Creating A Single Global Electronic Market

Collaboration-Protocol Profile and Agreement Specification

v1.0

Trading Partners Team

10 May 2001

(This document is the non-normative version formatted for printing, July 2001)

Table of Contents

1 Status of this Document

This document specifies an ebXML Technical 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/ebCPP.pdf

Latest version:

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

2 ebXML Participants

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

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

3 Introduction

3.1 Summary of contents of document

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

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

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

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

This specification is organized as follows:

- Section 4 defines the objectives of this specification.
- Section 5 provides a system overview.
- Section 6 contains the definition of the *CPP*, identifying the structure and all necessary fields.
- Section 7 contains the definition of the *CPA*.
- The appendices include examples of XML *CPP* and *CPA* documents (non-normative), the DTD (normative), an XML Schema document equivalent to the DTD (normative), formats of

information in the *CPP* and *CPA* (normative), and composing a *CPA* from two *CPPs* (non-normative).

3.2 Document conventions

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

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

References to external documents are represented with BLOCK text enclosed in brackets, e.g. [RFC2396]. The references are listed in Section 8, "References".

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

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

3.3 Use of XML schema

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

3.4 Version of the specification

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

3.5 Definitions

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

3.6 Audience

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

3.7 Assumptions

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

3.8 Related documents

Related documents include ebXML Specifications on the following topics:

[ebTA] ebXML Technical Architecture Specification v1.04

[ebMS] ebXML *Message* Service Specification v1.0

[ebBPSS] ebXML Business Process Specification Schema v1.01

[ebGLOSS] ebXML Glossary

[ccOVER] ebXML Core Component and Business Document Overview v1.05

[ebRS] ebXML Registry Services Specification v1.0

See Section 8 for the complete list of references.

4 Design Objectives

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

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

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

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

5 System Overview

5.1 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*.

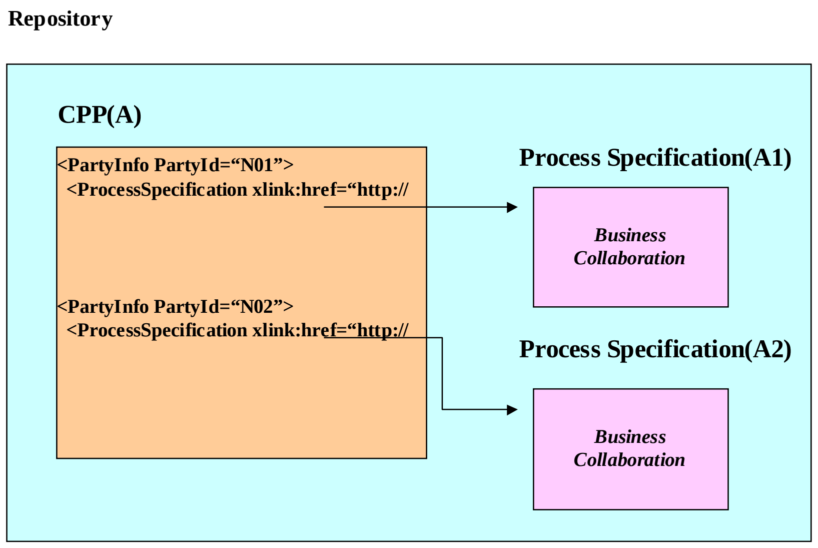
The document that defines the interactions between two *Parties* is a *Process-Specification* document that MAY conform to the ebXML Business Process Specification Schema[ebBPSS]. The *CPP* and *CPA* include references to this *Process-Specification* document. The *Process-Specification* document MAY be stored in a repository such as the ebXML Registry. See NOTE about alternative *Business-Collaboration* descriptions in section 6.5.4.

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 set up and then execute the desired *Business* information exchanges. The "legal" terms and conditions of a *Business* agreement are outside the scope of this specification and therefore are 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://
<PartyInfo PartyId="N02">
<ProcessSpecification xlink:href="http://
Process Specification(A1)
Business Collaboration
Process Specification(A2)
Business Collaboration

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

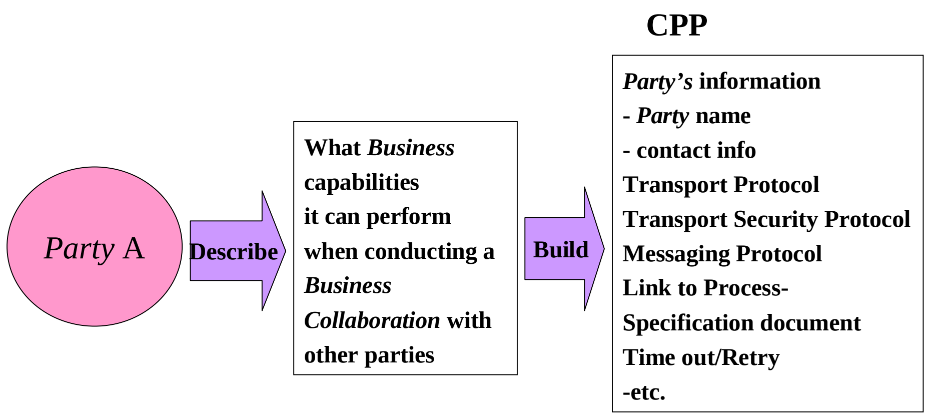
In general, the *Parties* to a *CPA* can have both client and server characteristics. A client requests services and a server provides services to the *Party* requesting services. In some applications, one *Party* only requests services and one *Party* only provides services. These applications have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

5.2 Forming a CPA from two CPPs

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

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

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.

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

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

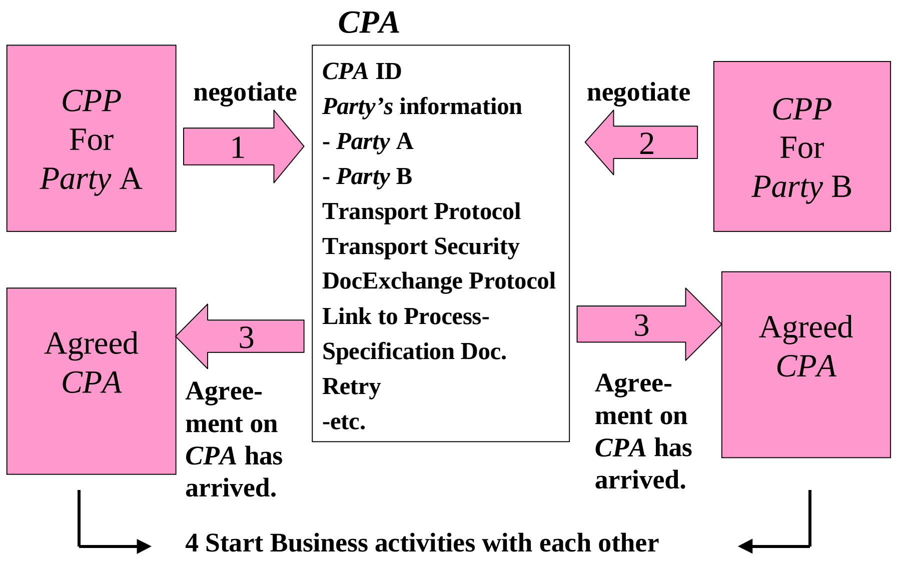


Figure 4 illustrates the entire process. The steps are listed at the left. The end of the process is that the two *Parties* configure their systems from identical copies of the agreed *CPA* and they are then ready to do *Business*.

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

1. Any *Party* may register its CPPs to an ebXML Registry.

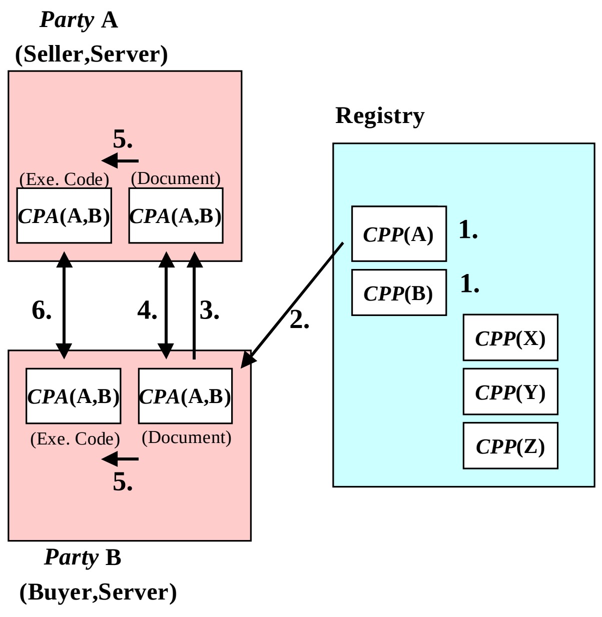
2. *Party* B discovers trading partner A (Seller) by searching in the Registry and downloads *CPP*(A) to *Party* B's server.

3. *Party* B creates *CPA*(A,B) and sends *CPA*(A,B) to *Party* A.

4. *Parties* A and B negotiate and store identical copies of the completed *CPA* as a document in both servers. This process is done manually or automatically.

5. *Parties* A and B configure their run-time systems with the information in the *CPA*.

6. *Parties* A and B do business under the new *CPA*.



Note This specification makes the assumption that a *CPP* that has been registered in an ebXML or other Registry will be referenced by some Registry-assigned globally-unique identifier that MAY be used to distinguish among multiple *CPPs* belonging to the same *Party*. See section 6.1 for more information.

5.3 How the CPA works

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

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

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

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

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

Note The run-time system SHOULD provide an interface by which the *Business* application can request initiation and ending of conversations.

5.4 Where the CPA may be implemented

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

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

5.5 Definition and scope

This specification defines and explains the contents of the *CPP* and *CPA* XML documents. Its scope is limited to these definitions. It does not define how to compose a *CPA* from two *CPPs* nor does it define anything related to run-time support for the *CPP* and *CPA*. It does include some non-normative suggestions and recommendations regarding run-time support where these notes serve to clarify the *CPP* and *CPA* definitions. See section 9 for a discussion of conformance to this specification.

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

6 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 Cand Appendix Dfor the DTD and XML Schema, respectively, and Appendix Afor 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.

6.1 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.

6.2 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>
```

6.3 CPP structure

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

```
<CollaborationProtocolProfile
      xmlns="http://www.ebxml.org/namespaces/tradePartner"
      xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      version="1.1">
      <PartyInfo>  <!--one or more-->
       ...
      </PartyInfo>
      <Packaging id="ID"> <!--one or more-->
            ...
      <Packaging>
      <ds:Signature>  <!--zero or one-->
      ...
      </ds:Signature>
      <Comment>text</Comment> <!--zero or more-->
      </CollaborationProtocolProfile>
```

6.4 CollaborationProtocolProfile element

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

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

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

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

Note The method of assigning the version-identifier value is left to the implementation.

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

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

- One REQUIRED **Packaging** element,

- Zero or one **ds:Signature** elements that contain the digital signature that signs the *CPP* document,

- Zero or more **Comment** elements.

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

6.5 PartyInfo element

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

```
<PartyInfo>
      <PartyId type="...">  <!--one or more-->
            ...
      </PartyId>
      <PartyRef xlink:type="...", xlink:href="..."/>
      <CollaborationRole>   <!--one or more-->
            ...
      </CollaborationRole>
      <Certificate>  <!--one or more-->
            ...
      </Certificate>
      <DeliveryChannel>  <!--one or more-->
      ...
      </DeliveryChannel>
      <Transport>  <!--one or more-->
      ...
      </Transport>
      <DocExchange>  <!--one or more-->
      ...
      </DocExchange>
</PartyInfo>
```

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

- One or more REQUIRED **PartyId** elements that provide a logical identifier for the organization.

- A REQUIRED **PartyRef** element that provides a pointer to more information about the *Party*.

- One or more REQUIRED **CollaborationRole** elements that identify the roles that this *Party* can play in the context of a *Process Specification*.

- One or more REQUIRED **Certificate** elements that identify the certificates used by this *Party* in security functions.

- One or more REQUIRED **DeliveryChannel** elements that define the characteristics of each delivery channel that the *Party* can use to receive *Messages*. It includes both the transport level (e.g. HTTP) and the messaging protocol (e.g. ebXML *Message* Service).

- One or more REQUIRED **Transport** elements that define the characteristics of the transport protocol(s) that the *Party* can support to receive *Messages*.

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

6.5.1 PartyId element

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

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

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

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

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

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

The following example illustrates two URI references.

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

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

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

6.5.2 PartyRef element

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

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

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

6.5.2.1 xlink:type attribute

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

6.5.2.2 xlink:href attribute

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

6.5.2.3 type attribute

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

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

```
	<PartyRef xlink:type="simple"
```

```
            xlink:href="http://example2.com/ourInfo.xml"
            type="uri-reference"/>
```

6.5.3 CollaborationRole element

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

```
<CollaborationRole id="N11" >
    <ProcessSpecification name="BuySell" version="1.0">
     ...
    </ProcessSpecification>
    <Role name="buyer" xlink:href="..."/>
    <CertificateRef certId = "N03"/>
      <!-- primary binding with "preferred" DeliveryChannel -->
    <ServiceBinding name="some process" channelId="N02"  packageId="N06">
        <!-- override "default" deliveryChannel  for selected message(s)-->
        <Override action="OrderAck" channelId="N05" packageId="N09"
            xlink:type="simple"
            xlink:href="..."/>
    </ServiceBinding>
    <!-- the first alternate binding -->
    <ServiceBinding channelId="N04" packageId="N06">
        <Override action="OrderAck" channelId="N05" packageId="N09"
      xlink:type="simple"
            xlink:href="..."/>
    </ServiceBinding>
</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.* The delivery channel selected would be chosen, based on present conditions, from those identified by **ServiceBinding** elements that refer to the *Business Collaboration* that is sending the *Message*. If more than one delivery channel is applicable, the one referred to by the highest-preference **ServiceBinding** element is used.

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

- a REQUIRED **id** attribute.

6.5.3.1 id attribute

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

6.5.3.2 CertificateRef element

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

6.5.3.3 certId attribute

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

Note This **certId** attribute relates to the authorizing role in the *Process Specification* while the certificates identified in the delivery-channel description relate to *Message* exchanges.

6.5.4 ProcessSpecification element

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

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

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

```
<ProcessSpecification
      name="BuySell"
      version="1.0"
      xlink:type="simple"
      xlink:href="http://www.ebxml.org/services/purchasing.xml"
      <ds:Reference ds:URI="http://www.ebxml.org/services/purchasing.xml">
            <ds:Transforms>
                  <ds:Transform
            ds:Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
            </ds:Transforms>
            <ds:DigestMethod
                  ds:Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1">
                  String
            </ds:DigestMethod>
            <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
      </ds:Reference>
</ProcessSpecification>
```

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

- a REQUIRED **name** attribute, with type ID,
- a REQUIRED **version** attribute,
- a FIXED **xlink:type** attribute,
- a REQUIRED **xlink:href** attribute.

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

6.5.4.1 name attribute

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

6.5.4.2 version attribute

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

6.5.4.3 xlink:type attribute

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

6.5.4.4 xlink:href attribute

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

6.5.4.5 ds:Reference element

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

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

- test of the validity of a *CPP* and the referenced *Process-Specification* documents at the time composition of a *CPA* begins in case they have changed since they were created,

- test of the validity of a *CPA* and the referenced *Process-Specification* documents at the time a *CPA* is installed into a *Party's* system,

- test of the validity of a *CPA* at intervals after the *CPA* has been installed into a *Party's* system. The *CPA* and the referenced *Process-Specification* documents MAY be processed by an installation tool into a form suited to the particular middleware. Therefore, alterations to the *CPA* and the referenced *Process-Specification* documents do not necessarily affect ongoing run-time operations. Such alterations might not be detected until it becomes necessary to reinstall the *CPA* and the referenced *Process-Specification* documents.

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

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

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

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

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

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

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

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

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

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

6.5.5 Role element

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

The **Role** element has the following attributes:

- a REQUIRED **name** attribute,
- a FIXED **xlink:type** attribute,
- a REQUIRED **xlink:href** attribute.

6.5.5.1 name attribute

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

- **name** attribute of a **BinaryCollaboration/initiatingRole** element,
- **name** attribute of a **BinaryCollaboration/respondingRole** element,
- **fromAuthorizedRole** attribute of a **BusinessTransactionActivity** element,
- **toAuthorizedRole** attribute of a **BusinessTransactionActivity** element,
- **fromAuthorizedRole** attribute of a **CollaborationActivity** element,

- **toAuthorizedRole** attribute of a **CollaborationActivity** element,
- **name** attribute of the **business-partner-role** element.

See NOTE in section 6.5.4 regarding alternative *Business-Collaboration* descriptions.

6.5.5.2 xlink:type attribute

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

6.5.5.3 xlink:href attribute

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

Xlink:href="http://www.ebxml.org/processes/purchasing#N05

Where "N05" is the value of the ID attribute of the element in the Process-Specification document that defines the role name.

6.5.6 ServiceBinding element

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

```
<ServiceBinding channelId="X03" packageId="N06">
            <Service type="string">serviceName</Service>
            <Override action="OrderAck"
                  channelId="X04"
                  packageId="N09"
                  xlink:type="simple"
                  xlink"href="..."/>  <!--zero or more-->
      </ServiceBinding>
```

The **ServiceBinding** element SHALL have one child **Service** element and zero or more **Override** child elements.

The **ServiceBinding** element has the following attributes:

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

6.5.6.1 channelId attribute

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

6.5.6.2 packageId attribute

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

6.5.7 Service element

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

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

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

6.5.7.1 type attribute

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

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

6.5.8 Override element

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

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

- a REQUIRED **action** attribute,

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

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

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

6.5.8.1 action attribute

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

See NOTE in section 6.5.4 regarding alternative *Business-Collaboration* descriptions.

6.5.8.2 channelId attribute

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

6.5.8.3 packageId attribute

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

6.5.8.4 xlink:href attribute

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

6.5.8.5 xlink:type attribute

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

6.5.9 Certificate element

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

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

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

6.5.9.1 certId attribute

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

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

6.5.9.2 ds:KeyInfo element

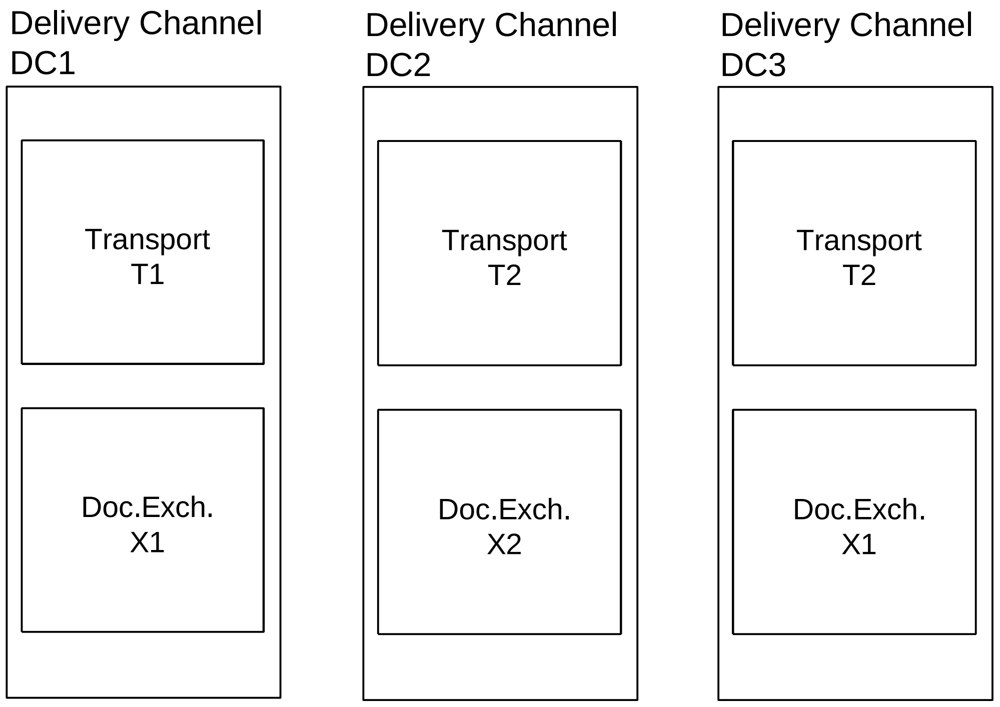
The **ds:KeyInfo** element defines the certificate information. The content of this element and any subelements are defined by the XML Digital Signature specification[XMLDSIG].

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

6.5.10 DeliveryChannel element

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

Figure 5: Three Delivery Channels



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

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

```
<DeliveryChannel channelId="N04" transportId="N05" docExchangeId="N06">
      <Characteristics
            syncReplyMode = "responseOnly"
            nonrepudiationOfOrigin = "true"
            nonrepudiationOfReceipt = "true"
            secureTransport = "true"
            confidentiality = "true"
            authenticated = "true"
            authorized = "true"/>
</DeliveryChannel>
```

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

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

- a REQUIRED **channelId** attribute,
- a REQUIRED **transportId** attribute,
- a REQUIRED **docExchangeId** attribute.

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

6.5.10.1 channelId attribute

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

6.5.10.2 transportId attribute

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

6.5.10.3 docExchangeId attribute

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

6.5.11 Characteristics element

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

See NOTE in section 6.5.4 regarding alternative *Business-Collaboration* descriptions.

The **Characteristics** element has the following attributes:

- An IMPLIED **syncReplyMode** attribute,
- an IMPLIED **nonrepudiationOfOrigin** attribute,
- an IMPLIED **nonrepudiationOfReceipt** attribute,

- an IMPLIED **secureTransport** attribute,
- an IMPLIED **confidentiality** attribute,
- an IMPLIED **authenticated** attribute,
- an IMPLIED **authorized** attribute.

6.5.11.1 syncReplyMode attribute

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

- "signalsOnly"
- "responseOnly"
- "signalsAndResponse"
- "none"

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

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

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

6.5.11.2 nonrepudiationOfOrigin attribute

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

6.5.11.3 nonrepudiationOfReceipt attribute

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

6.5.11.4 secureTransport attribute

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

6.5.11.5 confidentiality attribute

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

6.5.11.6 authenticated attribute

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

6.5.11.7 authorized attribute

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

6.5.12 Transport element

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

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

```
<Transport transportId = "N05">
      <!--protocols are HTTP, SMTP, and FTP-->
      <SendingProtocol version = "1.1">HTTP</SendingProtocol>
            <!--one or more SendingProtocol elements-->
      <ReceivingProtocol version = "1.1">HTTP</ReceivingProtocol>
      <!--one or more endpoints-->
      <Endpoint uri="http://example.com/servlet/ebxmlhandler"
            type = "request"/>
      <TransportSecurity>  <!--0 or 1 times-->
            <Protocol version = "3.0">SSL</Protocol>
            <CertificateRef certId = "N03"/>
      </TransportSecurity>
</Transport>
```

6.5.12.1 transportId attribute

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

6.5.12.2 Synchronous Responses

One distinguishing characteristic of transport protocols is whether a given transport protocol supports synchronous replies. See section 6.5.11.1 for a discussion of synchronous replies.

6.5.13 Transport protocol

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

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

6.5.13.1 SendingProtocol element

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

6.5.13.2 ReceivingProtocol element

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

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

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

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

6.5.14 Endpoint element

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

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

6.5.15 Transport protocols

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

6.5.15.1 HTTP

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

Following is an example of an HTTP endpoint:

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

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

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

6.5.15.2 SMTP

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

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

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

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

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

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

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

6.5.15.3 FTP

FTP is File Transfer Protocol[RFC959].

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

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

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

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

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

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

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

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

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

6.5.16 Transport security

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

Following is the syntax:

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

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

6.5.16.1 Protocol element

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

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

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

6.5.16.2 CertificateRef element

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

6.5.16.3 Specifics for HTTP

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

6.6 DocExchange element

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

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

```
<DocExchange docExchangeId = "N06">
      <ebXMLBinding version = "0.92">
            <ReliableMessaging>  <!--cardinality 0 or 1-->
            ...
            </ReliableMessaging>
            <NonRepudiation>  <!--cardinality 0 or 1-->
                  ...
            </NonRepudiation>
            <DigitalEnvelope>  <!--cardinality 0 or 1-->
                  ...
            </DigitalEnvelope>
            <NamespaceSupported> <!-- 1 or more -->
                  ...
            </NamespaceSupported>
      </ebXMLBinding>
</DocExchange>
```

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

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

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

6.6.1 docExchangeId attribute

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

6.6.2 ebXMLBinding element

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

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

6.6.3 version attribute

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

6.6.4 ReliableMessaging element

The **ReliableMessaging** element specifies the properties of reliable ebXML *Message* exchange. The default that applies if the **ReliableMessaging** element is omitted is "BestEffort". See Section 6.6.4.1. The following is the element structure:

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

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

- a **Retries** element,
- a **RetryInterval** element,
- a **PersistDuration** element.

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

- a REQUIRED **deliverySemantics** attribute,
- a REQUIRED **idempotency** attribute,
- an IMPLIED **messageOrderSemantics** attribute.

6.6.4.1 deliverySemantics attribute

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

- "OnceAndOnlyOnce",
- "BestEffort".

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

6.6.4.2 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*.

6.6.4.3 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*.

6.6.4.4 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*.

6.6.4.5 PersistDuration element

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

6.6.5 NonRepudiation element

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

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

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

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

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

- a REQUIRED **Protocol** element,
- a REQUIRED **HashFunction** (e.g. SHA1, MD5) element,
- a REQUIRED **SignatureAlgorithm** element,
- a REQUIRED **Certificate** element.

6.6.5.1 Protocol element

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

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

6.6.5.2 HashFunction element

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

6.6.5.3 SignatureAlgorithm element

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

6.6.5.4 CertificateRef element

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

6.6.6 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.

6.6.6.1 Protocol element

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

6.6.6.2 EncryptionAlgorithm element

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

6.6.6.3 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*.

6.6.7 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>
```

6.7 Packaging element

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

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

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

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

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

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

6.7.1 ProcessingCapabilities element

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

At least one of the **generate** or **parse** attributes MUST be true.

6.7.2 SimplePart element

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

6.7.3 SimplePart element

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

6.7.4 CompositeList element

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

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

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

- a REQUIRED **mimetype** attribute,
- a REQUIRED **id** attribute,
- an IMPLIED **mimeparameters** attribute.

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

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

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

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

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

- a REQUIRED **mimetype** attribute,
- a REQUIRED **id** attribute,
- an IMPLIED **mimeparameters** attribute.

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

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

6.8 ds:Signature element

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

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

The content of this element and any subelements are defined by the XML Digital Signature specification. See Section 7.7 for a detailed discussion. The following additional constraints on **ds:Signature** are imposed:

- A *CPP* MUST be considered invalid if any **ds:Signature** element fails core validation as defined by the XML Digital Signature specification[XMLDSIG].

- Whenever a *CPP* is signed, each **ds:Reference** element within a **ProcessSpecification** element MUST pass reference validation and each **ds:Signature** element MUST pass core validation.

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

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

Note See non-normative note in Section 6.5.4.5 for a discussion of times at which validity tests MAY be made.

6.9 Comment element

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

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

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

7 CPA Definition

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

Most of the XML elements in this section are described in detail in section 6, "CPP Definition". In general, this section does not repeat that information. The discussions in this section are limited to those elements that are not in the *CPP* or for which additional discussion is required in the *CPA* context. See also Appendix Cand Appendix Dfor the DTD and XML Schema, respectively, and Appendix Bfor an example of a *CPA* document.

7.1 CPA structure

Following is the overall structure of the *CPA:*

```
<CollaborationProtocolAgreement
      xmlns="http://www.ebxml.org/namespaces/tradePartner"
      xmlns:bpm="http://www.ebxml.org/namespaces/businessProcess"
      xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
      xmlns:xlink = "http://www.w3.org/1999/xlink"
      cpaid="YoursAndMyCPA"
      version="1.2">
      <Status value = "proposed"/>
      <Start>1988-04-07T18:39:09</Start>
      <End>1990-04-07T18:40:00</End>
      <!--ConversationConstraints MAY appear 0 or 1 times-->
      <ConversationConstraints invocationLimit = "100"
            concurrentConversations = "4"/>
      <PartyInfo>
            …
      </PartyInfo>
      <PartyInfo>
            …
      </PartyInfo>
      <Packaging id="N20"> <!--one or more-->
            ...
      </Packaging>
      <!--ds:signature MAY appear 0 or more times-->
      <ds:Signature>any combination of text and elements
      </ds:Signature>
      <Comment xml:lang="en-gb">any text</Comment> <!--zero or more-->
</CollaborationProtocolAgreement>
```

7.2 CollaborationProtocolAgreement element

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

Note Each *Party* MAY associate a local identifier with the **cpaid** attribute.

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

Note The method of assigning version identifiers is left to the implementation.

The **CollaborationProtocolAgreement** element has REQUIRED [XML] Namespace[XMLNS] declarations that are defined in Section 6, "CPP Definition".

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

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

7.3 Status element

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

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

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

- "proposed", meaning that the *CPA* is still being negotiated by the *Parties*,
- "agreed", meaning that the contents of the *CPA* have been agreed to by both *Parties*,
- "signed", meaning that the *CPA* has been "signed" by the *Parties*. This "signing" MAY take the form of a digital signature that is described in section 7.7 below.

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

7.4 CPA lifetime

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

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

7.4.1 Start element

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

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

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

7.4.2 End element

The **End** element specifies the ending date and time of the *CPA*. The **End** element SHALL be a string value that conforms to the content model of a canonical timeInstant as defined in the XML Schema Datatypes Specification[XMLSCHEMA-2]. For example, to indicate 1:20 pm UTC

(Coordinated Universal Time) on May 31, 1999, an **End** element would have the following value:

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

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

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

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

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

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

7.5 ConversationConstraints element

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

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

The **ConversationConstraints** element has the following attributes:

- an IMPLIED **invocationLimit** attribute,
- an IMPLIED **concurrentConversations** attribute.

7.5.1 invocationLimit attribute

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

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

7.5.2 concurrentConversations attribute

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

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

7.6 PartyInfo element

The general characteristics of the **PartyInfo** element are discussed in section 6.5.

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

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

7.6.1 ProcessSpecification element

The **ProcessSpecification** element identifies the *Business Collaboration* that the two *Parties* have agreed to perform. There MAY be one or more **ProcessSpecification** elements in a *CPA*. Each SHALL be a child element of a separate **CollaborationRole** element. See the discussion in Section 6.5.3.

7.7 ds:Signature element

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

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

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

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

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

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

Note See non-normative note in section 6.5.4.5 for a discussion of times at which a *CPA* MAY be validated.

7.7.1 Persistent digital signature

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

7.7.1.1 Signature Generation

Following are the steps to create a digital signature:

1. Create a **SignedInfo** element, a child element of **ds:Signature. SignedInfo** SHALL have child elements **SignatureMethod**, **CanonicalizationMethod**, and **Reference** as prescribed by [XMLDSIG].
2. Canonicalize and then calculate the **SignatureValue** over **SignedInfo** based on algorithms specified in **SignedInfo** as specified in [XMLDSIG].

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

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

7.7.1.2 ds:SignedInfo element

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

7.7.1.3 ds:CanonicalizationMethod element

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

7.7.1.4 ds:SignatureMethod element

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

http://www.w3.org/2000/09/xmldsig#dsa-sha1

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

7.7.1.5 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.

7.7.1.6 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>
```

7.7.1.7 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.

7.8 Comment element

The **CollaborationProtocolAgreement** element MAY contain zero or more **Comment** elements. See section 6.9 for details of the syntax of the **Comment** element.

7.9 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)".

7.9.1 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.

7.10 Modifying Parameters of the process-specification cocument 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.

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

8 References

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

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

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

[DIGENV] Digital Envelope, RSA Laboratories, http://www.rsasecurity.com/rsalabs/.

Note At this time, the only available specification for digital envelope appears to be the RSA Laboratories specification.

[ebBPSS] ebXML Business Process Specification Schema, http://www.ebxml.org/specs

[ebGLOSS] ebXML Glossary, http://www.ebxml.org/specs.

[ebMS] ebXML Message Service Specification, http://www.ebxml.org/specs.

[ebRS] ebXML Registry Services Specification, http://www.ebxml.org/specs.

[ebTA] ebXML Technical Architecture Specification, http://www.ebxml.org/specs.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[SSL] Secure Sockets Layer, Netscape Communications Corp. http://developer.netscape.com.

Note At this time, it appears that the Netscape specification is the only available specification of SSL. Work is in progress in IETF on "Transport Layer Security", which is intended as a replacement for SSL.

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

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

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

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

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

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

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

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

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

9 Conformance

In order to conform to this specification, an implementation:

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

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

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

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

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

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

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

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

10 Disclaimer

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

11 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

Appendix A Example of CPP Document (Non-Normative)

A text version of this schema is available on the ebXML web site at www.ebxml.org/specs/

```
      <?xml version="1.0" encoding="UTF-8"?>
<tp:CollaborationProtocolProfile

      xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"

      xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"

      xmlns:xlink="http://www.w3.org/1999/xlink"

      xmlns:ds="http://www.w3.org/2000/09/xmldsig#"

      xsi:schemaLocation="http://www.ebxml.org/namespaces/tradePartner
http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.xsd"

      tp:version="1.1">

      <tp:PartyInfo>

            <tp:PartyId tp:type="DUNS">123456789</tp:PartyId>

            <tp:PartyRef tp:href="http://example.com/about.html"/>

            <tp:CollaborationRole tp:id="N00">

                  <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
xlink:type="simple" xlink:href="http://www.ebxml.org/processes/buySell.xml"/>

                  <tp:Role tp:name="buyer" xlink:type="simple"
xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>

                  <tp:CertificateRef tp:certId="N03"/>

                  <tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402">

                        <tp:Service
tp:type="uriReference">uri:example.com/services/buyerService</tp:Service>

                        <tp:Override tp:action="orderConfirm"
tp:channelId="N07" tp:packageId="N0402"
xlink:href="http://ebxml.org/processes/buySell.xml#orderConfirm"
xlink:type="simple"/>

                  </tp:ServiceBinding>
            </tp:CollaborationRole>

            <tp:Certificate tp:certId="N03">

                  <ds:KeyInfo/>

            </tp:Certificate>

            <tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
tp:docExchangeId="N06">

                  <tp:Characteristics tp:syncReplyMode="none"
tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
tp:secureTransport="true" tp:confidentiality="true" tp:authenticated="true"
tp:authorized="false"/>

            </tp:DeliveryChannel>

            <tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
tp:docExchangeId="N06">

                  <tp:Characteristics tp:syncReplyMode="none"
tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
tp:secureTransport="false" tp:confidentiality="true" tp:authenticated="true"
tp:authorized="false"/>

            </tp:DeliveryChannel>

            <tp:Transport tp:transportId="N05">

                  <tp:SendingProtocol
tp:version="1.1">HTTP</tp:SendingProtocol>

                  <tp:ReceivingProtocol
tp:version="1.1">HTTP</tp:ReceivingProtocol>

                  <tp:Endpoint
tp:uri="https://www.example.com/servlets/ebxmlhandler" tp:type="allPurpose"/>

                  <tp:TransportSecurity>

                        <tp:Protocol tp:version="3.0">SSL</tp:Protocol>

                        <tp:CertificateRef tp:certId="N03"/>

                  </tp:TransportSecurity>

            </tp:Transport>

            <tp:Transport tp:transportId="N08">

                  <tp:SendingProtocol
tp:version="1.1">HTTP</tp:SendingProtocol>
```

```
                  <tp:ReceivingProtocol
tp:version="1.1">SMTP</tp:ReceivingProtocol>

                  <tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
tp:type="allPurpose"/>

            </tp:Transport>

            <tp:DocExchange tp:docExchangeId="N06">

                  <tp:ebXMLBinding tp:version="0.98b">

                        <tp:ReliableMessaging
tp:deliverySemantics="OnceAndOnlyOnce" tp:idempotency="true"
tp:messageOrderSemantics="Guaranteed">

                              <tp:Retries>5</tp:Retries>

                              <tp:RetryInterval>30</tp:RetryInterval>

                              <tp:PersistDuration>P1D</tp:PersistDuration>

                        </tp:ReliableMessaging>

                        <tp:NonRepudiation>


      <tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>


      <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunctio
n>


      <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
sha1</tp:SignatureAlgorithm>

                              <tp:CertificateRef tp:certId="N03"/>

                        </tp:NonRepudiation>

                        <tp:DigitalEnvelope>

                              <tp:Protocol
tp:version="2.0">S/MIME</tp:Protocol>

                              <tp:EncryptionAlgorithm>DES-
CBC</tp:EncryptionAlgorithm>

                              <tp:CertificateRef tp:certId="N03"/>

                        </tp:DigitalEnvelope>

                  </tp:ebXMLBinding>
            </tp:DocExchange>

      </tp:PartyInfo>

      <tp:Packaging tp:id="N0402">

            <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>

            <tp:SimplePart tp:id="N40" tp:mimetype="text/xml">

                  <tp:NamespaceSupported
tp:location="http://ebxml.org/project_teams/transport/messageService.xsd"
tp:version="0.98b">http://www.ebxml.org/namespaces/messageService</tp:Namespa
ceSupported>

                  <tp:NamespaceSupported
tp:location="http://ebxml.org/project_teams/transport/xmldsig-core-
schema.xsd"
tp:version="1.0">http://www.w3.org/2000/09/xmldsig</tp:NamespaceSupported>

            </tp:SimplePart>

            <tp:SimplePart tp:id="N41" tp:mimetype="text/xml">

                  <tp:NamespaceSupported
tp:location="http://ebxml.org/processes/buysell.xsd"
tp:version="1.0">http://ebxml.org/processes/buysell.xsd</tp:NamespaceSupporte
d>

            </tp:SimplePart>

            <tp:CompositeList>

                  <tp:Composite tp:id="N42" tp:mimetype="multipart/related"
tp:mimeparameters="type=text/xml;">

                        <tp:Constituent tp:idref="N40"/>

                        <tp:Constituent tp:idref="N41"/>

                  </tp:Composite>

            </tp:CompositeList>

      </tp:Packaging>

      <tp:Comment tp:xml_lang="en-us">buy/sell agreement between example.com
and contrived-example.com</tp:Comment>

</tp:CollaborationProtocolProfile>
```

Appendix B Example of CPA Document (Non-Normative)

The example in this appendix is to be parsed with an XML Schema parser.

A text version of this schema is available on the ebXML web site at www.ebxml.org/specs/

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

```
<?xml version="1.0"?>

<!-- edited with XML Spy v3.5 (http://www.xmlspy.com) by christopher ferris
(sun microsystems, inc) -->

<tp:CollaborationProtocolAgreement

      xmlns:tp="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-v1_0.xsd"

      xmlns:xlink="http://www.w3.org/1999/xlink"

      xmlns:ds="http://www.w3.org/2000/09/xmldsig#"

      tp:cpaid="uri:yoursandmycpa"

      tp:version="1.2">

      <tp:Status tp:value="proposed"/>

      <tp:Start>2001-05-20T07:21:00Z</tp:Start>

      <tp:End>2002-05-20T07:21:00Z</tp:End>

      <tp:ConversationConstraints tp:invocationLimit="100"
tp:concurrentConversations="100"/>

      <tp:PartyInfo>

            <tp:PartyId tp:type="DUNS">123456789</tp:PartyId>

            <tp:PartyRef xlink:href="http://example.com/about.html"/>
            <tp:CollaborationRole tp:id="N00">

                  <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
xlink:type="simple" xlink:href="http://www.ebxml.org/processes/buySell.xml"/>

                  <tp:Role tp:name="buyer" xlink:type="simple"
xlink:href="http://ebxml.org/processes/buySell.xml#buyer"/>

                  <tp:CertificateRef tp:certId="N03"/>

                  <tp:ServiceBinding tp:channelId="N04" tp:packageId="N0402">

                        <tp:Service
tp:type="uriReference">uri:example.com/services/buyerService</tp:Service>

                        <tp:Override tp:action="orderConfirm"
tp:channelId="N08" tp:packageId="N0402"
xlink:href="http://ebxml.org/processes/buySell.xml#orderConfirm"
xlink:type="simple"/>

                  </tp:ServiceBinding>

            </tp:CollaborationRole>

            <tp:Certificate tp:certId="N03">

                  <ds:KeyInfo/>

            </tp:Certificate>

            <tp:DeliveryChannel tp:channelId="N04" tp:transportId="N05"
tp:docExchangeId="N06">

                  <tp:Characteristics tp:syncReplyMode="none"
tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
tp:secureTransport="true" tp:confidentiality="true" tp:authenticated="true"
tp:authorized="false"/>

            </tp:DeliveryChannel>

            <tp:DeliveryChannel tp:channelId="N07" tp:transportId="N08"
tp:docExchangeId="N06">

                  <tp:Characteristics tp:syncReplyMode="none"
tp:nonrepudiationOfOrigin="true" tp:nonrepudiationOfReceipt="false"
tp:secureTransport="false" tp:confidentiality="true" tp:authenticated="true"
tp:authorized="false"/>

            </tp:DeliveryChannel>

            <tp:Transport tp:transportId="N05">

                  <tp:SendingProtocol
tp:version="1.1">HTTP</tp:SendingProtocol>
```

```
                  <tp:ReceivingProtocol
tp:version="1.1">HTTP</tp:ReceivingProtocol>

                  <tp:Endpoint
tp:uri="https://www.example.com/servlets/ebxmlhandler" tp:type="allPurpose"/>

                  <tp:TransportSecurity>

                        <tp:Protocol tp:version="3.0">SSL</tp:Protocol>

                        <tp:CertificateRef tp:certId="N03"/>

                  </tp:TransportSecurity>

            </tp:Transport>

            <tp:Transport tp:transportId="N18">

                  <tp:SendingProtocol
tp:version="1.1">HTTP</tp:SendingProtocol>

                  <tp:ReceivingProtocol
tp:version="1.1">SMTP</tp:ReceivingProtocol>

                  <tp:Endpoint tp:uri="mailto:ebxmlhandler@example.com"
tp:type="allPurpose"/>

            </tp:Transport>

            <tp:DocExchange tp:docExchangeId="N06">

                  <tp:ebXMLBinding tp:version="0.98b">

                        <tp:ReliableMessaging
tp:deliverySemantics="OnceAndOnlyOnce" tp:idempotency="true"
tp:messageOrderSemantics="Guaranteed">

                              <tp:Retries>5</tp:Retries>

                              <tp:RetryInterval>30</tp:RetryInterval>

                              <tp:PersistDuration>P1D</tp:PersistDuration>

                        </tp:ReliableMessaging>

                        <tp:NonRepudiation>


      <tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>


      <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunctio
n>
```

```
      <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
sha1</tp:SignatureAlgorithm>

                              <tp:CertificateRef tp:certId="N03"/>

                        </tp:NonRepudiation>

                        <tp:DigitalEnvelope>

                              <tp:Protocol
tp:version="2.0">S/MIME</tp:Protocol>

                              <tp:EncryptionAlgorithm>DES-
CBC</tp:EncryptionAlgorithm>

                              <tp:CertificateRef tp:certId="N03"/>

                        </tp:DigitalEnvelope>

                  </tp:ebXMLBinding>

            </tp:DocExchange>

      </tp:PartyInfo>

      <tp:PartyInfo>

            <tp:PartyId tp:type="DUNS">987654321</tp:PartyId>

            <tp:PartyRef xlink:type="simple" xlink:href="http://contrived-
example.com/about.html"/>

            <tp:CollaborationRole tp:id="N30">

                  <tp:ProcessSpecification tp:version="1.0" tp:name="buySell"
xlink:type="simple" xlink:href="http://www.ebxml.org/processes/buySell.xml"/>

                  <tp:Role tp:name="seller" xlink:type="simple"
xlink:href="http://ebxml.org/processes/buySell.xml#seller"/>

                  <tp:CertificateRef tp:certId="N33"/>

                  <tp:ServiceBinding tp:channelId="N34" tp:packageId="N0402">

                        <tp:Service
tp:type="uriReference">uri:example.com/services/sellerService</tp:Service>

                  </tp:ServiceBinding>

            </tp:CollaborationRole>

            <tp:Certificate tp:certId="N33">
```

```
                  <ds:KeyInfo/>

            </tp:Certificate>

            <tp:DeliveryChannel tp:channelId="N34" tp:transportId="N35"
tp:docExchangeId="N36">

                  <tp:Characteristics tp:nonrepudiationOfOrigin="true"
tp:nonrepudiationOfReceipt="false" tp:secureTransport="true"
tp:confidentiality="true" tp:authenticated="true" tp:authorized="false"/>

            </tp:DeliveryChannel>

            <tp:Transport tp:transportId="N35">

                  <tp:SendingProtocol
tp:version="1.1">HTTP</tp:SendingProtocol>

                  <tp:ReceivingProtocol
tp:version="1.1">HTTP</tp:ReceivingProtocol>

                  <tp:Endpoint tp:uri="https://www.contrived-
example.com/servlets/ebxmlhandler" tp:type="allPurpose"/>

                  <tp:TransportSecurity>

                        <tp:Protocol tp:version="3.0">SSL</tp:Protocol>

                        <tp:CertificateRef tp:certId="N33"/>

                  </tp:TransportSecurity>

            </tp:Transport>

            <tp:DocExchange tp:docExchangeId="N36">

                  <tp:ebXMLBinding tp:version="0.98b">

                        <tp:ReliableMessaging
tp:deliverySemantics="OnceAndOnlyOnce" tp:idempotency="true"
tp:messageOrderSemantics="Guaranteed">

                              <tp:Retries>5</tp:Retries>

                              <tp:RetryInterval>30</tp:RetryInterval>

                              <tp:PersistDuration>P1D</tp:PersistDuration>

                        </tp:ReliableMessaging>

                        <tp:NonRepudiation>


      <tp:Protocol>http://www.w3.org/2000/09/xmldsig#</tp:Protocol>
      <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1</tp:HashFunctio
n>

      <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
sha1</tp:SignatureAlgorithm>

                              <tp:CertificateRef tp:certId="N33"/>

                        </tp:NonRepudiation>

                        <tp:DigitalEnvelope>

                              <tp:Protocol
tp:version="2.0">S/MIME</tp:Protocol>

                              <tp:EncryptionAlgorithm>DES-
CBC</tp:EncryptionAlgorithm>

                              <tp:CertificateRef tp:certId="N33"/>

                        </tp:DigitalEnvelope>

                  </tp:ebXMLBinding>

            </tp:DocExchange>

      </tp:PartyInfo>

      <tp:Packaging tp:id="N0402">

            <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>

            <tp:SimplePart tp:id="N40" tp:mimetype="text/xml">

                  <tp:NamespaceSupported
tp:location="http://ebxml.org/project_teams/transport/messageService.xsd"
tp:version="0.98b">http://www.ebxml.org/namespaces/messageService</tp:Namespa
ceSupported>

                  <tp:NamespaceSupported
tp:location="http://ebxml.org/project_teams/transport/xmldsig-core-
schema.xsd"
tp:version="1.0">http://www.w3.org/2000/09/xmldsig</tp:NamespaceSupported>

            </tp:SimplePart>

            <tp:SimplePart tp:id="N41" tp:mimetype="text/xml">

                  <tp:NamespaceSupported
tp:location="http://ebxml.org/processes/buysell.xsd"
tp:version="1.0">http://ebxml.org/processes/buysell.xsd</tp:NamespaceSupporte
d>
```

```
            </tp:SimplePart>

            <tp:CompositeList>

                  <tp:Composite tp:id="N42" tp:mimetype="multipart/related"
tp:mimeparameters="type=text/xml;">

                        <tp:Constituent tp:idref="N40"/>

                        <tp:Constituent tp:idref="N41"/>

                  </tp:Composite>

            </tp:CompositeList>

      </tp:Packaging>

      <tp:Comment xml:lang="en-us">buy/sell agreement between example.com and
contrived-example.com</tp:Comment>

</tp:CollaborationProtocolAgreement>
```

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

A text version of this schema is available on the ebXML web site at www.ebxml.org/specs/

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Generated by XML Authority-->
<!ELEMENT CollaborationProtocolAgreement (Status, Start, End,
ConversationConstraints?, PartyInfo+, Packaging, ds:Signature*, Comment*)>
<!ATTLIST CollaborationProtocolAgreement
      cpaid CDATA #IMPLIED
      version CDATA #IMPLIED
>
<!ELEMENT CollaborationProtocolProfile (PartyInfo+, Packaging, ds:Signature?,
Comment*)>
<!ATTLIST CollaborationProtocolProfile
      version CDATA #IMPLIED
>
<!ELEMENT ProcessSpecification (ds:Reference?)>
<!ATTLIST ProcessSpecification
      version CDATA #REQUIRED
      name CDATA #REQUIRED
      xlink:type CDATA #FIXED "simple"
      xlink:href CDATA #IMPLIED
>
<!ELEMENT Protocol (#PCDATA)>
<!ATTLIST Protocol
      version CDATA #IMPLIED
>
<!ELEMENT SendingProtocol (#PCDATA)>
<!ATTLIST SendingProtocol
      version CDATA #IMPLIED
>
<!ELEMENT ReceivingProtocol (#PCDATA)>
<!ATTLIST ReceivingProtocol
      version CDATA #IMPLIED
>
<!ELEMENT CollaborationRole (ProcessSpecification, Role, CertificateRef?,
ServiceBinding+)>
<!ATTLIST CollaborationRole
      id ID #IMPLIED
>
<!ELEMENT PartyInfo (PartyId+, PartyRef, CollaborationRole+, Certificate+,
DeliveryChannel+, Transport+, DocExchange+)>
<!ELEMENT PartyId (#PCDATA)>
<!ATTLIST PartyId
      type CDATA #IMPLIED
```

```
>
<!ELEMENT PartyRef EMPTY>
<!ATTLIST PartyRef
      xlink:type (simple) #IMPLIED
      xlink:href CDATA #IMPLIED
>
<!ELEMENT DeliveryChannel (Characteristics)>
<!ATTLIST DeliveryChannel
      channelId ID #REQUIRED
      transportId IDREF #REQUIRED
      docExchangeId IDREF #REQUIRED
>
<!ELEMENT Transport (SendingProtocol+, ReceivingProtocol, Endpoint+,
TransportSecurity?)>
<!ATTLIST Transport
      transportId ID #REQUIRED
>
<!ELEMENT Endpoint EMPTY>
<!ATTLIST Endpoint
      uri CDATA #REQUIRED
      type (login | request | response | error | allPurpose) "allPurpose"
>
<!ELEMENT Retries (#PCDATA)>
<!ELEMENT RetryInterval (#PCDATA)>
<!ELEMENT TransportSecurity (Protocol, CertificateRef?)>
<!ELEMENT Certificate (ds:KeyInfo)>
<!ATTLIST Certificate
      certId ID #REQUIRED
>
<!ELEMENT DocExchange (ebXMLBinding)>
<!ATTLIST DocExchange
      docExchangeId ID #REQUIRED
>
<!ELEMENT PersistDuration (#PCDATA)>
<!ATTLIST PersistDuration
      e-dtype NMTOKEN #FIXED "timeDuration"
>
<!ELEMENT ReliableMessaging (Retries, RetryInterval, PersistDuration)?>
<!ATTLIST ReliableMessaging
      deliverySemantics (OnceAndOnlyOnce | BestEffort) #REQUIRED
      messageOrderSemantics (Guaranteed | NotGuaranteed) "NotGuaranteed"
      idempotency CDATA #REQUIRED
>
<!ELEMENT NonRepudiation (Protocol, HashFunction, SignatureAlgorithm,
CertificateRef)>
<!ELEMENT HashFunction (#PCDATA)>
<!ELEMENT EncryptionAlgorithm (#PCDATA)>
<!ELEMENT SignatureAlgorithm (#PCDATA)>
<!ELEMENT DigitalEnvelope (Protocol, EncryptionAlgorithm, CertificateRef)>
<!ELEMENT CertificateRef EMPTY>
<!ATTLIST CertificateRef
      certId IDREF #REQUIRED
```

```
>
<!ELEMENT ebXMLBinding (ReliableMessaging?, NonRepudiation?,
DigitalEnvelope?, NamespaceSupported*)>
<!ATTLIST ebXMLBinding
      version CDATA #REQUIRED
>
<!ELEMENT NamespaceSupported (#PCDATA)>
<!ATTLIST NamespaceSupported
      location CDATA #REQUIRED
      version CDATA #IMPLIED
>
<!ELEMENT Characteristics EMPTY>
<!ATTLIST Characteristics
      syncReplyMode (responseOnly | signalsAndResponse | signalsOnly | none)
#IMPLIED
      nonrepudiationOfOrigin CDATA #IMPLIED
      nonrepudiationOfReceipt CDATA #IMPLIED
      secureTransport CDATA #IMPLIED
      confidentiality CDATA #IMPLIED
      authenticated CDATA #IMPLIED
      authorized CDATA #IMPLIED
>
<!ELEMENT ServiceBinding (Service, Override*)>
<!ATTLIST ServiceBinding
      channelId IDREF #REQUIRED
      packageId IDREF #REQUIRED
>
<!ELEMENT Service (#PCDATA)>
<!ATTLIST Service
      type CDATA #IMPLIED>

<!ELEMENT Status EMPTY>
<!ATTLIST Status
      value (agreed | signed | proposed) #REQUIRED
>
<!ELEMENT Start (#PCDATA)>
<!ELEMENT End (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT ConversationConstraints EMPTY>
<!ATTLIST ConversationConstraints
      invocationLimit CDATA #IMPLIED
      concurrentConversations CDATA #IMPLIED
>
<!ELEMENT Override EMPTY>
<!ATTLIST Override
      action CDATA #REQUIRED
      channelId ID #REQUIRED
      packageId IDREF #REQUIRED
      xlink:href CDATA #IMPLIED
      xlink:type CDATA #FIXED "simple"
>
<!ELEMENT Role EMPTY>
```

```
<!ATTLIST Role
      name CDATA #REQUIRED
      xlink:type CDATA #FIXED "simple"
      xlink:href CDATA #IMPLIED
>
<!ELEMENT Constituent EMPTY>
<!ATTLIST Constituent
      idref CDATA #REQUIRED
>
<!ELEMENT ProcessingCapabilities EMPTY>
<!ATTLIST ProcessingCapabilities
      parse CDATA #REQUIRED
      generate CDATA #REQUIRED
>
<!ELEMENT SimplePart (NamespaceSupported*)>
<!ATTLIST SimplePart
      id ID #IMPLIED
      mimetype CDATA #REQUIRED
>
<!ELEMENT Encapsulation (Constituent)>
<!ATTLIST Encapsulation
      id ID #IMPLIED
      mimetype CDATA #REQUIRED
      mimeparameters CDATA #IMPLIED
>
<!ELEMENT Composite (Constituent+)>
<!ATTLIST Composite
      id ID #IMPLIED
      mimetype CDATA #REQUIRED
      mimeparameters CDATA #IMPLIED
>
<!ELEMENT CompositeList (Encapsulation | Composite)+>
<!ELEMENT Packaging (ProcessingCapabilities, SimplePart+, CompositeList?)>
<!ATTLIST Packaging
      id ID #REQUIRED
>
<!ELEMENT Comment (#PCDATA)>
<!ATTLIST Comment
      xml:lang CDATA #REQUIRED
>
<!ELEMENT ds:Signature ANY>
<!ELEMENT ds:Reference ANY>
<!ELEMENT ds:KeyInfo ANY>
```

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

A text version of this schema is available on the ebXML web site at www.ebxml.org/specs/

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ebxml.org/namespaces/tradePartner"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:tns="http://www.ebxml.org/namespaces/tradePartner"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="1.0">
      <import namespace="http://www.w3.org/1999/xlink"
schemaLocation="http://ebxml.org/project_teams/transport/xlink.xsd"/>
      <import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://ebxml.org/project_teams/transport/xmldsig-core-
schema.xsd"/>
      <import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://ebxml.org/project_teams/transport/xml_lang.xsd"/>
      <attributeGroup name="pkg.grp">
            <attribute ref="tns:id"/>
            <attribute name="mimetype" type="tns:non-empty-string"
use="required"/>
            <attribute name="mimeparameters" type="tns:non-empty-string"/>
      </attributeGroup>
      <attributeGroup name="xlink.grp">
            <attribute ref="xlink:type"/>
            <attribute ref="xlink:href"/>
      </attributeGroup>
      <element name="CollaborationProtocolAgreement">
            <complexType>
                  <sequence>
                        <element ref="tns:Status"/>
                        <element ref="tns:Start"/>
                        <element ref="tns:End"/>
                        <element ref="tns:ConversationConstraints"
minOccurs="0"/>
                        <element ref="tns:PartyInfo" maxOccurs="unbounded"/>
                        <element ref="tns:Packaging"/>
                        <element ref="ds:Signature" minOccurs="0"
maxOccurs="unbounded"/>
```

```
                        <element ref="tns:Comment" minOccurs="0"
maxOccurs="unbounded"/>
                  </sequence>
                  <attribute name="cpaid" type="tns:non-empty-string"/>
                  <attribute ref="tns:version"/>
                  <anyAttribute namespace="##targetNamespace
http://www.w3.org/2000/10/XMLSchema-instance" processContents="lax"/>
            </complexType>
      </element>
      <element name="CollaborationProtocolProfile">
            <complexType>
                  <sequence>
                        <element ref="tns:PartyInfo" maxOccurs="unbounded"/>
                        <element ref="tns:Packaging"/>
                        <element ref="ds:Signature" minOccurs="0"/>
                        <element ref="tns:Comment" minOccurs="0"
maxOccurs="unbounded"/>
                  </sequence>
                  <attribute ref="tns:version"/>
                  <anyAttribute namespace="##targetNamespace
http://www.w3.org/2000/10/XMLSchema-instance" processContents="lax"/>
            </complexType>
      </element>
      <element name="ProcessSpecification">
            <complexType>
                  <sequence>
                        <element ref="ds:Reference" minOccurs="0"/>
                  </sequence>
                  <attribute ref="tns:version"/>
                  <attribute name="name" type="tns:non-empty-string"
use="required"/>
                  <attributeGroup ref="tns:xlink.grp"/>
            </complexType>
      </element>
      <element name="Service" type="tns:service.type"/>
      <element name="Protocol" type="tns:protocol.type"/>
      <element name="SendingProtocol" type="tns:protocol.type"/>
      <element name="ReceivingProtocol" type="tns:protocol.type"/>
      <element name="CollaborationRole">
            <complexType>
                  <sequence>
                        <element ref="tns:ProcessSpecification"/>
                        <element ref="tns:Role"/>
                        <element ref="tns:CertificateRef" minOccurs="0"/>
                        <element ref="tns:ServiceBinding"
maxOccurs="unbounded"/>
                  </sequence>
                  <attribute ref="tns:id"/>
            </complexType>
      </element>
      <element name="PartyInfo">
            <complexType>
```

```
                  <sequence>
                        <element ref="tns:PartyId" maxOccurs="unbounded"/>
                        <element ref="tns:PartyRef"/>
                        <element ref="tns:CollaborationRole"
maxOccurs="unbounded"/>
                        <element ref="tns:Certificate"
maxOccurs="unbounded"/>
                        <element ref="tns:DeliveryChannel"
maxOccurs="unbounded"/>
                        <element ref="tns:Transport" maxOccurs="unbounded"/>
                        <element ref="tns:DocExchange"
maxOccurs="unbounded"/>
                  </sequence>
            </complexType>
      </element>
      <element name="PartyId">
            <complexType>
                  <simpleContent>
                        <extension base="tns:non-empty-string">
                              <attribute name="type" type="tns:non-empty-
string"/>
                        </extension>
                  </simpleContent>
            </complexType>
      </element>
      <element name="PartyRef">
            <complexType>
                  <attributeGroup ref="tns:xlink.grp"/>
                  <attribute name="type" type="tns:non-empty-string"/>
            </complexType>
      </element>
      <element name="DeliveryChannel">
            <complexType>
                  <sequence>
                        <element ref="tns:Characteristics"/>
                  </sequence>
                  <attribute name="channelId" type="ID" use="required"/>
                  <attribute name="transportId" type="IDREF" use="required"/>
                  <attribute name="docExchangeId" type="IDREF"
use="required"/>
            </complexType>
      </element>
      <element name="Transport">
            <complexType>
                  <sequence>
                        <element ref="tns:SendingProtocol"
maxOccurs="unbounded"/>
                        <element ref="tns:ReceivingProtocol"/>
                        <element ref="tns:Endpoint" maxOccurs="unbounded"/>
                        <element ref="tns:TransportSecurity" minOccurs="0"/>
                  </sequence>
                  <attribute name="transportId" type="ID" use="required"/>
            </complexType>
      </element>
      <element name="Endpoint">
            <complexType>
                  <attribute name="uri" type="uriReference" use="required"/>
                  <attribute name="type" type="tns:endpointType.type"
use="default" value="allPurpose"/>
            </complexType>
      </element>
      <element name="Retries" type="string"/>
      <element name="RetryInterval" type="string"/>
      <element name="TransportSecurity">
            <complexType>
                  <sequence>
                        <element ref="tns:Protocol"/>
                        <element ref="tns:CertificateRef" minOccurs="0"/>
                  </sequence>
            </complexType>
      </element>
      <element name="Certificate">
            <complexType>
                  <sequence>
                        <element ref="ds:KeyInfo"/>
                  </sequence>
                  <attribute name="certId" type="ID" use="required"/>
            </complexType>
      </element>
      <element name="DocExchange">
            <complexType>
                  <sequence>
                        <element ref="tns:ebXMLBinding"/>
                  </sequence>
                  <attribute name="docExchangeId" type="ID" use="required"/>
            </complexType>
      </element>
      <element name="ReliableMessaging">
            <complexType>
                  <sequence minOccurs="0">
                        <element ref="tns:Retries"/>
                        <element ref="tns:RetryInterval"/>
                        <element name="PersistDuration" type="timeDuration"/>
                  </sequence>
                  <attribute name="deliverySemantics" type="tns:ds.type"
use="required"/>
                  <attribute name="idempotency" type="boolean"
use="required"/>
                  <attribute name="messageOrderSemantics" type="tns:mos.type"
use="optional" value="NotGuaranteed"/>
            </complexType>
            <!-- <element name="PersistDuration" type="duration"/> -->
      </element>
      <element name="NonRepudiation">
```

```
            <complexType>
                  <sequence>
                        <element ref="tns:Protocol"/>
                        <element ref="tns:HashFunction"/>
                        <element ref="tns:SignatureAlgorithm"/>
                        <element ref="tns:CertificateRef"/>
                  </sequence>
            </complexType>
      </element>
      <element name="HashFunction" type="string"/>
      <element name="EncryptionAlgorithm" type="string"/>
      <element name="SignatureAlgorithm" type="string"/>
      <element name="DigitalEnvelope">
            <complexType>
                  <sequence>
                        <element ref="tns:Protocol"/>
                        <element ref="tns:EncryptionAlgorithm"/>
                        <element ref="tns:CertificateRef"/>
                  </sequence>
            </complexType>
      </element>
      <element name="CertificateRef">
            <complexType>
                  <attribute name="certId" type="IDREF" use="required"/>
            </complexType>
      </element>
      <element name="ebXMLBinding">
            <complexType>
                  <sequence>
                        <element ref="tns:ReliableMessaging" minOccurs="0"/>
                        <element ref="tns:NonRepudiation" minOccurs="0"/>
                        <element ref="tns:DigitalEnvelope" minOccurs="0"/>
                        <element ref="tns:NamespaceSupported" minOccurs="0"
maxOccurs="unbounded"/>
                  </sequence>
                  <attribute ref="tns:version"/>
            </complexType>
      </element>
      <element name="NamespaceSupported">
            <complexType>
                  <simpleContent>
                        <extension base="uriReference">
                              <attribute name="location" type="uriReference"
use="required"/>
                              <attribute ref="tns:version"/>
                        </extension>
                  </simpleContent>
            </complexType>
      </element>
      <element name="Characteristics">
            <complexType>
                  <attribute ref="tns:syncReplyMode"/>
                  <attribute name="nonrepudiationOfOrigin" type="boolean"/>
                  <attribute name="nonrepudiationOfReceipt" type="boolean"/>
                  <attribute name="secureTransport" type="boolean"/>
                  <attribute name="confidentiality" type="boolean"/>
                  <attribute name="authenticated" type="boolean"/>
                  <attribute name="authorized" type="boolean"/>
            </complexType>
      </element>
      <element name="ServiceBinding">
            <complexType>
                  <sequence>
                        <element ref="tns:Service"/>
                        <element ref="tns:Override" minOccurs="0"
maxOccurs="unbounded"/>
                  </sequence>
                  <attribute name="channelId" type="IDREF" use="required"/>
                  <attribute name="packageId" type="IDREF" use="required"/>
            </complexType>
            <unique name="action.const">
                  <selector xpath=".//Override"/>
                  <field xpath="@action"/>
            </unique>
      </element>
      <element name="Status">
            <complexType>
                  <attribute name="value" type="tns:statusValue.type"
use="required"/>
            </complexType>
      </element>
      <element name="Start" type="timeInstant"/>
      <element name="End" type="timeInstant"/>
      <!--
      <element name="Start" type="dateTime"/>
      <element name="End" type="dateTime"/>
      -->
      <element name="Type" type="string"/>
      <element name="ConversationConstraints">
            <complexType>
                  <attribute name="invocationLimit" type="int"/>
                  <attribute name="concurrentConversations" type="int"/>
            </complexType>
      </element>
      <element name="Override">
            <complexType>
                  <attribute name="action" type="tns:non-empty-string"
use="required"/>
                  <attribute name="channelId" type="ID" use="required"/>
                  <attribute name="packageId" type="IDREF" use="required"/>
                  <attributeGroup ref="tns:xlink.grp"/>
            </complexType>
      </element>
      <element name="Role">
```

```
            <complexType>
                  <attribute name="name" type="tns:non-empty-string"
use="required"/>
                  <attributeGroup ref="tns:xlink.grp"/>
            </complexType>
      </element>
      <element name="Constituent">
            <complexType>
                  <attribute ref="tns:idref"/>
            </complexType>
      </element>
      <element name="Packaging">
            <complexType>
                  <sequence>
                        <element name="ProcessingCapabilities">
                              <complexType>
                                    <attribute name="parse" type="boolean"
use="required"/>
                                    <attribute name="generate" type="boolean"
use="required"/>
                              </complexType>
                        </element>
                        <element name="SimplePart" maxOccurs="unbounded">
                              <complexType>
                                    <sequence>
                                          <element
ref="tns:NamespaceSupported" minOccurs="0" maxOccurs="unbounded"/>
                                    </sequence>
                                    <attributeGroup ref="tns:pkg.grp"/>
                              </complexType>
                        </element>
                        <element name="CompositeList" minOccurs="0">
                              <complexType>
                                    <choice maxOccurs="unbounded">
                                          <element name="Encapsulation">
                                                <complexType>
                                                      <sequence>
                                                            <element
ref="tns:Constituent"/>
                                                      </sequence>
                                                      <attributeGroup
ref="tns:pkg.grp"/>
                                                </complexType>
                                          </element>
                                          <element name="Composite">
                                                <complexType>
                                                      <sequence>
                                                            <element
ref="tns:Constituent" maxOccurs="unbounded"/>
                                                      </sequence>
                                                      <attributeGroup
ref="tns:pkg.grp"/>
```

```
                                          </complexType>
                                    </element>
                              </choice>
                        </complexType>
                  </element>
            </sequence>
            <attribute ref="tns:id"/>
      </complexType>
</element>
<element name="Comment">
      <complexType>
            <simpleContent>
                  <extension base="tns:non-empty-string">
                        <attribute ref="xml:lang"/>
                  </extension>
            </simpleContent>
      </complexType>
</element>
<!-- COMMON -->
<simpleType name="ds.type">
      <restriction base="NMTOKEN">
            <enumeration value="OnceAndOnlyOnce"/>
            <enumeration value="BestEffort"/>
      </restriction>
</simpleType>
<simpleType name="mos.type">
      <restriction base="NMTOKEN">
            <enumeration value="Guaranteed"/>
            <enumeration value="NotGuaranteed"/>
      </restriction>
</simpleType>
<simpleType name="statusValue.type">
      <restriction base="NMTOKEN">
            <enumeration value="agreed"/>
            <enumeration value="signed"/>
            <enumeration value="proposed"/>
      </restriction>
</simpleType>
<simpleType name="endpointType.type">
      <restriction base="NMTOKEN">
            <enumeration value="login"/>
            <enumeration value="request"/>
            <enumeration value="response"/>
            <enumeration value="error"/>
            <enumeration value="allPurpose"/>
      </restriction>
</simpleType>
<simpleType name="non-empty-string">
      <restriction base="string">
            <minLength value="1"/>
      </restriction>
</simpleType>
```

```
      <simpleType name="syncReplyMode.type">
            <restriction base="NMTOKEN">
                  <enumeration value="responseOnly"/>
                  <enumeration value="signalsAndResponse"/>
                  <enumeration value="signalsOnly"/>
                  <enumeration value="none"/>
            </restriction>
      </simpleType>
      <complexType name="service.type">
            <simpleContent>
                  <extension base="tns:non-empty-string">
                        <attribute name="type" type="tns:non-empty-string"/>
                  </extension>
            </simpleContent>
      </complexType>
      <complexType name="protocol.type">
            <simpleContent>
                  <extension base="tns:non-empty-string">
                        <attribute ref="tns:version"/>
                  </extension>
            </simpleContent>
      </complexType>
      <attribute name="idref" type="IDREF" form="unqualified"/>
      <attribute name="id" type="ID" form="unqualified"/>
      <attribute name="version" type="tns:non-empty-string"/>
      <attribute name="syncReplyMode" type="tns:syncReplyMode.type"/>
</schema>
```

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

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

Formats of character strings

Protocol and version elements

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

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

Alphanumeric strings

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

Values of elements are case insensitive unless otherwise stated.

Strings which represent file or directory names are case sensitive to ensure that they are acceptable to both UNIX and Windows systems.

Numeric Strings

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

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

Overview and limitations

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

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

1. The precise informational inputs to the *CPA* formation procedure vary.

2. There exist at least two distinct approaches to *CPA* formation. One useful approach for certain situations involves basing *CPA* formation from a *CPA* template; the other approach involves composition from *CPPs*.

3. The conditions for output of a given *CPA* given two *CPPs* can involve different levels and extents of interoperability. In other words, when an optimal solution that satisfies every level of requirement and every other additional constraint does not exist, a *Party* MAY propose a *CPA* that satisfies enough of the requirements for "a good enough" implementation. User input MAY be solicited to determine what is a good enough implementation, and so MAY be as varied as there are user configuration options to express preferences. In practice, compromises MAY be made on security, reliable messaging, levels of signals and acknowledgements, and other matters in order to find some acceptable means of doing *Business*.

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

Variability in inputs

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

A *CPA* formation tool MAY have access to local user information not advertised in the *CPP* that MAY contribute to the *CPA* that is formed. A user MAY have chosen to only advertise those system capabilities that reflect nondeprecated capabilities. For example, a user MAY only advertise HTTP and omit FTP, even when capable of using FTP. The reason for omitting FTP might be concerns about the scalability of managing user accounts, directories, and passwords for FTP sessions. Despite not advertising an FTP capability, configuration software MAY use tacit knowledge about its own FTP capability to form a *CPA* with an intended collaborator who happens to have only an FTP capability for implementing a desired *Business Collaboration*. In other words, *Business* interests MAY, in this case, override the deprecation policy. Both tacit knowledge and detailed preference information account for variability in inputs into the *CPA* formation process.

Different approaches

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

Variable output "satisficing" policies

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

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

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

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

CPA formation component tasks

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

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

CPA formation from CPPs: enumeration of tasks

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

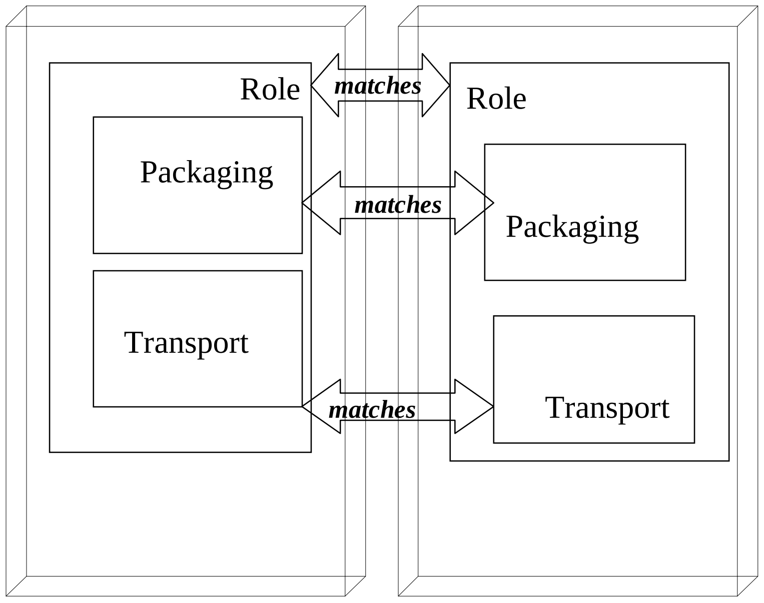
1. an intended collaborator is known and the collaborator's *CPP* has been retrieved,
2. the *Business Collaboration* between us and our intended collaborator has been selected,
3. the specific role that our software agent is to play in the *Business Collaboration* is known, and
4. the capabilities that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

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

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

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

Figure 6: Basic Tasks in Forming a CPA



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

Matching roles

Our agent has its role already selected in the *Business Collaboration*. So it now begins to check the **Role** elements in its collaborator's *CPP*. The first element to examine is the **PartyInfo** element that contains a subtree of elements called **CollaborationRole**. This set is searched to discover a role that complements the role of our agent within the *Business Collaboration* that we have chosen. For simple binary collaboration cases, it is typically sufficient to find that our intended collaborator's **CollaborationRole** set contains **ProcessSpecification** elements that we intend to implement and where the role is not identical to our role. For more general collaborations, we would need to know the list of roles available within the process, and keep track that for each of the collaborators, the roles chosen instantiate those that have been specified within the *Process-Specification* document. Collaborations involving more than two roles are not discussed further.

Matching transport

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

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

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

```
<Transport transportId = "buyerid001">
      <SendingProtocol>HTTP</SendingProtocol>
      <ReceivingProtocol>
      HTTP
      </ReceivingProtocol>
      <Endpoint uri = "https://www.buyername.com/po-response"
                  type = "allPurpose"/>
      <TransportSecurity>
            <Protocol version = "1.0">TLS</Protocol>
            <CertificateRef certId =  certid001">BuyerName</CertificateRef>
      </TransportSecurity>
</Transport>
and our seller has a Transport entry:
<Transport transportId = "sellid001">
      <SendingProtocol>HTTP</SendingProtocol>
      <ReceivingProtocol>
      HTTP
      </ReceivingProtocol>
      <Endpoint uri = "https://www.sellername.com/pos_here"
                  type = "allPurpose"/>
      <TransportSecurity>
            <Protocol version = "1.0">TLS</Protocol>
            <CertificateRef certId ="certid002">Sellername</CertificateRef>
      </TransportSecurity>
</Transport>
```

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

Matching transport security

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

Matching document packaging

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

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

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

1. The buyer asserting what packaging it can generate for its purchase order, and what packaging it can parse for its purchase order response *Messages*.

2. The seller asserting what packaging it can generate for its purchase order responses and what packaging it can parse for received purchase orders.

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

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

```
<Packaging id="I1001">
      <ProcessingCapabilities parse = "true" generate = "true"/>
      <SimplePart id = "P1" mimetype = "text/xml"/>
         <NamespaceSupported location
               = "http://schemas.xmlsoap.org/soap/envelope/" version = "1.1">
              http://schemas.xmlsoap.org/soap/envelope
         </NamespaceSupported>
         <NamespaceSupported location =
              "http://www.ebxml.org/namespaces/messageHeader"

              version = "1.0">

              http://www.ebxml.org/namespaces/messageHeader

         </NamespaceSupported>         <NamespaceSupported location =

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

                version = "1.0">

                http://www.w3.org/2000/09/xmldsig#

         </NamespaceSupported>
      <SimplePart id = "P2" mimetype = "application/xml"/>
      <CompositeList>
            <Composite mimetype = "multipart/related" id = "P3"
                  mimeparameters = "type=text/xml">
                  <Constituent idref = "P1"/>
                  <Constituent idref = "P2"/>
            </Composite>
      </CompositeList>
</Packaging>
<Packaging id="I2001">
      <ProcessingCapabilities parse = "true" generate = "true"/>
      <SimplePart id = "P11" mimetype = "text/xml"/>
      <SimplePart id = "P12" mimetype = "application/xml"/>
      <CompositeList>
            <Composite mimetype = "multipart/related" id = "P13"
                  mimeparameters = "type=text/xml">
                  <Constituent idref = "P11"/>
                  <Constituent idref = "P12"/>
            </Composite>
      </CompositeList>
</Packaging>
```

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

Matching document-level security

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

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

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

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

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

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

Other considerations

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