          <complexType name="service.type">
3324                  <simpleContent>
3325                          <extension base="tns:non-empty-string">
3326                                  <attribute name="type" type="tns:non-empty-string"/>
3327                          </extension>
3328                  </simpleContent>
3329          </complexType>
3330          <complexType name="protocol.type">
3331                  <simpleContent>
3332                          <extension base="tns:non-empty-string">
3333                                  <attribute ref="tns:version"/>
3334                          </extension>
3335                  </simpleContent>
3336          </complexType>
3337          <attribute name="idref" type="IDREF" form="unqualified"/>
3338          <attribute name="id" type="ID" form="unqualified"/>
3339          <attribute name="version" type="tns:non-empty-string"/>
3340          <attribute name="syncReplyMode" type="tns:syncReplyMode.type"/>
```

Collaboration-Protocol Profile Tend Agreement Specification   **Page 79 of 90**

```
3341    </schema>
```

3342 **Appendix E** Formats of Information in the CPP and CPA

3343   (Normative)

3344 This section defines format information that is not defined by the [XML] specification and is not
3345 defined in the descriptions of specific elements.

3346

3347 Formats of Character Strings

3348

3349 **Protocol and Version Elements**

3350

3351 Values of *Protocol* , *Version*, and similar elements are flexible. In general, any protocol and
3352 version for which the support software is available to both *Parties* to a *CPA* MAY be selected as
3353 long as the choice does not require changes to the DTD or schema and therefore a change to this
3354 specification.

3355

3356   NOTE: A possible implementation MAY be based on the use of plug-ins or exits to
3357   support the values of these elements.

3358

3359 **Alphanumeric Strings**

3360

3361 Alphanumeric strings not further defined in this section follow these rules unless otherwise
3362 stated in the description of an individual element:

3363

3364 • Values of elements are case insensitive unless otherwise stated.
3365 • Strings which represent file or directory names are case sensitive to ensure that they are
3366   acceptable to both UNIX and Windows systems.

3367

3368 **Numeric Strings**

3369

3370 A numeric string is a signed or unsigned decimal integer in the range imposed by a 32-bit binary
3371 number, i.e. -2,147,483,648 to +2,417,483,647. Negative numbers MAY or MAY not be
3372 permitted in particular elements.

3373   **Appendix F**   Composing a CPA from Two CPPs (Non-
3374   Normative)

3375
3376   Overview and Limitations

3377
3378   In this appendix, we discuss the tasks involved in *CPA* formation from *CPPs*. The detailed
3379   procedures for *CPA* formation are currently left for implementers. Therefore, no normative
3380   specification is provided for algorithms for *CPA* formation. In this initial section, we provide
3381   some background on *CPA* formation tasks.

3382
3383   There are three basic reasons why we prefer to provide information about the component tasks
3384   involved in *CPA* formation rather than attempt to provide an algorithm for *CPA* formation:

3385
3386   1. The precise informational inputs to the *CPA* formation procedure vary.
3387   2. There exist at least two distinct approaches to *CPA* formation. One useful approach for
3388   certain situations involves basing *CPA* formation from a *CPA* template; the other approach
3389   involves composition from *CPPs*.
3390   3. The conditions for output of a given *CPA* given two *CPPs* can involve different levels and
3391   extents of interoperability. In other words, when an optimal solution that satisfies every level
3392   of requirement and every other additional constraint does not exist, a *Party* MAY propose a
3393   *CPA* that satisfies enough of the requirements for "a good enough" implementation. User
3394   input MAY be solicited to determine what is a good enough implementation, and so MAY
3395   be as varied as there are user configuration options to express preferences. In practice,
3396   compromises MAY be made on security, reliable messaging, levels of signals and
3397   acknowledgements, and other matters in order to find some acceptable means of doing
3398   *Business*.

3399
3400   Each of these reasons is elaborated in greater detail in the following sections.

3401
3402
3403   Variability in Inputs

3404
3405   User preferences provide one source of variability in the inputs to the *CPA* formation process.
3406   Let us suppose in this section that each of the *Parties* has made its *CPP* available to potential
3407   collaborators. Normally one *Party* will have a desired *Business Collaboration* (defined in a
3408   *Process-Specification* document*)* to implement with its intended collaborator. So the information
3409   inputs will normally involve a user preference about intended *Business Collaboration* in addition
3410   to just the *CPPs*.

3411
3412   A *CPA* formation tool MAY have access to local user information not advertised in the *CPP* that
3413   MAY contribute to the *CPA* that is formed. A user MAY have chosen to only advertise those
3414   system capabilities that reflect nondeprecated capabilities. For example, a user MAY only
3415   advertise HTTP and omit FTP, even when capable of using FTP. The reason for omitting FTP
3416   might be concerns about the scalability of managing user accounts, directories, and passwords

3417 for FTP sessions. Despite not advertising an FTP capability, configuration software MAY use
3418 tacit knowledge about its own FTP capability to form a *CPA* with an intended collaborator who
3419 happens to have only an FTP capability for implementing a desired *Business Collaboration*. In
3420 other words, *Business* interests MAY, in this case, override the deprecation policy. Both tacit
3421 knowledge and detailed preference information account for variability in inputs into the *CPA*
3422 formation process.

3423
3424

3425 Different Approaches

3426
3427 When a *CPA* is formed from a *CPA* template, it is typically because the capabilities of one of the
3428 *Parties* are limited, and already tacitly known. For example, if a *CPA* template were implicitly
3429 presented to a Web browser for use in an implementation using browser based forms capabilities,
3430 then the template maker can assume that the other *Party* has suitable web capabilities (or is about
3431 to download them). Therefore, all that really needs to be done is to supply ***PartyRef, Certificate***,
3432 and similar items for substitution into a *CPA* template. The *CPA* template will already have all
3433 the capabilities of both *Parties* specified at the various levels, and will have placeholders for
3434 values to be supplied by one of the *Partners*. A simple form might be adequate to gather the
3435 needed information and produce a *CPA*.

3436
3437

3438 Variable Output "Satisficing" Policies

3439
3440 A *CPA* can support a fully interoperable configuration in which agreement has been reached on
3441 all technical levels needed for *Business Collaboration*. In such a case, matches in capabilities
3442 will have been found in all relevant technical levels.

3443
3444 However, there can be interoperable configurations agreed to in a *CPA* in which not all aspects
3445 of a *Business Collaboration* match. Gaps MAY exist in packaging, security, signaling, reliable
3446 messaging and other areas and yet the systems can still transport the *Business* data, and special
3447 means can be employed to handle the exceptions. In such situations, a *CPA* MAY reflect
3448 configured policies or expressly solicited user permission to ignore some shortcomings in
3449 configurations. A system might not be capable of responding in a *Business Collaboration* so as
3450 to support a recommended ability to supply nonrepudiation of receipt, but might still be
3451 acceptable for *Business* reasons. A system might not be able to handle all the processing required
3452 to support, for example, SOAP with Attachments and yet still be able to treat the multipart
3453 according to "multipart/mixed" handling and allow *Business Collaboration* to take place. In fact,
3454 short of a failure to be able to transport data and a failure to be able to provide data relevant to
3455 the *Business Collaboration*, there are few features that might not be temporarily or indefinitely
3456 compromised about, given overriding *Business* interests. This situation of "partial
3457 interoperability" is to be expected to persist for some time, and so interferes with formulating a
3458 "clean" algorithm for deciding on what is sufficient for interoperability.

3459
3460 In summary, the previous considerations indicate that at the present it is at best premature to seek
3461 a simple algorithm for *CPA* formation from *CPPs*. It is to be expected that as capability
3462 characterization and exchange becomes a more refined subject, that advances will be made in

3463   characterizing *CPA* formation and negotiation.

3464

3465   Despite it being too soon to propose a simple algorithm for *CPA* formation that covers all the
3466   above variations, it is currently possible to enumerate the basic tasks involved in matching
3467   capabilities within *CPPs*. This information might assist the software implementer in designing a
3468   partially automated and partially interactive software system useful for configuring *Business*
3469   *Collaboration* so as to arrive at satisfactorily complete levels of interoperability. To understand
3470   the context for characterizing the constituent tasks, the general perspective on *CPPs* and *CPAs*
3471   needs to be briefly recalled.

3472
3473

3474   ## CPA Formation Component Tasks

3475

3476   Technically viewed, a *CPA* provides "bindings" between *Business-Collaboration* specifications
3477   (as defined in the *Process-Specification* document) and those services and protocols that are used
3478   to implement these specifications. The implementation takes place at several levels and involves
3479   varied services at these levels. A *CPA* that arrives at a fully interoperable binding of a *Business*
3480   *Collaboration* to its implementing services and protocols can be thought of as arriving at
3481   interoperable, application-to-application integration. *CPAs* MAY fall short of this goal and still
3482   be useful and acceptable to the collaborating *Parties*. Certainly, if no matching data-transport
3483   capabilities can be discovered, a *CPA* would not provide much in the way of interoperable
3484   *Business*-to-*Business* integration. Likewise, partial *CPAs* will leave significant system work to be
3485   done before a completely satisfactory application-to-application integration is realized. Even so,
3486   partial integration MAY be sufficient to allow collaboration, and to enjoy payoffs from increased
3487   levels of automation.

3488

3489   In practice, the *CPA* formation process MAY produce a complete *CPA*, a failure result, a gap list
3490   that drives a dialog with the user, or perhaps even a *CPA* that implements partial interoperability
3491   "good enough" for the *Business* collaborators. Because both matching capabilities and
3492   interoperability can be matters of degree, the constituent tasks are finding the matches in
3493   capabilities at different levels and for different services. We next proceed to characterize many
3494   of these constituent tasks.

3495
3496

3497   ## CPA Formation from CPPs: Enumeration of Tasks

3498

3499   To simplify discussion, assume in the following that we are viewing the tasks faced by a
3500   software agent when:
3501   1. an intended collaborator is known and the collaborator's *CPP* has been retrieved,
3502   2. the *Business Collaboration* between us and our intended collaborator has been selected,
3503   3. the specific role that our software agent is to play in the *Business Collaboration* is
3504   known, and
3505   4. the capabilities that are to be advertised in our *CPP* are known.

3506

3507   For vividness, we will suppose that our example agent wishes to play the role of supplier and
3508   seeks to find one of its current customers to begin a Purchase Order *Business Collaboration* in

3509   which the intended player plays a complementary role. For simplicity, we assume that the
3510   information about capabilities is restricted to what is available in our agent's *CPP* and in the
3511   *CPP* of its intended collaborator.
3512
3513   In general, the constituent tasks consist of finding "matches" between our capabilities and our
3514   intended collaborator's at the various levels of the protocol stacks and with respect to the
3515   services supplied at these various levels.
3516
3517   Figure 6 illustrates the basic tasks informing a *CPA* from two *CPPs*: matching roles, matching
3518   packaging, and matching transport.
3519

Figure 6: Basic Tasks in Forming a CPA
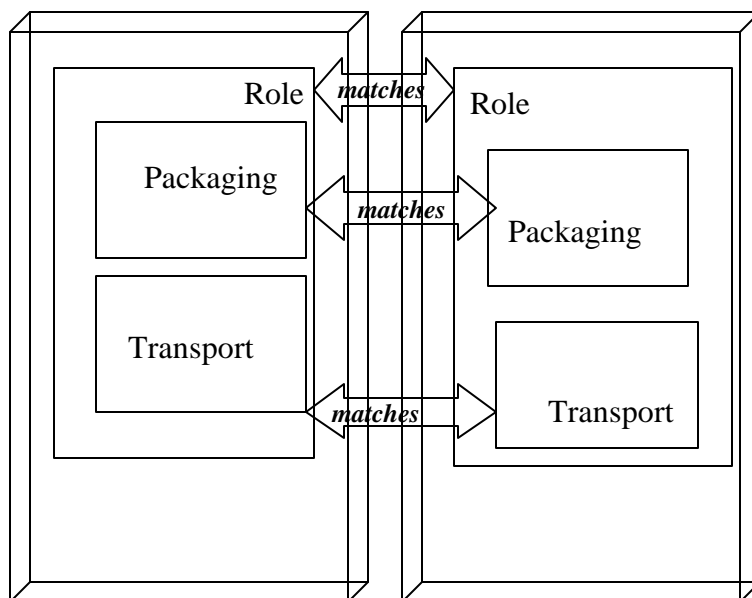
3520
3521   The first task to be considered is certainly the most basic: finding that our intended collaborator
3522   and ourselves have complementary role capabilities.
3523
3524

Matching Roles

3526
3527   Our agent has its role already selected in the *Business Collaboration*. So it now begins to check
3528   the **Role** elements in its collaborator's *CPP*. The first element to examine is the **PartyInfo**
3529   element that contains a subtree of elements called **CollaborationRole**. This set is searched to
3530   discover a role that complements the role of our agent within the *Business Collaboration* that we
3531   have chosen. For simple binary collaboration cases, it is typically sufficient to find that our

Collaboration-Protocol Profile Tend Agreement Specification   **Page 85 of 90**

3532 intended collaborator's *CollaborationRole* set contains *ProcessSpecification* elements that we
3533 intend to implement and where the role is not identical to our role. For more general
3534 collaborations, we would need to know the list of roles available within the process, and keep
3535 track that for each of the collaborators, the roles chosen instantiate those that have been specified
3536 within the *Process-Specification* document. Collaborations involving more than two roles are not
3537 discussed further.
3538
3539

Matching Transport

3541
3542 We now have available a list of candidate *CollaborationRole* elements with the desired
3543 *ProcessSpecification* element (Purchase Ordering) and where our intended collaborator plays the
3544 buyer role. For simplicity, we shall suppose just one *CollaborationRole* element meets these
3545 conditions within each of the relevant *CPPs* and not discuss iterating over lists. (Within these
3546 remarks, where repetition is possible, we will frame the discussion by assuming that just one
3547 element is present.)
3548
3549 Matching transport first means matching the *SendingProtocol* capabilities of our intended
3550 collaborator with the *ReceivingProtocol* capabilities found on our side. Perusal of the *CPP* DTD
3551 or Schema will reveal that the *ServiceBinding* element provides the doorway to the relevant
3552 information from each side's *CollaborationRole* element with the *channelId* attribute. This
3553 *channelId* attribute's value allows us to find *DeliveryChannels* within each *CPP*. The
3554 *DeliveryChannel* has a *transportId* attribute that allows us to find the relevant *Transport*
3555 subtrees.
3556
3557 For example, suppose that our intended buyer has a *Tranport* entry:
3558
```
3559 <Transport transportId = "buyerid001">
3560       <SendingProtocol>HTTP</SendingProtocol>
3561       <ReceivingProtocol>
3562       HTTP
3563       </ReceivingProtocol>
3564       <Endpoint uri = "https://www.buyername.com/po-response"
3565                 type = "allPurpose"/>
3566       <TransportSecurity>
3567             <Protocol version = "1.0">TLS</Protocol>
3568             <CertificateRef certId =  certid001">BuyerName</CertificateRef>
3569       </TransportSecurity>
3570 </Transport>
```
3571
3572 and our seller has a Transport entry:
3573
```
3574 <Transport transportId = "sellid001">
3575       <SendingProtocol>HTTP</SendingProtocol>
3576       <ReceivingProtocol>
3577       HTTP
3578       </ReceivingProtocol>
3579       <Endpoint uri = "https://www.sellername.com/pos_here"
3580                 type = "allPurpose"/>
3581       <TransportSecurity>
```

```
3582                    <Protocol version = "1.0">TLS</Protocol>
3583                    <CertificateRef certId ="certid002">Sellername</CertificateRef>
3584            </TransportSecurity>
3585    </Transport>
3586
```

3587   A transport match for requests involves finding the initiator role or buyer has a **SendingProtocol**
3588   that matches one of our **ReceivingProtocol**s. So here, "HTTP" provides a match. A transport
3589   match for responses involves finding the responder role or seller has a **SendingProtocol** that
3590   matches one of the buyer's **ReceivingProtocol**s. So in the above example, "HTTP" again
3591   provides a match. When such matches exist, we then have discovered an interoperable solution at
3592   the transport level. If not, no *CPA* will be available, and a high-priority gap has been identified
3593   that will need to be remedied by whatever exception handling procedures are in place.
3594
3595

3596   ## Matching Transport Security
3597

3598   Matches in transport security, such as in the above, will reflect agreement in versions and values
3599   of protocols. Software can supply some knowledge here so that if one side has SSL-3 and the
3600   other TLS-1, it can guess that security is available by means of a fallback of TLS to SSL.
3601
3602

3603   ## Matching Document Packaging
3604

3605   Probably one of the most complex matching problems arises when it comes to finding whether
3606   there are matches in document-packaging capabilities. Here both security and other MIME
3607   handling capabilities can combine to create complexity for appraising whether full
3608   interoperability can be attained.
3609

3610   Access to the information needed for undertaking this task is found under the **ServiceBinding**
3611   elements, and again we suppose that each side has just one **ServiceBinding** element. However,
3612   we will initially suppose that two **Packaging** elements are available to consider under each role.
3613   Several quite different ways of thinking about the matching task are available, and several
3614   methods for the tasks MAY be performed when assessing whether a good enough match exists.
3615

3616   To continue our previous purchase-ordering example, we recall that the packaging is the
3617   particular combination of body parts, XML instances (*Header*s and payloads), and security
3618   encapsulations used in assembling the *Message* from its data sources. Both requests and
3619   responses will have packaging. The most complete specification of packaging, which MAY not
3620   always be needed, would consist of:
3621

3622   1. The buyer asserting what packaging it can generate for its purchase order, and what
3623   packaging it can parse for its purchase order response *Messages*.
3624   2. The seller asserting what packaging it can generate for its purchase order responses and
3625   what packaging it can parse for received purchase orders.
3626

3627   Matching by structural comparison would then involve comparing the packaging details of the
3628   purchase orders generated by the seller with the purchase orders parsable by the buyer. The

3629   comparison would seek to establish that the MIME types of the *SimplePart* elements of
3630   corresponding subtrees match and would then proceed to check that the *CompositeList* matched
3631   in MIME types and in sequence of composition.
3632
3633   For example, if each *CPP* contained the packaging subtrees below, and under the appropriate
3634   *ServiceBindings,* then there would be a straightforward match by structural comparison:
3635

```
3636    <Packaging id="I1001">
3637            <ProcessingCapabilities parse = "true" generate = "true"/>
3638            <SimplePart id = "P1" mimetype = "text/xml"/>
3639                <NamespaceSupported location
3640                     = "http://schemas.xmlsoap.org/soap/envelope/" version = "1.1">
3641                    http://schemas.xmlsoap.org/soap/envelope
3642                </NamespaceSupported>
3643                <NamespaceSupported location =
3644                    "http://www.ebxml.org/namespaces/messageHeader"
3645                    version = "1.0">
3646                    http://www.ebxml.org/namespaces/messageHeader
3647            </NamespaceSupported>              <NamespaceSupported location =
3648                     "http://www.w3.org/2000/09/xmldsig#"
3649                    version = "1.0">
3650                   http://www.w3.org/2000/09/xmldsig#
3651            </NamespaceSupported>
3652        <SimplePart id = "P2" mimetype = "application/xml"/>
3653        <CompositeList>
3654                <Composite mimetype = "multipart/related" id = "P3"
3655                    mimeparameters = "type=text/xml">
3656                    <Constituent idref = "P1"/>
3657                    <Constituent idref = "P2"/>
3658            </Composite>
3659        </CompositeList>
3660    </Packaging>
3661    <Packaging id="I2001">
3662            <ProcessingCapabilities parse = "true" generate = "true"/>
3663            <SimplePart id = "P11" mimetype = "text/xml"/>
3664            <SimplePart id = "P12" mimetype = "application/xml"/>
3665            <CompositeList>
3666                <Composite mimetype = "multipart/related" id = "P13"
3667                    mimeparameters = "type=text/xml">
3668                    <Constituent idref = "P11"/>
3669                    <Constituent idref = "P12"/>
3670            </Composite>
3671        </CompositeList>
3672    </Packaging>
```

3673
3674   However, it is to be expected that over time it will become possible only to assert what
3675   packaging is *generated* within each *ServiceBinding* for the requester and responder roles. This
3676   simplification assumes that each side has knowledge of what MIME types it handles correctly,
3677   what encapsulations it handles correctly, and what composition modes it handles correctly. By
3678   scanning the packaging specifications against its lists of internal capabilities, it can then look up
3679   whether other side's generated packaging scheme is one it can process and accept it under those
3680   conditions. Knowing what generated packaging style was produced by the other side could
3681   enable the software agent to propose a packaging scheme using only the MIME types and

3682   packaging styles used in the incoming *Message*. Such a packaging scheme would be likely to be
3683   acceptable to the other side when included within a proposed *CPA*. Over time, and as proposal
3684   and negotiation conventions get established, it is to be expected that the methods used for
3685   determining a match in packaging capabilities will move away from structural comparison to
3686   simpler methods, using more economical representations. For example, parsing capabilities may
3687   eventually be captured by using a compact description of the accepting grammar for the
3688   packaging and content labelling schemes that can be parsed and for which semantic handlers are
3689   available.
3690
3691   Matching Document-Level Security
3692
3693   Although the matching task for document-level security is a subtask of the Packaging-matching
3694   task, it is useful to discuss some specifics tied to the three major document-level security
3695   approaches found in [S/MIME], OpenPGP[RFC2015], and XMLDsig[XMLDSIG].
3696
3697   XMLDsig matching capability can be inferred from document-matching capabilities when the
3698   use of ebXML *Message* Service[ebMS] packaging is present. However, there are other sources
3699   that should be checked to confirm this match. A *SimplePart* element can have a
3700   *NameSpaceSupported* element. XMLDsig capability should be found there. Likewise, a detailed
3701   check on this match should examine the information under the *NonRepudiation* element and
3702   similar elements under the ebXMLBinding element to check for compatibility in hash functions
3703   and algorithms.
3704
3705   The existence of several radically different approaches to document-level security, together with
3706   the fact that it is unusual at present for a given *Party* to commit to more than one form of such
3707   security, means that there can be basic failures to match security frameworks. Therefore, there
3708   might be no match in capabilities that supports full interoperability at all levels. For the moment,
3709   we assume that document-level security matches will require both sides able to handle the same
3710   security composites (multipart/signed using S/MIME, for example.)
3711
3712   However, suppose that there are matches at the transport and transport layer security levels, but
3713   that the two sides have failures at the document-security layer because one side makes use of
3714   PGP signatures while the other uses S/MIME. Does this mean that no *CPA* can be proposed?
3715   That is not necessarily the case.
3716
3717   Both S/MIME and OpenPGP permit signatures to be packaged within "multipart/signed"
3718   composites. In such a case, it MAY be possible to extract the data and arrive at a partial
3719   implementation that falls short with respect to nonrepudiation. While neither side could check
3720   the other's signatures, it might still be possible to have confidential document transmission and
3721   transport-level authentication for the *Business* data. Eventually *CPA*-formation software MAY
3722   be created that is able to identify these exceptional situations and "salvage" a proposed *CPA* with
3723   downgraded security features. Whether the other side would accept such a proposed *CPA* would,
3724   naturally, involve what their preferences are with respect to initiating a *Business Collaboration*
3725   and sacrificing some security features. *CPA*-formation software MAY eventually be capable of
3726   these adaptations, but it is to be expected that human assistance will be required for such
3727   situations in the near term.

3728

3729   Of course, an implementation MAY simply decide to terminate looking for a *CPA* when a match
3730   fails in any crucial factor for an interoperable implementation. At the very least, the users should
3731   be warned that the only *CPAs* that can be proposed will be missing security or other normally
3732   desirable features or features recommended by the *Business Collaboration*.
3733
3734

3735   ## Other Considerations
3736   Though preferences among multiple capabilities are indicated by the document order in which
3737   they are listed, it is possible that ties may occur. At present, these ties are left to be resolved by a
3738   negotiation process not discussed here.
3739