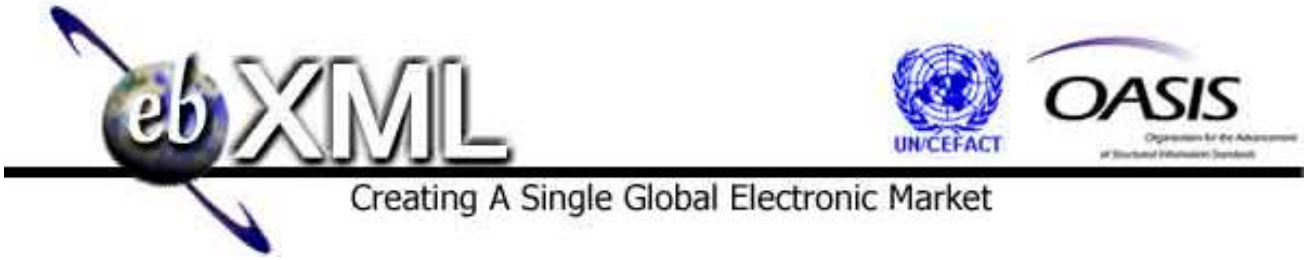


Collaboration-Protocol Profile and Agreement Specification Version 1.~~05~~06

OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee

~~24January~~ 4 February 2002

1 Status of this Document

This document specifies an ebXML SPECIFICATION for the eBusiness community.

Distribution of this document is unlimited.

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

This version:

http://www.oasis-open.org/committees/ebxml-cppa/documents/working_drafts/ebCPP-1_~~05~~06.pdf

Previous version:

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

2 Technical Committee Members

Selem Aissi (Voting Membership Pending as of San Francisco F2F)
Arvola Chan
James Bryce Clark
David Fischer (Voting Membership Pending as of San Francisco F2F)
Tony Fletcher
Brian Hayes
Neelakantan Kartha
Kevin Liu
Pallavi Malu
Dale Moberg
Himagiri Mukkamala
Peter Ogden
Marty Sachs
Yukinori Saito
David Smiley
Tony Weida
Pete Wenzel
Jean Zheng
Sinisa Zimek

3 ebXML Participants

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

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~~4 Table of Contents

45 Introduction

4.15.1 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*. A *CPA* MAY be created by computing the intersection of the two *Partners' CPPs.* 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 65 defines the objectives of this specification.
- Section 76 provides a system overview.
- Section 87 contains the definition of the *CPP*, identifying the structure and all necessary fields.
- Section 98 contains the definition of the *CPA*.
- The appendices include examples of an XML *Business Process Specification* (non-normative), *CPP* and *CPA* documents (non-normative), an example XML *Business Process Specification* (non-normative), an XML Schema document (normative), formats of information in the *CPP* and *CPA* (normative), and composing a *CPA* from two *CPPs* (non-normative) and a Glossary of Terms.

4.25.2 Document Conventions

Terms in *Italics* are defined in Appendix H Appendix H Appendix H (Glossary of Terms).

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

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

References to external documents are represented with BLOCK text enclosed in brackets, e.g. [RFC2396]. The references are listed in Section 10~~9~~, "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]. ~~They~~ In this document, they appear in all capital letters to emphasize their usage as keywords, as do the following [XML] keywords ID, IDREF, IMPLIED, FIXED.

> 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 elective 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.

~~4.3~~5.3 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.

~~4.4~~5.4 Definitions

Technical terms in this specification are defined in Appendix H ~~Appendix H~~ H.

~~4.5~~5.5 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*.

~~4.6~~5.6 Assumptions

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

4.75.7 Related Documents

Related documents include ebXML Specifications on the following topics:

- ebXML Technical Architecture Specification[ebTA]
- ebXML *Message* Service Specification[ebMS]
- ebXML Business Process Specification Schema[ebBPSS]
- ebXML Core Component and Business Document Overview[ccOVER]
- ebXML Registry Services Specification[ebRS]

See Section 109 for the complete list of references.

56 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.

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 F ~~Appendix F~~ ~~Appendix F~~ contains 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[X12] to more automated means of setting up *Business* relationships and doing *Business* under them.

6 7 System Overview

6.17.1 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 8.5.47.5.4.

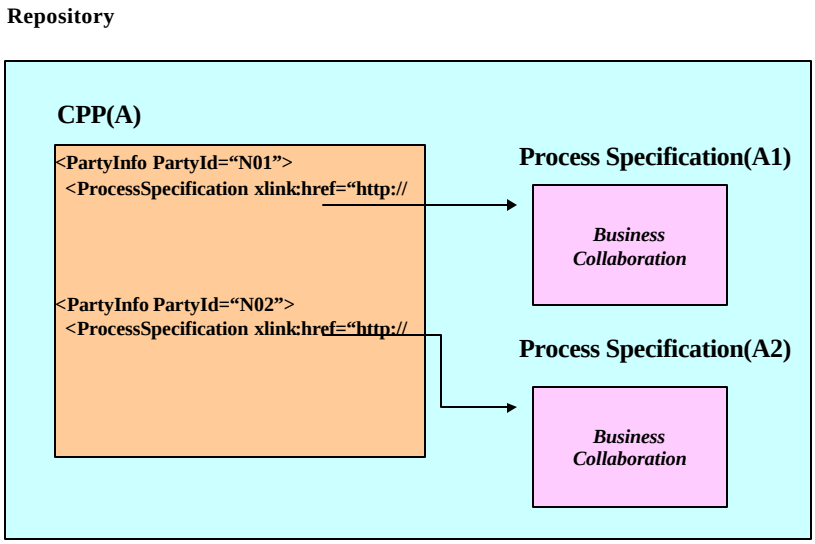
Figure 1 illustrates the relationships between a *CPP* and two *Process-Specification* documents, A1 and A2, in an ebXML Registry. On the left is a *CPP*, A, which includes information about two parts of an enterprise that are represented as different *Parties.* On the right are shown two *Process-Specification* 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 two sample *CPPs*, a sample *CPA* formed from the *CPPs*, a sample *Process-Specification* referenced by the *CPPs* and the *CPA*, and the XML Schema governing the structures of *CPPs* and *CPAs*.

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



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.

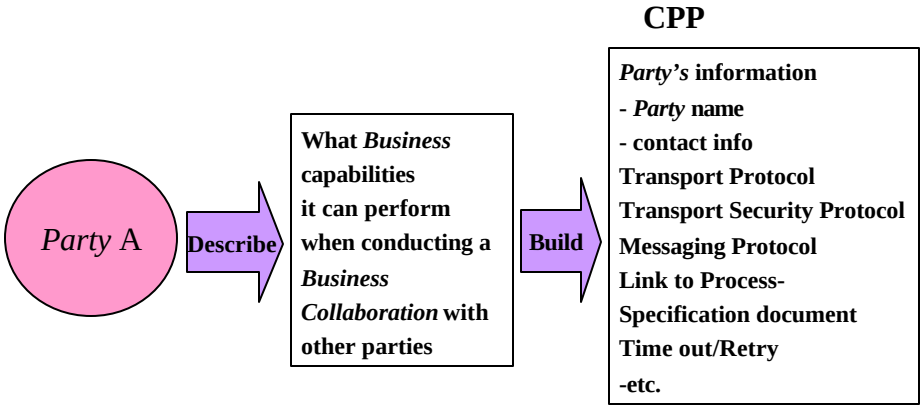
In general, the *Parties* to a *CPA* can have both client and server characteristics. A client requests services and a server provides services to the *Party* requesting services. In some applications, one *Party* only requests services and one *Party* only provides services. These applications have 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.

~~6.2~~7.2 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 ~~Appendix F~~ F "Composing a CPA from Two CPPs (Non-Normative)" for more information.

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

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



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

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

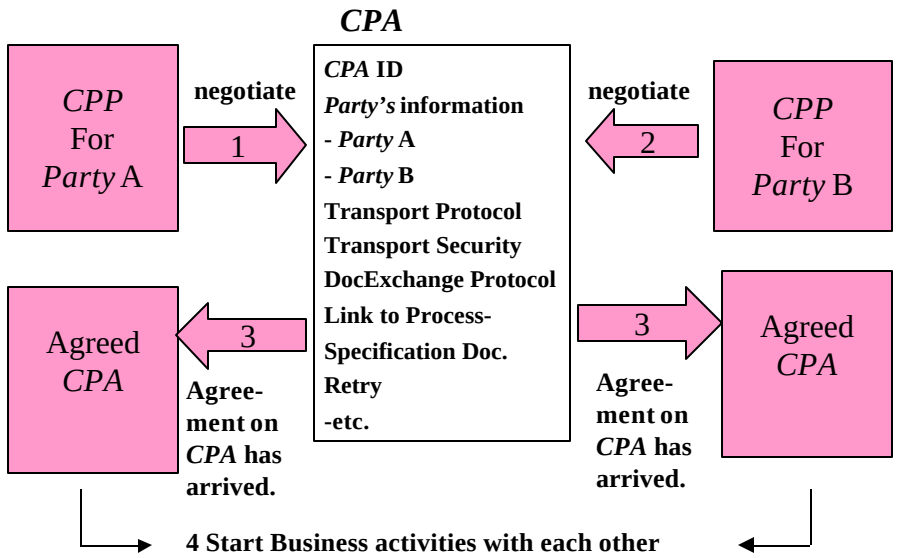


Figure 4 illustrates the entire process. The steps are listed at the left. The end of the process is that the two *Parties* configure their systems from identical copies of the agreed *CPA* and they are

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

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

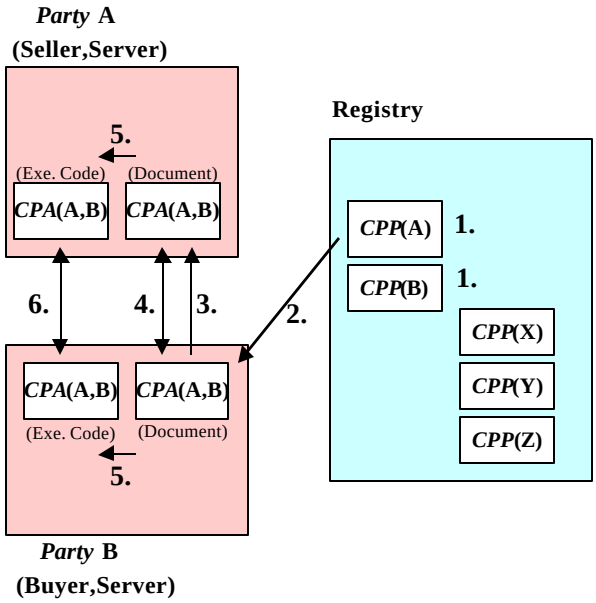
2. *Party* B discovers trading partner A (Seller) by searching in the Registry and downloads *CPP*(A) to *Party* B's server.

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*.



then ready to do *Business.*

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 8.1~~7.1~~ for more information.

~~6.3~~7.3 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.

~~6.~~4~~7~~.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 graphical *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.

~~6.~~5~~7~~.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 CPA composition from two CPPs and run-time support where these notes serve to clarify the *CPP* and *CPA* definitions. See Section 11~~10~~ 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 XML Schema document 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*.

78 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 D Appendix D D for the XML Schema, and Appendix A Appendix A A for sample *CPP* documents.

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 and *Message*-sending 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 non-repudiation if specified, and passes it to the transport layer for transmission to the other *Party*. It performs the inverse steps for received *Messages*. 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.

Document-Exchange Layer - The Document-exchange layer specifies processing of the business documents by the Message-exchange function. Properties specified include encryption, digital signature, and reliable-messaging characteristics. 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 Message encryption MUST be specified in the Document-exchange layer. The

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

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

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

~~7.1~~8.1 Globally-Unique Identifier of CPP Instance Document

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

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

~~7.2~~8.2 SchemaLocation Attribute

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.

An example of the use of the schemaLocation attribute follows:

```
      <tp:CollaborationProtocolAgreement
          xmlns:tp="http://www.oasis-open.org/committees/ebxml-
cppa/schema/cpp-cpa-1_1.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-
cppa/schema/cpp-cpa-1_1.xsd http://www.oasis-open.org/committees/ebxml-
cppa/schema/cpp-cpa-1_1.xsd">
          ...
      </tp:CollaborationProtocolAgreement>
```

~~7.3~~8.3 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.

```
      <tp:CollaborationProtocolProfile
          xmlns:tp="http://www.oasis-open.org/committees/ebxml-
cppa/schema/cpp-cpa-1_1.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-
cppa/schema/cpp-cpa-1_1.xsd http://www.oasis-open.org/committees/ebxml-
cppa/schema/cpp-cpa-1_1.xsd"
          xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
          xmlns:xlink="http://www.w3.org/1999/xlink"
          tp:version="1.1">
          <tp:PartyInfo>  <!-- one or more -->
          ...
          </tp:PartyInfo>
          <tp:SimplePart>  <!-- one or more -->
          ...
          </tp:SimplePart>
          <tp:Packaging id="ID">  <!-- one or more -->
          ...
          </tp:Packaging>
          <ds:Signature>  <!--zero or one-->
          ...
          </ds:Signature>
          <tp:Comment>text</tp:Comment>  <!--zero or more-->
      </tp:CollaborationProtocolProfile>
```

~~7.~~4.8.4 CollaborationProtocolProfile element

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

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

- The *CPP/CPA* namespace: `xmlns:tp="http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-1_1.xsd"`,
- 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 or more REQUIRED ***SimplePart*** elements that describe the constituents used to make up composite *Messages.*
- One or more REQUIRED ***Packaging*** elements that describe how the *Message Header* and payload constituents are packaged for transmittal,
- Zero or one ***ds:Signature*** element that contains 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].

7.58.5 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:

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

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

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

The following example illustrates two possible party names.

```
<tp:PartyInfo tp:partyName="Example, Inc."...</tp:PartyInfo>

<tp:PartyInfo tp:partyName="Example, Inc. US Western Division">
...
</tp:PartyInfo>
```

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

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~~ protocol (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 send and 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.
- Zero or more ***OverrideMshActionBinding*** elements that specify the DeliveryChannel to use for asynchronously delivered *Message Service Handler* level messages.

~~7.5.1~~8.5.1 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 (Dun and Bradstreet) 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 in a well-known directory of organization identifiers.

The following example illustrates two URI references.

```
<tp:PartyId tp:type="anyURI">urn:duns:123456789</tp:PartyId>

<tp:PartyId tp:type="anyURI"> urn:icann:example.com</tp: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.

7.5.28.5.2 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 (www.uddi.org), or an a Lightweight Directory Access Protocol[RFC2251] (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.

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

7.5.2.18.5.2.1 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.

7.5.2.28.5.2.2 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.*

7.5.2.38.5.2.3 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:

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

7.5.38.5.3 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, based on RosettaNet™ PIP 3A4 is:~~element is~~:

```
<tp:CollaborationRole tp:id="BuyerId">
    <tp:ProcessSpecification
        tp:version="2.0"
        tp:name="PIP3A4RequestPurchaseOrder"
        xlink:type="simple"
        xlink:href="http://www.rosettanet.org/processes/3A4.xml"/>
    <tp:Role
        tp:name="Buyer"
        xlink:type="simple"

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

```
            <tp:ApplicationCertificateRef
                tp:certId="company1SigningCertificate"/>
            <!-- This service binding uses an asynchronous delivery channel to
                 receive the signals and response -->
            <tp:ServiceBinding>
                <tp:Service
tp:type="anyURI">bpid:RosettaNet:PIP3A4RequestPurchaseOrder$2.0</tp:Service>
                <tp:ActionBinding
                    tp:action="Purchase Order Confirmation Action"
                    tp:channelId="channel1"
                    tp:packageId="ResponsePackage">
                    <tp:ActionContext
                        tp:binaryCollaboration="Request Purchase Order"
                        tp:businessTransactionActivity="Request Purchase Order"
                        tp:requestOrResponseAction="Purchase Order Confirmation
Action"/>
                </tp:ActionBinding>
                <!-- Receipt Acknowledgment and Exception signals are delivered
                     using the designated delivery channels -->
                <tp:ActionBinding
                    tp:action="ReceiptAcknowledgment"
                    tp:channelId="channel1"
                    tp:packageId="ReceiptAcknowledgmentPackage"/>
                <tp:ActionBinding
                    tp:action="Exception"
                    tp:channelId="channel1"
                    tp:packageId="ExceptionPackage"/>
            </tp:ServiceBinding>
</tp: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 ***ApplicationCertificateRef*** element, zero or one ***ApplicationSecurityDetailsRef*** 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 ***ApplicationCertificateRef*** element identifies the certificate to be used for application level signature and encryption. The ***ApplicationSecurityDetailsRef*** element identifies the trust anchors and security policy that will be applied to any application-level certificate offered by the other *Party*. Each ***ServiceBinding*** element provides a binding of the role to a default ***DeliveryChannel*** (through the ***defaultSignalChannelId*** attribute) for sending business signal messages like *Receipt Acknowledgment* and *Exception*. The ***ActionBinding*** elements identify the ***DeliveryChannel*** elements that are relevant for delivering business action messages received by the ***Role*** in question. They MAY also be used for specifying ***DeliveryChannels*** for business signal messages.

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.

> 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 the delivery of standalone *Message* Service Handler level signals like (Reliable Messaging) Acknowledgments, Errors, StatusRequest, StatusResponse, etc.

> 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.

~~7.5.3.1~~8.5.3.1 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.

~~7.5.4~~8.5.4 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 accordance with the ebXML Business Process Specification Schema[ebBPSS]. The *Process-Specification* document MAY be kept in an ebXML Registry.

> NOTE: A *Party* ~~MAY~~ can 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 ***ActionBinding*** element, the ***ActionContext*** element, and some attributes of the ***BusinessProcessCharacteristics*** element.

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

```
        <tp:ProcessSpecification
            tp:version="2.0"
            tp:name="PIP3A4RequestPurchaseOrder"
            xlink:type="simple"
            xlink:href="http://www.rosettanet.org/processes/3A4.xml"
            <ds:Reference ds:URI="http://www.rosettanet.org/processes/3A4.xml">
                <ds:Transforms>
                    <ds:Transform
ds:Algorithm="http://www.w3.org/TR/20002001/REC-xml-c14n-20010315"/>
                </ds:Transforms>
                <ds:DigestMethod
                    ds:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
            </ds:Reference>
        </tp:ProcessSpecification>
```

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

- a REQUIRED ***name*** attribute,
- 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 CPP (CPA) document is signed, ~~it~~ each ***ProcessSpecification*** element MUST ~~use~~ contain 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. The ***ds:Reference*** element specifies a digest method and digest value to enable verification that the referenced *Process-Specification* document has not changed.

~~7.5.4.1~~8.5.4.1 name attribute

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

~~7.5.4.2~~8.5.4.2 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).

~~7.5.4.3~~8.5.4.3 xlink:type attribute

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

7.5.4.48.5.4.4 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].

7.5.4.58.5.4.5 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, through the use of a digest method and digest value as described below.

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.

1. The syntax and semantics of the ***ds:Reference*** element and its child elements are defined in the XML Digital Signature specification[XMLDSIG]. In addition, To identify the *Process-Specification* document, the ***ds:Reference*** MUST include a ***ds:URI*** attribute whose value is identical to that of the ***xlink:href*** attribute in the enclosing ***ProcessSpecification*** element.
2. 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/2001/Rec-xml-c14n-20010315`

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

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

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*** element 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].

7.5.58.5.5 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.

7.5.5.18.5.5.1 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 ***BusinessPartnerRole*** element.

See NOTE in Section 8.5.47.5.4 regarding alternative *Business-Collaboration* descriptions.

7.5.5.28.5.5.2 xlink:type attribute

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

7.5.5.38.5.5.3 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.rosettanet.org/processes/3A4.xml#Buyer"
```

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

7.5.68.5.6 ApplicationCertificateRef element

The OPTIONAL ***ApplicationCertificateRef*** element identifies a signing certificate for use by the

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

The ***ApplicationCertificateRef*** element has

- A REQUIRED ***certId*** attribute

7.5.6.18.5.6.1 certId attribute

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

7.5.78.5.7 ApplicationSecurityDetailsRef element

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

The ***ApplicationSecurityDetailsRef*** element has

- A REQUIRED ***securityId*** attribute

7.5.7.18.5.7.1 SecurityId attribute

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

7.5.88.5.8 ServiceBinding element

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

```
<tp:ServiceBinding tp:defaultSignalChannelId="channelA1">
   <tp:Service tp:type="anyURI">
   bpid:RosettaNet:PIP3A4RequestPurchaseOrder$2.0
   </tp:Service>
   <tp:WillInitiate>
      <tp:ThisPartyActionBinding tp:action="Purchase Order Request Action"
                                 tp:channelId="channelA1"
                                 tp:packageId="RequestPackage">
         <tp:ActionContext tp:binaryCollaboration="Request Purchase Order"
                           tp:businessTransactionActivity="Request Purchase Order"
                           tp:requestOrResponseAction="Purchase Order Request Action"/>
      </tp:ThisPartyActionBinding>
      <tp:OtherPartyActionBinding tp:action="Purchase Order Request Action"
                                  tp:channelId="channelB1"
                                  tp:packageId="RequestPackage">
         <tp:ActionContext tp:binaryCollaboration="Request Purchase Order"
                           tp:businessTransactionActivity="Request Purchase Order"
                           tp:requestOrResponseAction="Purchase Order Request Action"/>
      </tp:OtherPartyActionBinding>
   </tp:WillInitiate>
```

```
   <tp:WillInitiate>
      <tp:ThisPartyActionBinding tp:action="ReceiptAcknowledgement"
                                 tp:channelId="channelA1"
                                 tp:packageId="ReceiptAcknowledgmentPackage"/>
      <tp:OtherPartyActionBinding tp:action="ReceiptAcknowledgement"
                                  tp:channelId="channelB1"
                                  tp:packageId="ReceiptAcknowledgmentPackage"/>
   </tp:WillInitiate>
   <tp:WillRespond>
      <tp:ThisPartyActionBinding tp:action="Purchase Order Confirmation Action"
                                 tp:channelId="channelA1"
                                 tp:packageId="ResponsePackage">
         <tp:ActionContext tp:binaryCollaboration="Request Purchase Order"
                           tp:businessTransactionActivity="Request Purchase Order"
                           tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
      </tp:ThisPartyActionBinding>
      <tp:OtherPartyActionBinding tp:action="Purchase Order Confirmation Action"
                                  tp:channelId="channelB1"
                                  tp:packageId="ResponsePackage">
         <tp:ActionContext tp:binaryCollaboration="Request Purchase Order"
                           tp:businessTransactionActivity="Request Purchase Order"
                           tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
      </tp:OtherPartyActionBinding>
   </tp:WillRespond>
   <!-- Receipt Acknowledgment and Exception signals are delivered
        using the designated delivery channels -->
   <tp:WillRespond>
      <tp:ThisPartyActionBinding tp:action="ReceiptAcknowledgment"
                                 tp:channelId="channelA1"
                                 tp:packageId="ReceiptAcknowledgmentPackage"/>
      <tp:OtherPartyActionBinding tp:action="ReceiptAcknowledgment"
                                  tp:channelId="channelB1"
                                  tp:packageId="ReceiptAcknowledgmentPackage"/>
   </tp:WillRespond>
   <tp:WillRespond>
      <tp:ThisPartyActionBinding tp:action="Exception"
                                 tp:channelId="channelA1"
                                 tp:packageId="ExceptionPackage"/>
      <tp:OtherPartyActionBinding tp:action="Exception"
                                  tp:channelId="channelB1"
                                  tp:packageId="ExceptionPackage"/>
   </tp:WillRespond>
</tp:ServiceBinding>
      <tp:ServiceBinding>
         <tp:Service
tp:type="anyURI">bpid:RosettaNet:PIP3A4RequestPurchaseOrder$2.0</tp:Service>
         <tp:WillInitiate>
            <tp:ThisPartyActionBinding tp:action="Purchase Order
Request Action" tp:channelId="channelA1" tp:packageId="RequestPackage">
               <tp:ActionContext tp:binaryCollaboration="Request
Purchase Order" tp:businessTransactionActivity="Request Purchase Order"
tp:requestOrResponseAction="Purchase Order Request Action"/>
            </tp:ThisPartyActionBinding>
            <tp:OtherPartyActionBinding tp:action="Purchase Order
Request Action" tp:channelId="channelB1" tp:packageId="RequestPackage">
               <tp:ActionContext tp:binaryCollaboration="Request
Purchase Order" tp:businessTransactionActivity="Request Purchase Order"
tp:requestOrResponseAction="Purchase Order Request Action"/>
            </tp:OtherPartyActionBinding>
         </tp:WillInitiate>
         <tp:WillInitiate>
            <tp:ThisPartyActionBinding
tp:action="ReceiptAcknowledgement" tp:channelId="channelA1"
tp:packageId="ReceiptAcknowledgmentPackage"/>
```

```
            <tp:OtherPartyActionBinding
tp:action="ReceiptAcknowledgement" tp:channelId="channelB1"
tp:packageId="ReceiptAcknowledgmentPackage"/>
        </tp:WillInitiate>
        <tp:WillRespond>
            <tp:ThisPartyActionBinding tp:action="Purchase Order
Confirmation Action" tp:channelId="channelA1" tp:packageId="ResponsePackage">
                <tp:ActionContext tp:binaryCollaboration="Request
Purchase Order" tp:businessTransactionActivity="Request Purchase Order"
tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
            </tp:ThisPartyActionBinding>
            <tp:OtherPartyActionBinding tp:action="Purchase Order
Confirmation Action" tp:channelId="channelB1" tp:packageId="ResponsePackage">
                <tp:ActionContext tp:binaryCollaboration="Request
Purchase Order" tp:businessTransactionActivity="Request Purchase Order"
tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
            </tp:OtherPartyActionBinding>
        </tp:WillRespond>
        <!-- Receipt Acknowledgment and Exception signals are delivered
using the designated delivery channels -->
        <tp:WillRespond>
            <tp:ThisPartyActionBinding
tp:action="ReceiptAcknowledgment" tp:channelId="channelA1"
tp:packageId="ReceiptAcknowledgmentPackage"/>
            <tp:OtherPartyActionBinding
tp:action="ReceiptAcknowledgment" tp:channelId="channelB1"
tp:packageId="ReceiptAcknowledgmentPackage"/>
        </tp:WillRespond>
        <tp:WillRespond>
            <tp:ThisPartyActionBinding tp:action="Exception"
tp:channelId="channelA1" tp:packageId="ExceptionPackage"/>
            <tp:OtherPartyActionBinding tp:action="Exception"
tp:channelId="channelB1" tp:packageId="ExceptionPackage"/>
        </tp:WillRespond>
    </tp:ServiceBinding>
```

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

The ***ServiceBinding*** element also has:

- A REQUIRED ***defaultSignalChannelId*** attribute.

7.5.8.18.5.8.1 defaultSignalChannelId attribute

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

7.5.98.5.9 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 the uuid (URI) specified for the ProcessSpecification Element in the Business Process Specification Schema instance document.

> NOTE: The purpose of the ***Service*** element is 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*. The **Service** element MAY be used along with the ***WillInitiate*** and ***WillRespond*** elements (and their descendants) to provide routing of received messages to the correct application entry point.~~to provide application routing of received messages.~~

~~7.5.9.1~~8.5.9.1 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]. ~~If using the ebXML Business Process Specification[ebBPSS] for defining~~ If using the ebXML Business Process Specification[ebBPSS] for defining the *Process-Specification* document, the type attribute MUST be a URI[RFC2396].

~~7.5.10~~8.5.10 WillInitiate element

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

~~7.5.11~~8.5.11 WillRespond element

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

~~7.5.12~~8.5.12 ThisPartyActionBinding element and OtherPartyActionBinding element

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

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

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

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

~~7.5.12.1~~8.5.12.1 action attribute

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

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

7.5.12.28.5.12.2 channelId attribute

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

7.5.12.38.5.12.3 packageId attribute

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

7.5.12.48.5.12.4 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 ***RequestingBusinessActivity*** or ***RespondingBusinessActivity*** element within the associated *Process-Specification* document[ebBPSS] that is identified by the ***ProcessSpecification*** element.

7.5.12.58.5.12.5 xlink:type attribute

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

7.5.138.5.13 ActionContext element

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

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

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

- an OPTIONAL ***CollaborationActivity*** element
- an OPTIONAL #***wildcard*** element

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

- a REQUIRED ***binaryCollaboration*** attribute,
- a REQUIRED ***businessTransactionActivity*** attribute,
- a REQUIRED ***requestOrResponseAction*** attribute.

7.5.13.18.5.13.1 binaryCollaboration attribute

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

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

7.5.13.28.5.13.2 businessTransactionActivity attribute

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

7.5.13.38.5.13.3 requestOrResponseAction attribute

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

7.5.148.5.14 CollaborationActivity element

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

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

```
<tp:CollaborationActivity
            tp:name="Credit Check">
      <tp:CollaborationActivity
            tp:name="Credit History Check">
      </tp:CollaborationActivity>
</tp:CollaborationActivity>
```

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

The ***CollaborationActivity*** element also has one attribute:

- a REQUIRED ***name*** attribute

7.5.14.18.5.14.1 name attribute

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

7.5.158.5.15 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:

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

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

7.5.15.18.5.15.1 certId attribute

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

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

7.5.15.28.5.15.2 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~~ MUST recognize the ***ds:KeyInfo*** element and insert the subelement structure necessary to define the certificate.

7.5.168.5.16 SecurityDetails element

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

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

```
                <tp:AnchorCertificateRef tp:certId="TrustedRootCertA3"/>
                <tp:AnchorCertificateRef tp:certId="TrustedRootCertA5"/>
            </tp:TrustAnchors>
            <tp:SecurityPolicy> ... </tp:SecurityPolicy>
        </tp:SecurityDetails>
```

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

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

The ***SecurityDetails*** element has one attribute:

- A REQUIRED ***securityId*** attribute

~~7.5.16.1~~8.5.16.1 securityId attribute

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

```
        <tp:SigningSecurityDetailsRef
tp:securityId="CompanyA_MessageSecurity"/>
```

~~7.5.17~~8.5.17 TrustAnchors element

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

~~7.5.18~~8.5.18 SecurityPolicy element

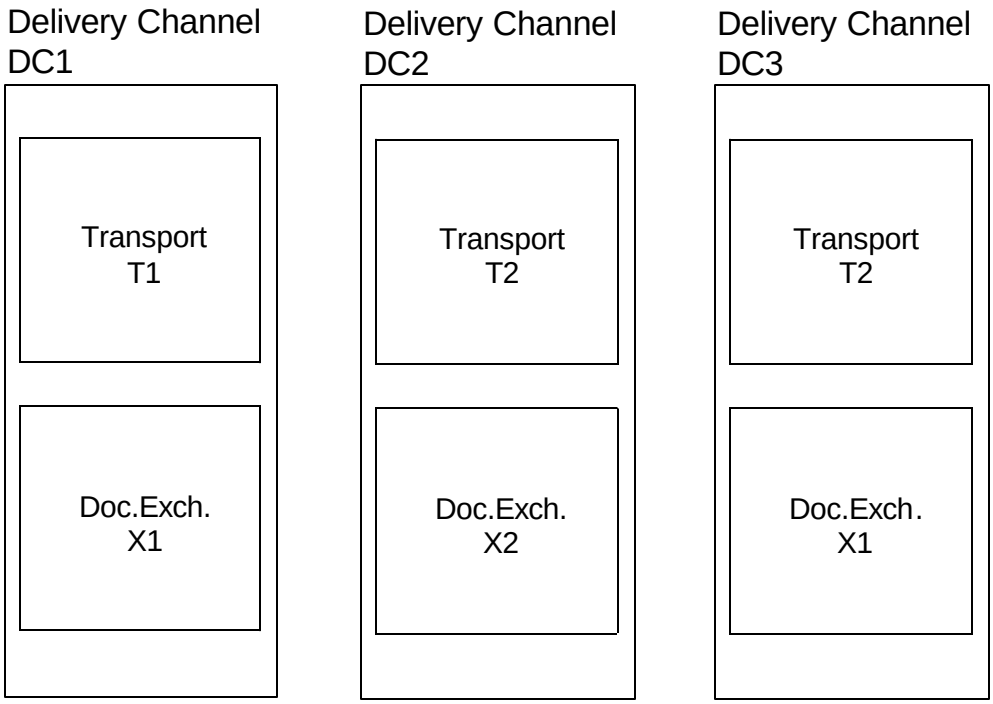
The ~~OPTIONAL~~ ***SecurityPolicy*** element is a placeholder future apparatus that will enable the *Party* to specify its policy and compliance regarding specific components of its public key infrastructure. For example, it might stipulate revocation checking procedures or constraints related to name,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

~~7.5.19~~8.5.19 DeliveryChannel element

A delivery channel is a combination of a ***Transport*** element and a ***DocExchange*** element that describes the *Party's Message* communication 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~~ SHALL refer to any combination of a

DocExchange element and a ***Transport*** element. The same ***DocExchange*** element or the same ***Transport*** element ~~MAY~~ can be be referred to by more than one delivery channel. Two delivery channels ~~MAY~~ can 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~~ can 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 ***PartyInfo*** element, ***OverrideMshActionBinding*** element, ***ServiceBinding*** element, ***ThisPartyActionBinding,*** or ***OtherPartyActionBinding*** element (***action*** attribute). Following is the delivery-channel syntax.

```
<tp:DeliveryChannel
    tp:channelId="channel1"
    tp:transportId="transport1"
    tp:docExchangeId="docExchange1"
    <tp:BusinessProcessCharacteristics
        tp:isNonRepudiationRequired="true"
```

```
            tp:isNonRepudiationReceiptRequired="false"
            tp:osSecureTransportRequired="true"
            tp:isConfidential="true"
            tp:isAuthenticated="true"
            tp:isAuthorizationRequired="false"/>
        <tp:MessagingCharacteristics
            tp:syncReplyMode="none"
            tp:ackRequested="always"
            tp:ackSignatureRequested="always"
            tp:duplicateElimination="always"
            tp:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH"/>
    </tp: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 two REQUIRED child elements, ***BusinessProcessCharacteristics*** and ***MessagingCharacteristics.***

~~7.5.19.1~~8.5.19.1 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.*

~~7.5.19.2~~8.5.19.2 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.

~~7.5.19.3~~8.5.19.3 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.

~~7.5.20~~8.5.20 BusinessProcessCharacteristics element

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

See NOTE in Section 8.5.4~~7.5.4~~ regarding alternative *Business-Collaboration* descriptions.

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

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

- an IMPLIED ***isNonRepudiationRequired*** attribute,
- an IMPLIED ***isNonRepudiationReceiptRequired*** attribute,
- an IMPLIED ***isSecureTransportRequired*** attribute,
- an IMPLIED ***isConfidential*** attribute,
- an IMPLIED ***isAuthenticated*** attribute,
- an IMPLIED ***isAuthorizationRequired*** attribute.

~~7.5.20.1~~8.5.20.1 isNonRepudiationRequired attribute

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

~~7.5.20.2~~8.5.20.2 isNonRepudiationReceiptRequired attribute

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

~~7.5.20.3~~8.5.20.3 isSecureTransportRequired attribute

The ***isSecureTransportRequired*** 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].

~~7.5.20.4~~8.5.20.4 isConfidential attribute

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

~~7.5.20.5~~8.5.20.5 isAuthenticated attribute

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

~~7.5.20.6~~8.5.20.6 isAuthorizationRequired attribute

The ***isAuthorizationRequired*** 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~~ MUST

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

~~7.5.2~~8.5.21 MessagingCharacteristics element

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

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

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

- An IMPLIED ***syncReplyMode*** attribute,
- an IMPLIED ***ackRequested*** attribute,
- an IMPLIED ***ackSignatureRequested*** attribute,
- an IMPLIED ***duplicateElimination*** attribute,
- an IMPLIED ***actor*** attribute.

~~7.5.21.1~~8.5.21.1 syncReplyMode attribute

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

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

This attribute, when present, indicates what the sending application expects in a synchronous response when bound to a synchronous communication protocol such as HTTP. The value of "mshSignalsOnly" indicates that the response returned (on the HTTP 200 response in the case of HTTP) will only contain standalone *Message Service Handler (MSH)* level messages like Acknowledgment (for Reliable Messaging) and Error messages. All other application level responses are to be returned asynchronously (using a ***DeliveryChannel*** determined by the Service and Action in question). 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], plus any piggybacked MSH level signals, but not a *Business*-response *Message*. If the *Process-Specification* calls for the use of a *Business*-response Message, then the latter MUST be returned asynchronously. The value of "responseOnly" indicates that any *Business* signals indicated in the *Process Specification* are to be omitted and only the *Business*-response *Message* will be returned synchronously, plus any piggybacked MSH level signals. The value of "signalsAndResponse" indicates that the application will synchronously return the *Business*-response *Message* in addition to one or more *Business* signals, plus any piggybacked *MSH* level 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* signal(s) 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*** element is included in the SOAP Header 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), the ***BusinessProcessCharacteristics*** element SHALL NOT have a ***syncReplyMode*** attribute with a value other than "none".

> When the value of the **syncReplyMode** attribute is other than "none", a synchronous delivery channel SHALL be ~~NOTE: It is assumed that a synchronous ***DeliveryChannel*** is~~ used to exchange all messages necessary for conducting a business transaction. If the *Process Specification* calls for the use of ~~-~~non-repudiation of receipt for the response message, then the initiator is expected to return a signed *Receipt Acknowledgment* signal for the responder's response message. ~~However, this is incompatible with the ***syncReplyMode*** values "signalsAndResponse" and "responseOnly", which make no provision for the return for such a signal.~~
>
> NOTE: For HTTP 1.1 clients and servers, two HTTP requests and replies will be sent and received. In other cases, non-repudiation of receipt for the response message could be incompatible with the ***syncReplyMode*** values "signalsAndResponse" and "responseOnly".

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

~~7.5.21.2~~8.5.21.2 ackRequested attribute

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

- "always"
- "never"
- "perMessage"

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

~~7.5.21.3~~8.5.21.3 ackSignatureRequested attribute

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

values:

- "always"
- "never"
- "perMessage"

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

~~7.5.21.4~~8.5.21.4 duplicateElimination attribute

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

- "always"
- "never"
- "perMessage"

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

~~7.5.21.5~~8.5.21.5 actor attribute

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

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

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

~~7.5.22~~8.5.22 Transport element

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

```
<tp:Transport tp:transportId="transportA1">
    <tp:TransportSender> <!-- 0 or 1 times -->
        <tp:TransportProtocol tp:version="1.1">HTTP</tp:Protocol>
        <tp:TransportClientSecurity>
            <tp:TransportSecurityProtocol tp:version="3.0">
                SSL
            </tp:TransportSecurityProtocol>
            <tp:ClientCertificateRef tp:certId="CompanyA_ClientCert"/>
            <tp:ServerSecurityDetailsRef
                tp:securityId="CompanyA_TransportSecurity"/>
        </tp:TransportClientSecurity>
    </tp:TransportSender>
    <tp:TransportReceiver> <!-- 0 or 1 times -->
        <tp:TransportProtocol tp:version="1.1">HTTP</tp:Protocol>
        <tp:Endpoint
            tp:uri=https://www.CompanyA.com/servlets/ebxmlhandler
            tp:type="allPurpose"/>
        <tp:TransportServerSecurity>
            <tp:TransportSecurityProtocol tp:version="3.0">
                SSL
            </tp:TransportSecurityProtocol>
            <tp:ServerCertificateRef tp:certId="CompanyA_ServerCert"/>
            <tp:ClientSecurityDetailsRef
               tp:securityId="CompanyA_TransportSecurity"/>
        </tp:TransportServerSecurity>
    </tp:TransportReceiver>
</tp:Transport>
```

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

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

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

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

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

A *CPP* contains as many ***Transport*** elements as are needed to fully express the *Party*'s inbound

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

The ***Transport*** element has

- a REQUIRED ***transportId*** attribute

7.5.22.18.5.22.1 transportId attribute

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

```
<tp:DeliveryChannel tp:channelId="channelA1"
    tp:transportId="transportA1"
    tp:docExchangeId="docExchangeA1">
    <tp:BusinessProcessCharacteristics . . . />
    <tp:MessagingCharacteristics . . . />
</tp:DeliveryChannel>
```

7.5.238.5.23 TransportSender element

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

The ***TransportSender*** element has no attributes.

7.5.248.5.24 TransportProtocol element

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

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

7.5.258.5.25 TransportClientSecurity element

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

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

7.5.268.5.26 TransportSecurityProtocol element

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

7.5.26.18.5.26.1 Specifics for HTTP

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

7.5.278.5.27 ClientCertificateRef element

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

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

The ***ClientCertificateRef*** element has

- A REQUIRED ***certId*** attribute

7.5.288.5.28 ServerSecurityDetailsRef element

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

The ***ServerSecurityDetailsRef*** element has

- A REQUIRED ***securityId*** attribute

7.5.298.5.29 TransportReceiver element

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

security.

The ***TransportReceiver*** element has no attributes.

7.5.308.5.30 Endpoint element

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

Each ***Endpoint*** has the following attributes:

- a REQUIRED ***uri*** attribute
- an OPTIONAL IMPLIED ***type*** attribute

7.5.30.18.5.30.1 uri attribute

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

7.5.30.28.5.30.2 type attribute

The OPTIONAL ***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 the ***type*** attribute is omitted, its value defaults to "allPurpose". The "login" endpoint MAY beis 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 beis 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".

7.5.318.5.31 TransportServerSecurity element

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

> NOTE: See the note in Section 8.5.257.5.25 regarding the relevance of the ***TransportServerSecurity*** element when synchronous replies are in use.

7.5.328.5.32 ServerCertificateRef element

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

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

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

The ***ServerCertificateRef*** element has

- A REQUIRED ***certId*** attribute

~~7.5.33~~8.5.33 ClientSecurityDetailsRef element

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

The ***ClientSecurityDetailsRef*** element has

- A REQUIRED ***securityId*** attribute

~~7.5.34~~8.5.34 Transport protocols

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

~~7.5.34.1~~8.5.34.1 HTTP

HTTP is Hypertext Transfer Protocol[HTTP]. For HTTP, the endpoint 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:

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

The "request" and "response" endpoints ~~MAY~~ can 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.

~~7.5.34.2~~8.5.34.2 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:

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

SMTP with MIME automatically encodes or decodes the document as needed, 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.

~~7.5.3~~4.3~~8~~.5.3~~4~~.3 FTP

FTP is File Transfer Protocol[RFC959].

~~Since a delivery channel specifies receive characteristics, e~~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:

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

Since FTP needs to 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 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 might 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 might 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.

~~7.5.35~~8.5.35 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.

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

```
        <tp:ebXMLSenderBinding tp:version="1.1">    <!-- 0 or 1 -->
            <tp:ReliableMessaging>                  <!-- 0 or 1 -->
            . . .
            </tp:ReliableMessaging>
            <tp:SenderNonRepudiation>               <!-- 0 or 1 -->
            . . .
            </tp:SenderNonRepudiation>
            <tp:SenderDigitalEnvelope>              <!-- 0 or 1 -->
            . . .
            </tp:SenderDigitalEnvelope>
            <tp:NamespaceSupported>                 <!-- 1 or more -->
            . . .
            </tp:NamespaceSupported>
        </tp:ebXMLSenderBinding>
        <tp:ebXMLReceiverBinding tp:version="1.1">  <!-- 0 or 1 -->
            <tp:ReliableMessaging>                  <!-- 0 or 1 -->
            . . .
            </tp:ReliableMessaging>
            <tp:ReceiverNonRepudiation>             <!-- 0 or 1 -->
            . . .
            </tp:ReceiverNonRepudiation>
            <tp:ReceiverDigitalEnvelope>            <!-- 0 or 1 -->
            . . .
            </tp:ReceiverDigitalEnvelope>
            <tp:NamespaceSupported>                 <!-- 1 or more -->
            . . .
            </tp:NamespaceSupported>
        </tp:ebXMLReceiverBinding>
    </tp:DocExchange>
```

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

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

~~7.5.35.1~~8.5.35.1 docExchangeId attribute

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

~~7.5.36~~8.5.36 ebXMLSenderBinding element

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

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

messaging,

- zero or one ***SenderNonRepudiation*** element which specifies the sender's requirements and certificate for message signing,
- zero or one ***SenderDigitalEnvelope*** element which specifies the sender's 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.

The ***ebXMLSenderBinding*** element has one attribute:

- a REQUIRED ***version*** attribute

~~7.5.3~~6.~~1~~8.5.36.1 version attribute

The REQUIRED ***version*** attribute identifies the version of the ebXML *Message* Service specification being used.

~~7.5.3~~7~~8~~.5.37 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". The following is the element structure:

```
<tp:ReliableMessaging>
    <tp:Retries>5</tp:Retries>
<tp:RetryInterval>PT2H</tp:RetryInterval>
    <tp:PersistDuration>P1D</tp:PersistDuration
<tp:MessageOrderSemantics>Guaranteed</tp:MessageOrderSemantics>
</tp:ReliableMessaging>
```

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

- ~~an OPTIONAL~~zero or one ***Retries*** element,
- ~~an OPTIONAL~~zero or one ***RetryInterval*** element,
- a REQUIRED ***PersistDuration*** element,
- a REQUIRED ***MessageOrderSemantics*** element.

~~7.5.3~~8~~8~~.5.38 Retries and RetryInterval elements

The ***Retries*** and ***RetryInterval*** elements specify the permitted number of retries and the interval, expressed as an XML Schema[XMLSCHEMA-2] duration, between retries of sending a reliably delivered *Message* following a timeout waiting for the *Acknowledgment*. 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 RetryInterval applies to the time between sending of the original message and the first retry, as well as the time between all subsequent retries.

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*.

~~7.5.3~~8.5.39 PersistDuration element

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

~~7.5.4~~8.5.40 MessageOrderSemantics element

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

- "Guaranteed"
- "NotGuaranteed"

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

~~7.5.4~~8.5.41 SenderNonRepudiation element

The ~~OPTIONAL~~ ***SenderNonRepudiation element*** conveys the message sender's requirements and certificate for non-repudiation. 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:

```
<tp:SenderNonRepudiation>
    <tp:NonRepudiationProtocol>
        http://www.w3.org/2000/09/xmldsig#
    </tp:NonRepudiationProtocol>
    <tp:HashFunction>
        http://www.w3.org/2000/09/xmldsig#sha1
    </tp:HashFunction>
    <tp:SignatureAlgorithm>
        http://www.w3.org/2000/09/xmldsig#dsa-sha1
    </tp:SignatureAlgorithm>
    <tp:SigningCertificateRef tp:certId="CompanyA_SigningCert"/>
</tp:SenderNonRepudiation>
```

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

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

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

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

~~7.5.4~~28.5.42 NonRepudiationProtocol element

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

~~7.5.4~~38.5.43 HashFunction element

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

~~7.5.4~~48.5.44 SignatureAlgorithm element

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

~~7.5.4~~58.5.45 SigningCertificateRef element

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

~~7.5.4~~68.5.46 SenderDigitalEnvelope element

The ***SenderDigitalEnvelope*** element provides the sender's requirements for message encryption using the [DIGENV] digital-envelope method. Digital-envelope is a 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:

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

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

The ***SenderDigitalEnvelope*** element contains

- a REQUIRED ***DigitalEnvelopeProtocol*** element,
- a REQUIRED ***EncryptionAlgorithm*** element
- an OPTIONAL zero or one ***EncryptionSecurityDetailsRef*** element.

7.5.478.5.47 DigitalEnvelopeProtocol element

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

7.5.488.5.48 EncryptionAlgorithm element

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

7.5.498.5.49 EncryptionSecurityDetailsRef element

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

7.5.508.5.50 NamespaceSupported element

The OPTIONAL ***NamespaceSupported*** element identifies the namespaces supported by the messaging service implementation and by the business application. 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:

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

7.5.518.5.51 ebXMLReceiverBinding element

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

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

The ***ebXMLReceiverBinding*** element has one attribute:

- a REQUIRED ***version*** attribute (see Section 8.5.36.1~~7.5.36.1~~)

~~7.5.5~~28.5.52 ReceiverNonRepudiation element

The ~~OPTIONAL~~ ***ReceiverNonRepudiation element*** conveys the message receiver's requirements for non-repudiation. 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:

```
<tp:ReceiverNonRepudiation>
    <tp:NonRepudiationProtocol>
        http://www.w3.org/2000/09/xmldsig#
    </tp:NonRepudiationProtocol>
    <tp:HashFunction>
        http://www.w3.org/2000/09/xmldsig#sha1
    </tp:HashFunction>
    <tp:SignatureAlgorithm>
        http://www.w3.org/2000/09/xmldsig#dsa-sha1
    </tp:SignatureAlgorithm>
    <tp:SigningSecurityDetailsRef
tp:certId="CompanyA_MessageSecurity"/>
</tp:ReceiverNonRepudiation>
```

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

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

- a REQUIRED ***NonRepudiationProtocol*** element (see Section 8.5.42~~7.5.42~~),
- a REQUIRED ***HashFunction*** (e.g. SHA1, MD5) element (see Section 8.5.43~~7.5.43~~),
- a REQUIRED ***SignatureAlgorithm*** element (see Section 8.5.44~~7.5.44~~),
- ~~an OPTIONAL~~zero or one ***SigningSecurityDetailsRef*** element

~~7.5.5~~38.5.53 SigningSecurityDetailsRef element

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

~~7.5.5~~48.5.54 ReceiverDigitalEnvelope element

The ***ReceiverDigitalEnvelope*** element provides the receiver's requirements for message encryption using the [DIGENV] digital-envelope method. Digital-envelope is a 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:

```
<tp:ReceiverDigitalEnvelope>
    <tp:DigitalEnvelopeProtocol tp:version="2.0">
        S/MIME
    </tp:DigitalEnvelopeProtocol>
```

```
        <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
        <tp:EncryptionCertificateRef
            tp:certId="CompanyA_EncryptionCert"/>
    </tp:ReceiverDigitalEnvelope>
```

The ***ReceiverDigitalEnvelope*** element contains

- a REQUIRED ***DigitalEnvelopeProtocol*** element (see Section 8.5.477.5.47),
- a REQUIRED ***EncryptionAlgorithm*** element (see Section 8.5.487.5.48),
- a REQUIRED ***EncryptionCertificateRef*** element.

7.5.558.5.55 EncryptionCertificateRef element

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

7.5.568.5.56 OverrideMshActionBinding element

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

7.68.6 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, of 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. It also has an IMPLIED xlink:role attribute which identifies some resource that describes the mime part or its purpose. If present, then it SHALL have a value that is a valid URI in accordance with the [XLINK] specification. The following are examples of SimplePart elements:

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

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

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

7.78.7 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 The following is an example of the ***Packaging*** element which references the example ***SimplePart*** elements given in Section 8.67.6:

```
<!-- Simple ebXML S/MIME Packaging for application-based payload
     encryption -->
<tp:Packaging>
    <tp:ProcessingCapabilities tp:generate="true" tp:parse="true"/>
    <tp:SimplePart tp:id="I001" tp:mimetype="text/xml"/>
    <tp:SimplePart tp:id="I002" tp:mimetype="application/xml"/>
    <tp:CompositeList>
        <tp:Encapsulation
            tp:id="I003"
            tp:mimetype="application/pkcs7-mime"
            tp:mimeparameters="smime-type=&quot;enveloped-data&quot;">
            <Constituent tp:idref="I002"/>
        </tp:Encapsulation>
        <tp:Composite tp:id="I004"
            tp:mimetype="multipart/related"
            tp:mimeparameters="type=&quot;text/xml&quot;
version=&quot;1.0&quot;">
            <tp:Constituent tp:idref="I001"/>
            <tp:Constituent tp:idref="I003"/>
        </tp:Composite>
    </tp:CompositeList>
</tp:Packaging>
```

See "Matching Packaging" in Appendix F Appendix F 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 ***ActionBinding***, by using the IDREF attribute, ***packageId.***

The child elements of the ***Packaging*** element are ***ProcessingCapabilities*****, *SimplePart*,** and ***CompositeList.*** This set of elements MAY can 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.*

7.7.18.7.1 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.

~~**7.7.2SimplePart element**~~

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

~~7.7.3~~8.7.2 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~~ SHALL 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~~ can 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~~ can 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/ 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. 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.

7.88.8 ds:Signature element

The **ds:Signature** element (cardinality zero or one) enables the CPA to be ~~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 9.78.7 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~~ might 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 8.5.4.5~~7.5.4.5~~ for a discussion of times at which validity tests MAY be made.

~~7.9~~8.9 Comment element

The ***CollaborationProtocolProfile*** element ~~MAY~~ contains zero or more ***Comment*** elements. The ***Comment*** element is a textual note that ~~MAY~~ can 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~~ can have a different ***xml:lang*** attribute value. An example of a ***Comment*** element follows:

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

89 CPA Definition

A *Collaboration-Protocol Agreement (CPA)* defines the capabilities that two *Parties* need to 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 87, "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 needed in the *CPA* context. See also Appendix D Appendix D D for the XML Schema, and Appendix B Appendix B B for an example of a *CPA* document.

8.19.1 CPA Structure

Following is the overall structure of the *CPA:*

```
      <CollaborationProtocolAgreement
          xmlns:tp="http://www.oasis-open.org/committees/ebxml-
cppa/schema/cpp-cpa-1_1.xsd"
          xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
          xmlns:xlink="http://www.w3.org/1999/xlink"
          tp:cpaid="YoursAndMyCPA"
          tp:version="1.2">
          <tp:Status tp:value="proposed"/>
          <tp:Start>1988-04-07T18:39:09</Start>
          <tp:End>1990-04-07T18:40:00</End>
          <!-- ConversationConstraints MAY appear 0 or 1 times -->
          <tp:ConversationConstraints
              tp:invocationLimit="100"
              tp:concurrentConversations="4"/>
          <tp:PartyInfo>
          ...
          </tp:PartyInfo>
          <tp:PartyInfo>
          ...
          </tp:PartyInfo>
          <tp:SimplePart>  <!-- one or more -->
          ...
          </tp:SimplePart>
          <tp:Packaging tp:id="N20">  <!-- one or more -->
          ...
          </tp:Packaging>
          <!-- ds:signature MAY appear 0 or more times -->
          <ds:Signature>
          ...
          </ds:Signature>
          <tp:Comment xml:lang="en-GB">any text</Comment> <!-- zero or more -
->
```

```
</tp:CollaborationProtocolAgreement>
```

8.29.2 CollaborationProtocolAgreement element

The ***CollaborationProtocolAgreement*** element is the root element of a *CPA*. It has a REQUIRED ***cpaid*** attribute 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 SHALL 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~~ might 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 8~~7~~, "CPP Definition".

The ***CollaborationProtocolAgreement*** element is comprised of the following child elements, most of which are 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 ***SimplePart*** elements,
- one or more ***Packaging*** elements,
- one or more ***ds:Signature*** elements that provide signing of the *CPA* using the XML Digital Signature[XMLDSIG] standard,
- zero or more ***Comment*** elements.

8.39.3 Status Element

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

```
<tp:Status tp: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 takes the form of a digital signature that is described in Section 9.78.7 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*.

8.49.4 CPA Lifetime

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

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

8.4.19.4.1 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 dateTime 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).

8.4.29.4.2 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 dateTime 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: 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 MAY might 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 can 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.

8.59.5 ConversationConstraints Element

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

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

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

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

8.5.19.5.1 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.

8.5.29.5.2 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.

8.69.6 PartyInfo Element

The general characteristics of the ***PartyInfo*** element are discussed in Section 8.57.5.

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 or more ***PartyId*** elements under each ***PartyInfo*** element. The values of this these elements is are the same as the values of the ***PartyId*** elements in the ebXML *Message* Service specification[ebMS] or similar messaging service specification. One These ***PartyId*** elements SHALL be used within a ***To*** or ***From*** *Header* element of an ebXML *Message*.

8.6.19.6.1 ProcessSpecification element

The ***ProcessSpecification*** element identifies the *Business Collaboration* that the two *Parties* have agreed to perform. There MAY can 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 8.5.37.5.3.

8.79.7 SimplePart element

The ***CollaborationProtocolAgreement*** element SHALL contain one or more ***SimplePart*** elements. See Section 8.67.6 for details of the syntax of the ***SimplePart*** element.

9.8 Packaging element

The ***CollaborationProtocolAgreement*** element SHALL contain one or more ***Packaging*** elements. See Section 8.77.7 for details of the syntax of the ***Packaging*** element.

8.89.9 ds:Signature element

A *CPA* document MAY can 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] SHALL be used to digitally sign the document.

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

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 SHALL 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 8.5.4.57.5.4.5 for a discussion of times at which a *CPA* MAY be validated.

8.8.19.9.1 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.

8.8.1.19.9.1.1 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.

8.8.1.29.9.1.2 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.

8.8.1.39.9.1.3 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.

8.8.1.49.9.1.4 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#sha1
```

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

8.8.1.59.9.1.5 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~~ can 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~~ can include the ***id*** attribute, type ID, by which this ***ds:Reference*** element ~~MAY be~~is referenced from a ***ds:Signature*** element.

8.8.1.69.9.1.6 ds:Transform element

The ***ds:Reference*** element for the *CPA* or *CPP* document SHALL include a descendant ***ds:Transform*** element that excludes the containing ***ds:Signature*** element and all its descendants. This exclusion is achieved by means of specifying the ***ds:Algorithm*** attribute of the ***Transform*** element as
"http://www.w3.org/2000/09/xmldsig#enveloped-signature".

For example:

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

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

8.8.1.79.9.1.7 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.

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

8.99.10 Comment element

The ***CollaborationProtocolAgreement*** element ~~MAY~~ contains zero or more ***Comment*** elements. See Section 8.9~~7.9~~ for details of the syntax of the ***Comment*** element.

8.109.11 Composing a CPA from Two CPPs

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

8.10.19.11.1 ID Attribute Duplication

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

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

```
    <tp:Certificate
    tp:cert~~i~~Id="EncryptionCert"><ds:KeyInfo/></tp:Certificate>
```

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

```
<tp:Certificate
tp:cert~~i~~Id="CompanyA_EncryptionCert"><ds:KeyInfo/></tp:Certificate>
```

~~8.11~~9.12 Modifying Parameters of the Process-Specification Document Based on Information in the *CPA*

A *Process-Specification* document contains a number of parameters, expressed as XML attributes. An example is the security attributes that are counterparts of the attributes of the *CPA* ***BusinessProcessCharacteristics*** element. The values of these attributes can be considered to be default values or recommendations. When a *CPA* is created, the *Parties* ~~MAY~~ might 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* ***BusinessProcessCharacteristics*** 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.

910 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.

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

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

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

[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.

[RFC2251] Lightweight Directory Access Protocol (v3); Mark Wahl, Tim Howes, Steve Kille.

[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.

[X12] ANSI X12 Standard for Electronic Data Interchange, X12 Standard Release 4050, December 2001

[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.

~~10~~11 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 D and available from

`http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-1_1.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~~ might be available to allow vendors to test their products for interface compatibility, conformance, and interoperability.

1112 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.

1213 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 (Chair)
Cyclone Commerce
17767 North Perimeter Dr., Suite 103
Scottsdale, AZ 85255
USA
Phone: 480-627-1800
email: dmoberg@columbus.rr.com

Tony Weida (Coordinating Editor)
Edifecs
2310 130th Ave. NE, Suite 100
Bellevue, WA 98005
535 West 110th St., #4J
New York, NY 10025
USA
Phone: 212-678-5265
email: TonyW@edifecs.comrweida@hotmail.com

Notices

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

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

Copyright Statement

Appendix A Example of CPP Document (Non-Normative)

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

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

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

Appendix B Example of CPA Document (Non-Normative)

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

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

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

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

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

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<ProcessSpecification xmlns="http://www.ebxml.org/BusinessProcess"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ebxml.org/BusinessProcess ebBPSS.xsd"
name="PIP3A4RequestPurchaseOrder" uuid="bpid:RosettaNet:3A4$2.0</"
version="R02.00">
      <Documentation>This PIP enables a buyer to issue a purchase order and
obtain a quick response from the provider that acknowledges which of the
purchase order product line items are accepted, rejected, or
pending</Documentation>
      <!--Purchase order Request Document-->
      <BusinessDocument name="Puchase Order Request"
nameID="Pip3A4PurchaseOrderRequest" specificationLocation="%SYSTEM
/XMLPIPVALIDATION/3A4/PurchaseOrderRequest.xsd">
            <Documentation>The document is an XSD file that specifies the
rules for creating the XML document for the business action of requesting a
purchase order</Documentation>
      </BusinessDocument>
      <BusinessDocument name="Puchase Order Confirmation"
nameID="Pip3A4PurchaseOrderConfirmation" specificationLocation="%SYSTEM
/XMLPIPVALIDATION/3A4/PurchaseOrderConfirmation.xsd">
            <Documentation>The document is an XSD file that specifies the
rules for creating the XML document for the business action of making a
purchase order confirmation</Documentation>
      </BusinessDocument>
      <BusinessTransaction name="Request Purchase Order"
nameID="RequestPurchaseOrder_BT">
            <RequestingBusinessActivity name="Purchase Order Request Action"
nameID="PurchaseOrderRequestAction"  isAuthorizationRequired ="true"
isIntelligibleCheckRequired="true" isNonRepudiationReceiptRequired="true"
isNonRepudiationRequired="true" timeToAcknowledgeReceipt="P0Y0M0DT2H0M0S">
                  <DocumentEnvelope businessDocument="Puchase Order Request"
businessDocumentIDRef="Pip3A4PurchaseOrderRequest" isAuthenticated="true"
isConfidential="true" isTamperProof="true"/>
            </RequestingBusinessActivity>
            <RespondingBusinessActivity name="Purchase Order Confirmation
Action" nameID="PurchaseOrderConfirmationAction"
isAuthorizationRequired="true" isIntelligibleCheckRequired="true"
isNonRepudiationReceiptRequired="false" isNonRepudiationRequired="true"
timeToAcknowledgeReceipt="P0Y0M0DT2H0M0S">
                  <DocumentEnvelope businessDocument="Purchase Order
Confirmation" businessDocumentIDRef="Pip3A4PurchaseOrderConfirmation"
isAuthenticated="true" isConfidential="true" isPositiveResponse="true"
isTamperProof="true"/>
            </RespondingBusinessActivity>
      </BusinessTransaction>
```

```
      <BinaryCollaboration name="Request Purchase Order"
nameID="RequestPurchaseOrder_BC">
            <InitiatingRole name="Buyer" nameID="Buyer"/>
            <RespondingRole name="Seller" nameID="Seller"/>
            <BusinessTransactionActivity name="Request Purchase Order"
nameID="RequestPurchaseOrder_BTA"  businessTransaction="Request Purchase
Order" businessTransactionIDRef="RequestPurchaseOrder_BT"
fromAuthorizedRole="Buyer" fromAuthorizedRoleIDRef="Buyer"
toAuthorizedRole="Seller" toAuthorizedRoleIDRef="Seller"
isLegallyBinding="true" timeToPerform="P0Y0M0DT24H0M0S" isConcurrent="false"/>
      </BinaryCollaboration>
</ProcessSpecification>
```

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

This XML Schema document is available as an ASCII file at:
http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-1_1.xsd

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

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

in the elements CollaborationProtocolProfile and CollaborationProtocolAgreement, and in the form

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

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

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

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~~ might propose a *CPA* that satisfies enough of the requirements for "a good enough" implementation. User input ~~MAY~~ can be solicited to determine what is a good enough implementation, and so ~~MAY~~ might be as varied as there are user configuration options to express preferences. In practice, compromises ~~MAY~~ could 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~~ might have access to local user information not advertised in the *CPP* that ~~MAY~~ could contribute to the *CPA* that is formed. A user ~~MAY~~ might have chosen to only advertise those system capabilities that reflect nondeprecated capabilities. For example, a user ~~MAY~~ might 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~~ might 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~~could, 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~~ could 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~~ might 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 specified ability to supply non-repudiation of receipt, but might still be acceptable for *Business* reasons. A system might not be able to handle all the processing needed 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~~ might 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~~ could be sufficient to allow collaboration, and to enjoy payoffs from increased levels of automation.

In practice, the *CPA* formation process ~~MAY~~ might 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.

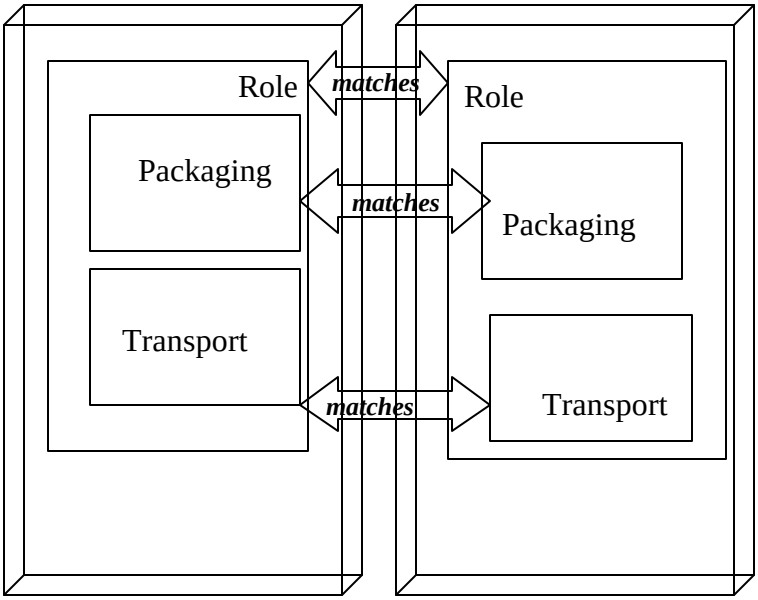
For vividness, we will suppose that our example agent wishes to play the role of supplier and seeks to find one of its current customers to begin a Purchase Order *Business Collaboration* in which the intended player plays a complementary role. For simplicity, we assume that the information about capabilities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

```
      <tp:Transport tp:transportId="buyerid001">
          <tp:SendingProtocol>HTTP</tp:SendingProtocol>
          <tp:ReceivingProtocol>HTTP</tp:ReceivingProtocol>
          <tp:Endpoint tp:uri="https://www.buyername.com/po-response"
              tp:type="allPurpose"/>
          <tp:TransportSecurity>
              <tp:Protocol tp:version="1.0">TLS</tp:Protocol>
              <tp:CertificateRef
tp:certId=certid001">BuyerName</tp:CertificateRef>
          </tp:TransportSecurity>
      </tp:Transport>
```

and our seller has a Transport entry:

```
      <tp:Transport tp:transportId="sellid001">
          <tp:SendingProtocol>HTTP</tp:SendingProtocol>
          <tp:ReceivingProtocol>HTTP</tp:ReceivingProtocol>
          <tp:Endpoint tp:uri="https://www.sellername.com/pos_here"
              tp:type="allPurpose"/>
          <tp:TransportSecurity>
              <tp:Protocol tp:version="1.0">TLS</Protocol>
              <tp:CertificateRef
tp:certId="certid002">Sellername</tp:CertificateRef>
```

```
            </tp:TransportSecurity>
        </tp: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~~ might 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~~ might 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:

```
      <tp:Packaging tp:id="I1001">
          <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>
          <tp:SimplePart tp:id="P1" tp:mimetype="text/xml"/>
          <tp:NamespaceSupported
tp:location="http://schemas.xmlsoap.org/soap/envelope/"
tp:version="1.1">http://schemas.xmlsoap.org/soap/envelope</tp:NamespaceSuppor
ted>
          <tp:NamespaceSupported
tp:location="http://www.ebxml.org/namespaces/messageHeader" tp:version="1.0">
http://www.ebxml.org/namespaces/messageHeader</NamespaceSupported>
          <tp:NamespaceSupported
tp:location="http://www.w3.org/2000/09/xmldsig#"
tp:version="1.0">http://www.w3.org/2000/09/xmldsig#</tp:NamespaceSupported>
          <tp:SimplePart tp:id="P2" tp:mimetype="application/xml"/>
          <tp:CompositeList>
              <tp:Composite tp:mimetype="multipart/related" tp:id="P3"
                  tp:mimeparameters="type=text/xml">
                  <tp:Constituent tp:idref="P1"/>
                  <tp:Constituent tp:idref="P2"/>
              </tp:Composite>
          </tp:CompositeList>
      </tp:Packaging>
      <tp:Packaging id="I2001">
          <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>
          <tp:SimplePart tp:id="P11" tp:mimetype="text/xml"/>
          <tp:SimplePart tp:id="P12" tp:mimetype="application/xml"/>
          <tp:CompositeList>
              <tp:Composite tp:mimetype="multipart/related" tp:id="P13"
                  tp:mimeparameters="type=text/xml">
                  <tp:Constituent tp:idref="P11"/>
                  <tp:Constituent tp:idref="P12"/>
              </tp:Composite>
          </tp:CompositeList>
      </tp: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 might 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~~ might be possible to extract the data and arrive at a partial implementation that falls short with respect to non-repudiation. 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~~ might 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~~ might eventually be capable of these adaptations, but it is to be expected that human assistance will be needed for such situations in the near term.

Of course, an implementation ~~MAY~~ could 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 specified 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 might occur. At present, these ties are left to be resolved by a negotiation process not discussed here.

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

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

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

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

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

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

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

Appendix H Glossary of Terms

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

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

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

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

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

3811

3812