Collaboration-Protocol Profile and Agreement Specification
Version 0.93

ebXML Trading-Partners Team

03/19/01 5:38 PM

1 Status of this Document

This document specifies an ebXML WORK IN PROGRESS for the eBusiness community.

Distribution of this document is unlimited.

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

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

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

Previous version:
http://www.ebxml.org/project_teams/trade_partner/private/cpa-cpp-spec-0.911.doc
2 ebXML participants

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

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


# Table of Contents

1 Status of this Document ................................................................. 1
2 ebXML participants .......................................................................... 2
3 Table of Contents ........................................................................... 3
4 Introduction .................................................................................. 5
  4.1 Summary of Contents of Document ............................................. 5
  4.2 Document Conventions ............................................................... 5
  4.3 Definitions .................................................................................. 6
  4.4 Audience ................................................................................... 6
  4.5 Assumptions ............................................................................. 6
  4.6 Related Documents .................................................................. 6
5 Design Objectives ........................................................................... 7
6 System Overview ............................................................................ 8
  6.1 What This Specification Does ................................................... 8
  6.2 How the CPA Works .................................................................. 11
  6.3 Where the CPA May Be Implemented ....................................... 12
  6.4 Definition and Scope ................................................................. 13
7 CPP Definition .............................................................................. 14
  7.1 CPP Structure ......................................................................... 15
  7.2 CollaborationProtocolProfile element ......................................... 15
  7.3 PartyInfo Element ...................................................................... 15
     7.3.1 PartyId element ................................................................. 16
     7.3.2 PartyRef element ............................................................. 17
     7.3.3 CollaborationRole element ................................................ 18
     7.3.4 ProcessSpecification Element ............................................. 20
     7.3.5 Role element .................................................................... 22
     7.3.6 ServiceBinding element ..................................................... 23
     7.3.7 Override element ............................................................... 26
     7.3.8 Certificate element ........................................................... 27
     7.3.9 DeliveryChannel element .................................................. 27
     7.3.10 Characteristics element .................................................... 29
     7.3.11 Transport element ......................................................... 30
     7.3.12 Transport Protocol ......................................................... 30
     7.3.13 Endpoint Element ........................................................... 31
     7.3.14 Transport Protocols ......................................................... 32
     7.3.15 Transport Security ........................................................... 34
  7.4 DocExchange element ............................................................... 35
     7.4.1 docExchangeld attribute ...................................................... 36
     7.4.2 ebXMLBinding element ....................................................... 36
     7.4.3 version attribute ............................................................... 36
     7.4.4 MessageEncoding element ................................................ 36
     7.4.5 ReliableMessaging element ............................................... 37
     7.4.6 NonRepudiation element ................................................... 38
     7.4.7 DigitalEnvelope element .................................................. 39
     7.4.8 NamespaceSupported element ........................................... 40
  7.5 dS:Signature element ............................................................... 40
  7.6 Comment element ..................................................................... 40
8 CPA Definition ............................................................................. 42
  8.1 CPA Structure ......................................................................... 42
  8.2 CollaborationProtocolAgreement element .................................. 42
  8.3 CPAType element ..................................................................... 43
  8.4 Status element ......................................................................... 44
8.5 CPA Lifetime .................................................................................................................................................. 44
8.5.1 Start element .............................................................................................................................................. 44
8.5.2 End element ............................................................................................................................................... 44
8.6 ConversationConstraints element ........................................................................................................... 45
8.6.1 invocationLimit attribute ...................................................................................................................... 45
8.6.2 concurrentConversations attribute ...................................................................................................... 45
8.7 PartyInfo element ......................................................................................................................................... 46
8.7.1 ProcessSpecification element .................................................................................................................. 46
8.8 ds:Signature element ....................................................................................................................................... 46
8.8.1 Persistent Digital Signature .................................................................................................................... 47
8.9 Comment element ......................................................................................................................................... 48
8.10 Composing a CPA from Two CPPs ........................................................................................................... 49
8.10.1 ID Attribute Duplication ....................................................................................................................... 49
8.11 Modifying Parameters of the Process-Specification Document Based on Information in the CPA ................................................................................................................................. 49
9 References ....................................................................................................................................................... 51
10 Conformance .................................................................................................................................................. 54
11 Disclaimer ....................................................................................................................................................... 55
Contact Information .................................................................................................................................................. 56
Copyright Statement ............................................................................................................................................... 57
Appendix A Example of CPP Document (Non-Normative) ............................................................................... 58
Appendix B Example of CPA Document (Non-normative) ................................................................................ 60
Appendix C DTD Corresponding to Complete CPP/CPA Definition (Normative) ............................................ 64
Appendix D XML Schema Document Corresponding to Complete CPA Definition (Normative) .................... 67
Appendix E Formats of Information in the CPP and CPA (Normative) ............................................................. 74
Appendix F Composing a CPA from Two CPPs (Non-Normative) ................................................................... 75
4 Introduction

4.1 Summary of Contents of Document

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

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

This specification is a draft standard for trial implementation. This specification contains the detailed definitions of the Collaboration-Protocol Profile (CPP) and the Collaboration-Protocol Agreement (CPA).

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

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

4.2 Document Conventions

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

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

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

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

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

4.4 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.

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

4.6 Related Documents
Related documents include ebXML Specifications on the following topics:
- ebXML Technical Architecture Specification[TECHARCH]
- ebXML Message Service Specification[MSSPEC]
- ebXML Business Process Specification SchemaBPMSPEC]
- ebXML Glossary [EBXMLGLOSS]
- ebXML Core Components Specification[EBXMLCC]
- ebXML Registry and Repository Specification[REGREP]

See Section 9 for the complete list of references.
5 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 CPA defines the way two Parties will interact in performing the chosen Collaborative Process. Both Parties SHALL use identical copies of the CPA to configure their run-time systems. This assures that they are compatibly configured to exchange Messages whether or not they have obtained their run-time systems from the same vendor. The configuration process MAY be automated by means of a suitable tool that reads the CPA and performs the configuration process.

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

It is an objective of this specification that a CPA SHALL be capable of being composed by intersecting the respective CPPs of the Parties involved. The resulting CPA SHALL contain only those elements that are in common, or compatible, between the two parties. Variable quantities, such as number of retries of errors, are then negotiated between the two Parties. The design of the CPP and CPA schemata facilitates this composition/negotiation process. However, the composition and negotiation processes themselves are outside the scope of this specification. Appendix 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 to more automated means of setting up business relationships and doing business under them.
6 System Overview

6.1 What This Specification Does

The exchange of information between two Parties requires each Party to know the other Party's supported Collaborative Processes, the other Party's role in the Collaborative Process, 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 Collaborative Process, can be described by a Collaboration-Protocol Profile (CPP). The agreement between the Parties can be expressed as a Collaboration-Protocol Agreement (CPA).

Figure 1 illustrates the relationships between a CPP and two Process-Specification documents.
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 Process
that the Party can perform.

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

The CPP describes the capabilities of an individual Party. A CPA describes the capabilities that
two Parties have agreed to use to perform a particular Business Process. 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 (TPA). 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.

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-to-peer relationship rather than a client-server relationship.

6.2 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' CPPs. In general, this section is an overview of a possible procedure
and is not to be considered a normative specification. See Appendix F "Composing a CPA from
Two CPPs (Non-Normative)" for more information.

Figure 2 illustrates forming a CPP. Party A tabulates the information to be placed in a
repository for the discovery process, constructs a CPP that contains this information, and enters
it into an ebXML Registry or similar repository.
In figure 3, Party A and Party B use their CPPs to jointly construct a single copy of a CPA by calculating the intersection of the information in their CPPs. The resulting CPA defines how the two parties will behave in performing their Collaborative Process.
Figure 4 illustrates the entire process. The steps are listed at the left. The end of the process is that the two Parties configure their systems from identical copies of the agreed CPA and they are then ready to do business.

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

1. Any company may register its CPPs to an ebXML Registry.
2. Company B discovers trading partner A (Seller) by searching CPPs in the Repository and downloads CPP(A) to Company-B’s server.
3. Company B makes CPA(A,B) and sends CPA(A,B) to Company A.
4. Companies 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. Companies A and B configure their runtime systems with the information in the CPA.
6. Do Business (e.g. submit purchase orders).

### 6.3 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 Collaborative Process 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 Process. Typically, the software that performs the Messages exchanges and otherwise supports the interactions between the Parties is middleware that can support any selected Business Process. One component of this middleware is the ebXML Message Service Handler[MSSPEC]. In this specification, the term "runtime system" or "runtime 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 a response Message from the other Party. The Process-Specification document defines, among other things, the request and response Messages for each Business Transaction and the order in which the Business Transactions are REQUIRED to occur. See [BPMSPEC] for a detailed explanation.


A new conversation is started each time a new unit of business is started. The Business Process 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 Where the CPA May Be Implemented

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

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

6.5 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 runtime support where these notes serve to clarify the CPP and CPA definitions. See section 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 DTD and XML Schema documents defined herein. It is, however, important to understand that the value of this specification lies in its enabling a runtime system that supports electronic commerce between two Parties under the guidance of the information in the CPA.
7 CPP Definition

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

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

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

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

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

Document-Exchange layer - The document-exchange layer accepts a business from the Process-Specification layer at one Party, encrypts it if specified, adds a digital signature for nonrepudiation if specified, and passes it to the transport layer for transmission to the other Party. It performs the inverse steps for received 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 Messages between two Parties is defined by the ebXML Message Service Specification MSSPEC] or other similar messaging service.

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

It should be understood that the functional layers encompassed by the CPP have no understanding of the contents of the payload of the business documents.
7.1 CPP Structure

This section describes 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.

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

7.2 CollaborationProtocolProfile element

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

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

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

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

7.3 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:

```xml
<PartyInfo>
    <PartyId type="..."> <!--one or more-->
```

Collaboration-Protocol Profile and Agreement Specification

Copyright © ebXML 2001. All Rights Reserved.
The **PartyInfo** element consists of the following child elements:

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

### 7.3.1 PartyId element

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

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

The **PartyId** element has a single **IMPLIED** attribute: **type** that has a string value.
If the `type` attribute is present, then it provides a scope or namespace for the content of the `PartyId` element.

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

The following example illustrates two URI references.

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

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

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

### 7.3.2 PartyRef element

The `PartyRef` element provides a link, in the form of a URI, to additional information about the Party. Typically, this would be the URL from which the information can be obtained. The information might be at the Party's web site or in a publicly accessible repository such as an ebXML Registry, a UDDI repository, or an LDAP directory. Information available at that URI MAY include contact names, addresses, and phone numbers, and perhaps more information about the Business Processes that the Party supports. This information MAY be in the form of an ebXML Core Component[EBXMLCC]. 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.

#### 7.3.2.1 xlink:type attribute

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

#### 7.3.2.2 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.

An example of the `PartyRef` element is:
7.3.3 CollaborationRole element

```xml
<CollaborationRole id="N11">
  <ProcessSpecification name="BuySell" version="1.0">
    ...
  </ProcessSpecification>
  <Role name="buyer" xlink:href="/ourInfo.html"/>
  <CertificateRef certId = "N03"/>
  <ServiceBinding name="some process" channelId="N02">
    <Override action="OrderAck" channelId="N05" xlink:type="locator" xlink:href="/ourInfo.html"/>
    <Packaging> ... </Packaging>
  </ServiceBinding>
</CollaborationRole>
```

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

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

The CollaborationRole element SHALL consist of the following child elements: a REQUIRED ProcessSpecification element, a REQUIRED Role element, zero or one CertificateRef element, and one or more ServiceBinding elements. The ProcessSpecification element identifies the Process-Specification document that defines such role. The Role element identifies which role the Party is capable of supporting. The CertificateRef element identifies the certificate to be used. Each ServiceBinding element provides a binding of the role to a default DeliveryChannel. The default DeliveryChannel describes the receive properties of all Message traffic that is to be received by the Party within the context of the role in the identified Process-Specification document. Alternative DeliveryChannels MAY be specified for specific purposes, using Override elements as described below.
When there are more than one ServiceBinding child elements of a CollaborationRole, then the order of the ServiceBinding elements SHALL be treated as signifying the Party's preference starting with highest and working towards lowest. The default delivery channel for a given Process-Specification document is the delivery channel identified by the highest-preference ServiceBinding element that references the particular Process-Specification document.

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

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

NOTE: An implementation MAY provide the capability of dynamically assigning delivery channels on a per Message basis during performance of the Business Process. The delivery channel selected would be chosen, based on present conditions, from those identified by ServiceBinding elements that refer to the Business Process 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.

7.3.3.1 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.

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

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

NOTE: This certId attribute relates to the authorizing role in the Process Specification while the certificates identified in the delivery-channel description relate to Message exchanges.
7.3.4 ProcessSpecification Element


The syntax of the ProcessSpecification element is:

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

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

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

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

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

7.3.4.1 name attribute

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

7.3.4.2 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).

7.3.4.3 xlink:type attribute

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

7.3.4.4 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].

7.3.4.5 ds:Reference Element

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

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

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

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

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

A ds:Reference element in a ProcessSpecification element has implications for CPP validity:
• A CPP MUST be considered invalid if any \textit{ds:Reference} element within a \textit{ProcessSpecification} element fails reference validation as defined by the XML Digital Signature specification[XMLDSIG].

• A CPP MUST be considered invalid if any \textit{ds:Reference} within it cannot be dereferenced.

Other validity implications of such \textit{ds:Reference} elements are specified in the description of the \textit{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 \textit{ds:Reference} element's \textit{ds:Transforms} subelement within a \textit{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 specification[BPMSPEC].

7.3.5 Role element

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

The \textit{Role} element has the following attributes:

• a REQUIRED \textit{name} attribute,

• a FIXED \textit{xlink:type} attribute,

• a REQUIRED \textit{xlink:href} attribute.
7.3.5.1 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 [BPMSPEC] that is referenced by the ProcessSpecification element depending upon which element is the "root" (highest order) of the process referenced:

- initiator attribute of the binary-collaboration element,
- responder attribute of the binary-collaboration element,
- from attribute of the business-transaction-activity element,
- to attribute of the business-transaction-activity element,
- from attribute of the collaboration-activity element,
- to attribute of the collaboration-activity element,
- name attribute of the business-partner-role element.

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

7.3.5.3 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 Process.

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

An example of the ServiceBinding element is:

```xml
<ServiceBinding name="SomeProcess" channelId="X03">
  <Packaging> <!--one or more-->
    ...
  </Packaging>
  <Override action="OrderAck"
    channelId="X04"
    xlink:type="locator"
    xlink:href="..."/> <!--zero or more-->
</ServiceBinding>
```

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

The ServiceBinding element has the following attributes:

- a REQUIRED name attribute,
- a REQUIRED channelId attribute.

7.3.6.1 name attribute
The value of the REQUIRED name attribute is a string value that labels the ServiceBinding element. The value of the name attribute SHALL be used as the value of the Service element in
the ebXML Message Header [MSSSPEC].

#### 7.3.6.2 channelId attribute

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

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

The packaging subtree 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.

Following is an example of the `Packaging` element:

```xml
<Packaging>
    <!--one or more-->
    <!--The triplet of child elements of Packaging MAY appear one or more times-->
    <ProcessingCapabilities parse="..." generate="...">
      <SimplePart id="id" mimetype="type"/>
    </ProcessingCapabilities>

    <!--The child of CompositeList is an enumeration of either Composite or Encapsulation. The enumeration MAY appear one or more times, with the two elements intermixed-->
    <CompositeList>
      <Composite mimetype="type" id="name" mimeparameters="parameter">
        <Constituent idref="name"/>
      </Composite>
      <Encapsulation mimetype="type" id="name">
        <Constituent idref="name"/>
      </Encapsulation>
    </CompositeList>
</Packaging>
```

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

The `ProcessingCapabilities` element has two attributes with REQUIRED 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.

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: \textit{id} and \textit{mimetype}. The \textit{id} attribute, type ID, provides the value that will be used later to reference this Message part when specifying how the parts are packaged into composites, if composite packaging is present. The \textit{mimetype} attribute provides the actual value of the Content-type for the simple Message part being specified.

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 multipart) or encapsulated within security-related MIME content-types. The CompositeList element MAY be omitted from Packaging when no security encapsulations or composite multipart are used. When the CompositeList element is present, the content model for CompositeList is a repeatable sequence of choices of Composite or Encapsulation elements. The Composite and Encapsulation elements MAY appear intermixed as desired.

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

The Composite element has the following attributes:

- a REQUIRED \textit{mimetype} attribute,
- a REQUIRED \textit{id} attribute,
- an IMPLIED \textit{mimeparameters} attribute.

The \textit{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 \textit{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 \textit{mimeparameters} attribute provides the values of any significant MIME parameter (such as “type=application/vnd.eb+xml”) that is needed to understand the processing demands of the content-type.

The Composite element has one child element, Constituent.

The Constituent element has one REQUIRED attribute, \textit{idref}, type IDREF, and has an EMPTY content model. The \textit{idref} attribute has as its value the value of the \textit{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 characterized. For convenience, we tag all such security structures under Encapsulation, 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* attribute and the *Composite* element have child elements consisting of a *Constituent* element or of a repeatable sequence of *Constituent* elements, respectively.

### 7.3.7 Override element

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

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

The *Override* element has the following attributes:

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

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

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

#### 7.3.7.1 action attribute

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

#### 7.3.7.2 channelId attribute

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

7.3.7.3 xlink:href attribute

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

7.3.7.4 xlink:type attribute

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

7.3.8 Certificate element

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

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

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

7.3.8.1 certId attribute

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

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

7.3.8.2 ds:KeyInfo element

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

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

7.3.9 DeliveryChannel element

A delivery channel is a combination of a Transport element and a DocExchange element that describes the Party's Message-receiving characteristics. The CPP SHALL contain one or more DeliveryChannel elements, one or more Transport elements, and one or more DocExchange elements. Each delivery channel MAY refer to any combination of a DocExchange element and a Transport element. The same DocExchange element or the same Transport element MAY be referred to by more than one delivery channel. Two delivery channels MAY use the same transport protocol and the same document-exchange protocol and differ only in details such as communication addresses or security definitions. Figure 5 illustrates three delivery channels.
The delivery channels have ID attributes with values "DC1", "DC2", and "DC3". Each delivery channel contains one transport definition and one document-exchange definition. Each transport definition and each document-exchange definition also has a name as shown in the figure. Note that delivery-channel DC3 illustrates that a delivery channel MAY refer to the same transport definition and document-exchange definition used by other delivery channels but a different combination. In this case delivery-channel DC3 is a combination of transport definition T2 (also referred to by delivery-channel DC2) and document-exchange definition X1 (also referred to by delivery-channel DC1).

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

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

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

The DeliveryChannel element has the following attributes:
• a REQUIRED `channelId` attribute,
• a REQUIRED `transportId` attribute,
• a REQUIRED `docExchangeId` attribute.

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

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

7.3.9.2 `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.

7.3.9.3 `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.

7.3.10 Characteristics element
The `Characteristics` element describes the security characteristics provided by the delivery channel. The `Characteristics` element has the following attributes:

• an IMPLIED `nonrepudiationOfOrigin` attribute,
• an IMPLIED `nonrepudiationOfReceipt` attribute,
• an IMPLIED `secureTransport` attribute,
• an IMPLIED `confidentiality` attribute,
• an IMPLIED `authenticated` attribute,
• an IMPLIED `authorized` attribute.

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

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

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

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

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

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

The overall structure of the Transport element is as follows:

```xml
<Transport transportId = "N05">
  <SendingProtocol version = "1.1">HTTP</SendingProtocol>
  <ReceivingProtocol version = "1.1">HTTP</ReceivingProtocol>
  <Endpoint uri="http://example.com/servlet/ebxmlhandler" type = "request"/>
  <TransportSecurity> <!--0 or 1 times--> 
    <Protocol version = "3.0">SSL</Protocol>
    <CertificateRef certId = "N03"/>
  </TransportSecurity>
</Transport>
```

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

7.3.12 Transport Protocol
Supported communication protocols are HTTP, SMTP, and FTP. The CPP MAY specify as many protocols as the Party is capable of supporting.
NOTE: It is the aim of this specification to enable support for any transport capable of carrying MIME content using the vocabulary defined herein.

7.3.12.1 SendingProtocol element

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

7.3.12.2 ReceivingProtocol element

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

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

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

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

7.3.13 Endpoint Element

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

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

7.3.14 Transport Protocols

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

7.3.14.1 HTTP

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

Following is an example of an HTTP endpoint:

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

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

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

7.3.14.2 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:

```xml
<Endpoint uri="mailto:ebxmlhandler@example.com"
   type = "request"/>
```
SMTP with MIME automatically encodes or decodes the document as required, on a link-by-link basis, and presents the decoded document to the destination document-exchange function. If the application design is such that the choices in the `documentExchange` element and the `ProcessSpecification` element are intended to be independent of the choice of transport protocol, it is permissible to specify a `MessageEncoding` element under the `DocExchange` element.

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. If the data are encoded in the document-exchange level (`MessageEncoding`), the information that the data are already encoded SHOULD be passed to the mail user agent.

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.

### 7.3.14.3 FTP

FTP is File Transfer Protocol [RFC959].

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

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

Since FTP must be compatible across all implementations, the FTP for ebXML will use the minimum sets of commands and parameters available for FTP as specified in [RFC959], section 5.1, and modified in [RFC1123], section 4.1.2.13. The mode SHALL be stream only and the type MUST be either ASCII Non-print (AN), Image (I) (binary), or Local 8 (L 8) (binary between 8-bit machines and machines with 36 bit words – for an 8-bit machine Local 8 is the same as Image).
Stream mode closes the data connection upon end of file. The server side FTP MUST set control to “PASV” before each transfer command to obtain a unique port pair if there are multiple third party sessions.

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

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

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

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

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

### 7.3.15 Transport Security

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

Following is the syntax:

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

### 7.3.15.1 Protocol element

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

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

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

### 7.3.15.2 CertificateRef element

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

### 7.3.15.3 Specifics for HTTP

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

### 7.4 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[MSSPEC]).

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

```xml
<DocExchange docExchangeId = "N06">
    <ebXMLBinding version = "0.92">
        <MessageEncoding> <!--cardinality 0 or 1-->
            ...
        </MessageEncoding>
        <ReliableMessaging> <!--cardinality 0 or 1-->
            ...
        </ReliableMessaging>
        <NonRepudiation> <!--cardinality 0 or 1-->
            ...
        </NonRepudiation>
        <DigitalEnvelope> <!--cardinality 0 or 1-->
            ...
        </DigitalEnvelope>
        <NamespaceSupported> <!-- 1 or more -->
    </ebXMLBinding>
</DocExchange>
```
The DocExchange element of the CPP defines the properties of the messaging service to be used with CPAs composed from the CPP.

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

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

7.4.1 docExchangeId attribute

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

7.4.2 ebXMLBinding element

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

- zero or one MessageEncoding element which specifies how Messages are to be encoded by the document-exchange layer,
- zero or one ReliableMessaging element which specifies the characteristics of reliable messaging,
- zero or one NonRepudiation element which specifies the requirements for signing the Message,
- zero or one DigitalEnvelope element which specifies the requirements for encryption by the digital-envelopeDIGENV method,
- zero or more NamespaceSupported elements which identify any namespace extensions supported by the messaging service implementation.

7.4.3 version attribute

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

7.4.4 MessageEncoding element

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

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

### 7.4.5 ReliableMessaging element

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

The default that applies if the **ReliableMessaging** element is omitted is "BestEffort". See Section 7.4.5.1. The following is the element structure:

```xml
<ReliableMessaging deliverySemantics="OnceAndOnlyOnce"
    idempotency="false"
    persistDuration="30S">
    <!--The pair of elements Retries, RetryInterval has cardinality 0 or 1-->
    <Retries>5</Retries>
    <RetryInterval>60</RetryInterval> <!--time in seconds-->  <!----time in seconds-->
</ReliableMessaging>
```

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

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

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

- a REQUIRED **deliverySemantics** attribute,
- a REQUIRED **idempotency** attribute,
- a REQUIRED **persistDuration** element.

#### 7.4.5.1 deliverySemantics attribute

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

- "OnceAndOnlyOnce",
- "BestEffort".

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

#### 7.4.5.2 idempotency attribute

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

**NOTE:** Additional testing for duplicates MAY take place in the business application based on application information in the *Messages* (e.g. purchase order number).
The idempotency test checks whether a Message duplicates a prior Message between the same
client and server. If the idempotency test is requested, the receiving messaging service passes a
duplicate Message to the recipient Business Process with a "duplicate" indication. The receiving
messaging service also returns a "duplicate" indication to the sender of the duplicate.

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

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

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

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

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

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

    <NonRepudiation>
        <Protocol version = "1.0">XMLDSIG</Protocol>
        <HashFunction>sha1</HashFunction>
        <SignatureAlgorithm>rsa</SignatureAlgorithm>
        <CertificateRef certId = "N03"/>
    </NonRepudiation>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

When a _CPA_ is composed from two _CPPs_, all _Comment_ elements from both _CPPs_ SHALL be
included in the _CPA_ unless the two _Parties_ agree otherwise.
8 CPA Definition

A Collaboration-Protocol Agreement (CPA) defines the capabilities that two Parties must agree 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 7, "CPP Definition". In general, this section does not repeat that information. The discussions in this section are limited to those elements that are not in the CPP or for which additional discussion is required in the CPA context. See also Appendix C and Appendix D for the DTD and XML Schema, respectively, and Appendix B for an example of a CPA document.

8.1 CPA Structure

```
<CollaborationProtocolAgreement id = "N01"
   xmlns="http://www.ebxml.org/namespaces/tradePartner"
   xmlns:bpm="http://www.ebxml.org/namespaces/businessProcess"
   xmlns:ds = "http://www.w3.org/2000/09/xmlsig#"
   xmlns:xlink = "http://www.w3.org/1999/xlink">
   <CPAType> <!--MAY appear 0 or 1 times-->
   ...
   </CPAType>
   <Status value = "proposed"/>
   <Start>1988-04-07T18:39:09</Start>
   <End>1990-04-07T18:40:00</End>
   <!--ConversationConstraints MAY appear 0 or 1 times-->
   <ConversationConstraints invocationLimit = "100"
      concurrentConversations = "4"/>
   <PartyInfo>
   ...
   </PartyInfo>
   <Comment xml:lang="en-gb">any text</Comment> <!--zero or more-->
</CollaborationProtocolAgreement>
```

8.2 CollaborationProtocolAgreement element

The CollaborationProtocolAgreement element is the root element of a CPA. It has a REQUIRED id attribute of type [XML] CDATA that supplies a unique identifier for the document. The value of the id attribute SHALL be assigned by one Party and used by both. It is RECOMMENDED that the value of the id attribute be a URI. The value of the id attribute MAY be used as the value of the CPAId element in the ebXML Message Header[MSSPEC].
NOTE: Each Party MAY associate a local identifier with the id attribute.

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

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

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

8.3 CPAType element

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

```xml
<CPAType>
  <Protocol version = "1.1">PIP3A4</Protocol>
  <Type>RNIF</Type>
</CPAType>
```

The CPAType element is comprised of the following child elements:

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

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

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

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

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

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

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

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

8.5 CPA Lifetime

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

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

8.5.1 Start element

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

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

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

8.5.2 End element

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

```
1999-05-31T13:20:00Z
```
The *End* element SHALL be represented as Coordinated Universal Time (UTC).

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

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

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

NOTE: The runtime 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.

### 8.6 ConversationConstraints element

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

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

The *ConversationConstraints* element has the following attributes:
- an IMPLIED *invocationLimit* attribute,
- an IMPLIED *concurrentConversations* attribute.

#### 8.6.1 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 performace parameter.

#### 8.6.2 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.

8.7 PartyInfo element

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

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

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

8.7.1 ProcessSpecification element

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

8.8 ds:Signature element

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

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

The content of this element and any subelements are defined by the XML Digital Signature specification[XMLDSIG]. The following additional constraints on ds:Signature are imposed:
A CPA MUST be considered invalid if any `ds:Signature` fails core validation as defined by the XML Digital Signature specification.

Whenever a CPA is signed, each `ds:Reference` within a `ProcessSpecification` MUST pass reference validation and each `ds:Signature` MUST pass core validation.

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

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

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

8.8.1 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.

8.8.1.1 Signature Generation

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.

8.8.1.2 `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.

8.8.1.3 `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.
8.8.1.4 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/2000/02/xmldsig#sha1

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

8.8.1.5 ds:Reference element

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

"http://www.w3.org/2000/02/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 for the CPP or CPA document MAY include the id attribute, type ID, by which this ds:Reference element MAY be referenced from a ds:Signature element.

8.8.1.6 ds:Transform element

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

8.8.1.7 ds:Xpath element

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

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

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

8.9 Comment element

The CollaborationProtocolAgreement element MAY contain zero or more Comment elements. See section 7.6 for details of the syntax of the Comment element.
8.10 Composing a CPA from Two CPPs

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

8.10.1 ID Attribute Duplication

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

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

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

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

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

- A separate copy of the Process-Specification document can be created by the CPA composition tool. 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 with information from the corresponding parameters from 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.


9 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.


[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.


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


1977
1979 http://www.w3.org/TR/xmlschema-2/
1980
10 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 [XMLSCHEMA] definition of
the CPP or CPA in Appendix D and available from

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

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

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

Publicly available test suites from vendor neutral organizations such as OASIS and the U.S.A.
National Institute of Science and Technology (NIST) SHOULD be used to verify the
conformance of implementations, applications, and components claiming conformance to this
specification. Open-source reference implementations MAY be available to allow vendors to test
their products for interface compatibility, conformance, and interoperability.
11 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.
**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
e-mail: mwsachs@us.ibm.com

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

Dale W. Moberg
Sterling Commerce
4600 Lakehurst Ct.
Dublin, OH 43016
USA
Phone: 614-793-5015
e-mail: dale_moberg@stercomm.com

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

Copyright © ebXML 2001. All Rights Reserved.

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

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

This document and the information contained herein is provided on an "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
Appendix A  Example of CPP Document (Non-Normative)

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

```xml
<?xml version = "1.0"?>
<!DOCTYPE CollaborationProtocolProfile SYSTEM "cppml%2cv0.23.dtd">
<!--Generated by XML Authority.-->
<CollaborationProtocolProfile id = "id"
 xmlns="http://www.ebxml.org/namespaces/tradePartner"
 xmlns:bpm = "http://www.namespaces/businessProcess"
 xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
 xmlns:xlink = "http://www.w3.org/1999/xlink">
 bpm:BinaryCollaboration | bpm:BusinessTransactionActivity)+ , ds:Signature?)-->  
  <Party partyId = "N01">
    <!--(PartyId+ , PartyDetails , Role+ , Certificate+ , DeliveryChannel+ ,
    Transport+ , DocExchange+)-->  
    <PartyId type = "uriReference">urn:duns.com:duns:1234567890123</PartyId>
    <PartyId type = "uriReference">urn:www.example.com</PartyId>
    <PartyDetails xlink:type="simple" xlink:href="http://example2.com/example.com"/>
    <CollaborationRole roleId="N07" certId="N03">
      <CollaborationProtocol name = "Buy Sell" version = "1.0"
       xlink:type = "locator"
       xlink:href = "http://www.example.com/services/purchasing.xml"/>
      <Role name = "buyer" certId = "N03" xlink:href="http://www.example.com/services/purchasing.xml"/>
    </CollaborationRole>
    <Certificate certId = "N03">
      <!--(ds:KeyInfo)-->  
      <ds:KeyInfo>REFERENCE [XMLDSIG]</ds:KeyInfo>
    </Certificate>
    <DeliveryChannel channelId = "N04"
     transportId = "N05" docExchangeId = "N06">
      <!--(Characteristics , ServiceBinding+)-->  
      <Characteristics nonrepudiationOfOrigin = "true"
       nonrepudiationOfReceipt = "true" secureTransport = "true" confidentiality = "true"
       authenticated = "true" authorized = "true"/>
    </DeliveryChannel>
    <Transport transportId = "N05">
      <!--(Protocol , Endpoint+ , TransportTimeout? ,
      TransportSecurity?)-->
      <Protocol version = "1.1">HTTP</Protocol>
      <Endpoint uri = "http://example.com/servlet/ebxmlhandler" type =
      "request"/>
    </Transport>
  </Party>
</CollaborationProtocolProfile>
```

Collaboration-Protocol Profile and Agreement Specification  Page 58 of 82
Copyright © ebXML 2001. All Rights Reserved.
<TransportSecurity>
  <!--(Protocol , CertificateRef?)-->
  <Protocol version = "3.0">SSL</Protocol>
  <CertificateRef certId = "N03"/>
</TransportSecurity>

<DocExchange docExchangeId = "N06">
  <!--(ebXMLBinding)-->
  <ebXMLBinding version = "0.9">
    <!--(MessageEncoding? , ReliableMessaging , NonRepudiation?,
         DigitalEnvelope? , NamespaceSupported+)-->
    <MessageEncoding version = "base64" packagingType = "need to
discuss">only text</MessageEncoding>
    <ReliableMessaging deliverySemantics = "BestEffort"
        idempotency = "false">
      <!--(Timeout , Retries , RetryInterval)?-->
      <Timeout>30</Timeout>
      <Retries>5</Retries>
      <RetryInterval>60</RetryInterval>
    </ReliableMessaging>
    <NonRepudiation>
      <!--(Protocol , HashFunction , EncryptionAlgorithm ,
           SignatureAlgorithm , CertificateRef)-->
      <Protocol version = "2.0">S/MIME</Protocol>
      <HashFunction>sha1</HashFunction>
      <SignatureAlgorithm>rsa</SignatureAlgorithm>
      <CertificateRef certId = "N03"/>
    </NonRepudiation>
  </ebXMLBinding>
  <DigitalEnvelope>
    <!--(Protocol , EncryptionAlgorithm ,
         CertificateRef)-->
    <Protocol version = "2.0">S/MIME</Protocol>
    <EncryptionAlgorithm>rsa</EncryptionAlgorithm>
    <CertificateRef certId = "N03"/>
  </DigitalEnvelope>
  <NamespaceSupported schemaLocation =
      "http://www.s2ml.org/s2ml.xsd" version =
      "0.7a">http://www.s2ml.org/s2ml</NamespaceSupported>
</DocExchange>
</Party>
<ds:Signature>any combination of text and elements</ds:Signature>
Appendix B  Example of CPA Document (Non-normative)

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

```xml
<?xml version = "1.0"?>
<!DOCTYPE CollaborationProtocolAgreement SYSTEM "cppml%2cv0.23.dtd">
<!--Generated by XML Authority.-->
<CollaborationProtocolAgreement id = "N01"
xmlns="http://www.ebxml.org/namespaces/tradePartner"
xmlns:bpm = "http://www.ebxml.org/namespaces/businessProcess"
xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
xmlns:xlink = "http://www.w3.org/1999/xlink">
  <!--(CPAType? , Status , Start , Duration , ConversationConstraints? , Party+ ,
  (CollaborationProtocol | bpm:BinaryCollaboration | bpm:BusinessTransactionActivity |
  bpm:ProcessSpecification)+ , ds:Signature?)-->
  <CPAType>
    <!--(Protocol , Type)-->
    <Protocol version = "1.1">PIP3A4</Protocol>
    <Type>RNIF</Type>
  </CPAType>
  <Status value = "proposed"/>
  <Start>1988-04-07T18:39:09</Start>
  <Duration>124</Duration>
  <ConversationConstraints invocationLimit = "100" concurrentConversations = "4"/>
  <Party partyId = "N01">
    <!--(PartyId+ , PartyDetails , Role+ , Certificate+ , DeliveryChannel+ ,
    Transport+ , DocExchange+)-->
    <PartyId type = "uriReference">urn:duns.com:duns:1234567890123</PartyId>
    <PartyId type = "uriReference">urn:www.example.com</PartyId>
    <PartyDetails xlink:type="simple" xlink:href="http://example.com/example2.com"/>
    <CollaborationRole roleId="N07" certId="N03">
      <CollaborationProtocol name = "Buy Sell" version = "1.0"
      xlink:type = "locator"
      xlink:href = "http://www.example.com/services/purchasing.xml"/>
      <Role name = "buyer" certId = "N03"
      xlink:href="http://www.example.com/services/purchasing.xml"/>
    </CollaborationRole>
    <Certificate certId = "N03">
      <!--(ds:KeyInfo)-->  
      <ds:KeyInfo>REFERENCE [XMLDSIG]</ds:KeyInfo>
    </Certificate>
    <DeliveryChannel channelId = "N04" transportId = "N05" docExchangeId = "N06">
      <!--(Characteristics , ServiceBinding+)-->
      <ServiceBinding name="MyShopper" channelId="N04"/>
    </DeliveryChannel>
  </Party>
</CollaborationProtocolAgreement>
```
<DeliveryChannel>
   <Transport transportId = "N05">
      <!--(Protocol , Endpoint+ , TransportTimeout? ,
      TransportSecurity?)-->
      <Protocol version = "1.1">HTTP</Protocol>
      <Endpoint uri = "http://example2.com/servlet/ebxmlhandler" type =
      "request"/>
      <TransportSecurity>
         <!--(Protocol , CertificateRef?)-->
         <Protocol version = "3.0">SSL</Protocol>
         <CertificateRef certId = "N03"/>
      </TransportSecurity>
   </Transport>
   <DocExchange docExchangeId = "N06">
      <!--(ebXMLBinding)-->
      <ebXMLBinding version = "0.9">
         <!--(MessageEncoding? , ReliableMessaging , NonRepudiation?,
         DigitalEnvelope? , NamespaceSupported?)-->
         <MessageEncoding version = "base64" packagingType = "need to
discuss">only text</MessageEncoding>
         <ReliableMessaging deliverySemantics = "BestEffort"
         idempotency = "false">
            <!--(Timeout , Retries , RetryInterval)?-->
            <Timeout>30</Timeout>
            <Retries>5</Retries>
            <RetryInterval>60</RetryInterval>
         </ReliableMessaging>
         <NonRepudiation>
            <!--(Protocol , HashFunction , EncryptionAlgorithm,
            SignatureAlgorithm , CertificateRef)-->
            <Protocol version = "2.0">S/MIME</Protocol>
            <HashFunction>sha1</HashFunction>
            <SignatureAlgorithm>rsa</SignatureAlgorithm>
            <CertificateRef certId = "N03"/>
         </NonRepudiation>
         <DigitalEnvelope>
            <!--(Protocol , EncryptionAlgorithm ,
            CertificateRef)-->
            <Protocol version = "2.0">S/MIME</Protocol>
            <EncryptionAlgorithm>rsa</EncryptionAlgorithm>
            <CertificateRef certId = "N03"/>
         </DigitalEnvelope>
         <NamespaceSupported schemaLocation =
         "http://www.s2ml.org/s2ml.xsd" version =
         "0.7a">http://www.s2ml.org/s2ml</NamespaceSupported>
      </ebXMLBinding>
   </DocExchange>
</Party>
<Party partyId = "N01">
   <!--(PartyId+ , Role+ , Certificate+ , DeliveryChannel+ , Transport+,
   DocExchange+)-->
   <PartyId type = "uriReference">urn:duns.com:duns:1234567890123</PartyId>
   <PartyId type = "uriReference">urn:www.example.com</PartyId>
<PartyDetails xlink:type="simple"
    xlink:href="http://example2.com/example.com"/>
<Role certId = "N03" roleId = "N08" name = "seller">
    <!--(ServiceBinding+)-->
    <ServiceBinding collaborationId="N09" channelId="N04"/>
</Role>
</CollaborationRole roleID="N07" certID="N03">
    <CollaborationProtocol name = "Buy Sell" version = "1.0"
        xlink:type = "locator"
        xlink:href = "http://www.example.com/services/purchasing.xml"/>
    <Role name = "buyer"
        xlink:href="http://www.example.com/services/purchasing.xml"/>
    <!--(+)-->  
    <ServiceBinding name="MyShopper" channelId="N04"/>
</CollaborationRole>
</Certificate>
<DeliveryChannel channelId = "N04" transportId = "N05" docExchangeId = "N06">
    <!--(Characteristics , ServiceBinding+)-->  
    <Characteristics nonrepudiationOfOrigin = "true"
        nonrepudiationOfReceipt = "true" secureTransport = "true" confidentiality = "true"
        authenticated = "true" authorized = "true"/>
</DeliveryChannel>
<Transport transportId = "N05">
    <!--(Protocol , Endpoint+ , TransportTimeout? ,
        TransportSecurity?)-->
    <Protocol version = "1.1">HTTP</Protocol>
    <Endpoint uri = "http://example.com/servlet/ebxmlhandler" type = "request"/>
    <TransportSecurity>
        <!--(Protocol , CertificateRef?)-->
        <Protocol version = "3.0">SSL</Protocol>
        <CertificateRef certId = "N03"/>
    </TransportSecurity>
</Transport>
<DocExchange docExchangeId = "N06">
    <!--(ebXMLBinding)-->
    <ebXMLBinding version = "0.9">
        <!--(MessageEncoding? , ReliableMessaging , NonRepudiation?,
            DigitalEnvelope? , NamespaceSupported?)-->
        <MessageEncoding version = "base64" packagingType = "need to discuss">only text</MessageEncoding>
        <ReliableMessaging deliverySemantics = "BestEffort" idempotency = "false">
            <!--(Timeout , Retries , RetryInterval)?-->
            <Timeout>30</Timeout>
            <Retries>5</Retries>
            <RetryInterval>60</RetryInterval>
        </ReliableMessaging>
    </ebXMLBinding>
</DocExchange>
2331          </ReliableMessaging>
2332          <NonRepudiation>
2333          <!--(Protocol , HashFunction , EncryptionAlgorithm ,
2334          SignatureAlgorithm , CertificateRef)-->  
2335          <Protocol version = "2.0">S/MIME</Protocol>
2336          <HashFunction>sha1</HashFunction>
2337          <SignatureAlgorithm>rsa</SignatureAlgorithm>
2338          <CertificateRef certId = "N03"/>
2339          </NonRepudiation>
2340          <DigitalEnvelope>
2341          <!--(Protocol , EncryptionAlgorithm ,
2342          CertificateRef)-->  
2343          <Protocol version = "2.0">S/MIME</Protocol>
2344          <EncryptionAlgorithm>rsa</EncryptionAlgorithm>
2345          <CertificateRef certId = "N03"/>
2346          </DigitalEnvelope>
2347          <NamespaceSupported schemaLocation =
2348          "http://www.s2ml.org/s2ml.xsd" version =
2349          "0.7a">http://www.s2ml.org/s2ml</NamespaceSupported>
2350          </ebXMLBinding>
2351      </DocExchange>
2352      </Party>
2353      <CollaborationProtocol version = "1.0" id = "N07" xlink:type = "locator"
2354      xlink:href = "http://www.example.com/services/purchasing.xml">Buy and Sell
2355      </CollaborationProtocol>
2356      <ds:Signature>any combination of text and elements</ds:Signature>
2357  </CollaborationProtocolAgreement>
Appendix C  DTD Corresponding to Complete CPP/CPA Definition (Normative)

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

<?xml version='1.0' encoding='UTF-8' ?>
<!--Generated by XML Authority-->
<!ELEMENT CollaborationProtocolAgreement (CPAType?, Status, Start, End, ConversationConstraints?, PartyInfo*, ds:Signature+, Comment*)>
<!ATTLIST CollaborationProtocolAgreement id CDATA #IMPLIED>
<!ELEMENT CollaborationProtocolProfile (PartyInfo+, ds:Signature?, Comment*)>
<!ELEMENT ReceivingProtocol (#PCDATA)>
<!ATTLIST ReceivingProtocol version CDATA #IMPLIED
e-dtype NMTOKEN #FIXED 'string'>
<!ELEMENT SendingProtocol (#PCDATA)>
<!ATTLIST SendingProtocol version CDATA #IMPLIED
e-dtype NMTOKEN #FIXED 'string'>
<!ELEMENT Protocol (#PCDATA)>
<!ATTLIST Protocol version CDATA #IMPLIED
e-dtype NMTOKEN #FIXED 'string'>
<!ELEMENT CollaborationRole (ProcessSpecification, Role, CertificateRef?, ServiceBinding+)>
<!ATTLIST CollaborationRole id ID #REQUIRED>
<!ELEMENT PartyInfo (PartyId+, PartyRef, CollaborationRole+, Certificate+, DeliveryChannel+, Transport+, DocExchange+)>
<!ELEMENT PartyId (#PCDATA)>
<!ATTLIST PartyId type CDATA #IMPLIED
e-dtype NMTOKEN #FIXED 'string'>
<!ELEMENT PartyRef EMPTY>
<!ATTLIST PartyRef xlink:type (simple) #FIXED 'simple'
xlink:href CDATA #REQUIRED>
<!ELEMENT DeliveryChannel (Characteristics)>
<!ATTLIST DeliveryChannel channelId ID #REQUIRED
transportId IDREF #REQUIRED
docExchangeId IDREF #REQUIRED>
<!ELEMENT Transport (SendingProtocol+, ReceivingProtocol, Endpoint+, TransportSecurity?)>
<!ATTLIST Transport transportId ID #REQUIRED>
<!ELEMENT Endpoint EMPTY>
<!ATTLIST Endpoint uri CDATA #REQUIRED
type (login | request | response | error | allPurpose) 'allPurpose'
a-dtype NMTOKENS 'uri' uri ' '>
<!ELEMENT Retries (#PCDATA)>
<!ATTLIST Retries e-dtype NMTOKEN #FIXED 'string'>
<!ELEMENT RetryInterval (#PCDATA)>
<!ATTLIST RetryInterval e-dtype NMTOKEN #FIXED 'string'>
<!ELEMENT TransportSecurity (Protocol, CertificateRef)>
<!ELEMENT Certificate (ds:KeyInfo)>
<!ATTLIST Certificate certId ID #REQUIRED>
<!ELEMENT DocExchange (ebXMLBinding)>
<!ATTLIST DocExchange docExchangeId ID #IMPLIED>
<!ELEMENT ReliableMessaging (Retries, RetryInterval)>
<!ATTLIST ReliableMessaging deliverySemantics (OnceAndOnlyOnce | BestEffort) #REQUIRED
idempotency CDATA #REQUIRED
persistDuration CDATA #REQUIRED>a-dtype NMTOKENS 'idempotency boolean'
timeDuration' 'timeDuration' ' '>
<!ELEMENT NonRepudiation (Protocol, HashFunction, SignatureAlgorithm, CertificateRef)>
<!ELEMENT HashFunction (#PCDATA)>
<!ATTLIST HashFunction e-dtype NMTOKEN #FIXED 'string'>
<!ELEMENT EncryptionAlgorithm (#PCDATA)>
<!ATTLIST EncryptionAlgorithm e-dtype NMTOKEN #FIXED 'string'>
<!ELEMENT SignatureAlgorithm (#PCDATA)>
<!ATTLIST SignatureAlgorithm e-dtype NMTOKEN #FIXED 'string'>
<!ELEMENT DigitalEnvelope (Protocol, EncryptionAlgorithm, CertificateRef)>
<!ELEMENT ProcessSpecification (ds:Reference)>
<!ATTLIST ProcessSpecification
  name ID #REQUIRED
  version CDATA #REQUIRED>
  xlink:type (simple ) #FIXED 'simple'
  xlink:href CDATA #REQUIRED>
<!ELEMENT ds:Reference (ds:DigestMethod, ds:DigestValue)>
<!ATTLIST ds:Reference
ds:URI CDATA #REQUIRED>
type CDATA #IMPLIED >
<!ELEMENT ds:DigestMethod ( #PCDATA | ds:HMACOutputLength )>
<!ATTLIST ds:DigestMethod
ds:Algorithm CDATA #REQUIRED>
<!ELEMENT ds:HMACOutputLength (#PCDATA)>
<!ELEMENT ds:DigestValue (#PCDATA)>
<!ELEMENT CertificateRef EMPTY>
<!ATTLIST CertificateRef certId IDREF #IMPLIED>
ed-dtype NMTOKEN #FIXED 'string' >
<!ELEMENT MessageEncoding (#PCDATA)>
<!ATTLIST MessageEncoding
  version CDATA #REQUIRED>
  packagingType CDATA #IMPLIED>
ed-dtype NMTOKEN #FIXED 'string' >
<!ELEMENT ebXMLBinding (MessageEncoding?, ReliableMessaging?, NonRepudiation?,
  DigitalEnvelope?, NamespaceSupported+)>
<!ATTLIST ebXMLBinding
  version CDATA #REQUIRED >
<!ELEMENT ds:KeyInfo EMPTY>
<!ELEMENT ds:Signature EMPTY>
<!ELEMENT NamespaceSupported (#PCDATA)>
<!ATTLIST NamespaceSupported
  schemaLocation CDATA #IMPLIED>
  version CDATA #REQUIRED>
ed-dtype NMTOKEN #FIXED 'uri' >
a-dtype NMTOKENS 'schemaLocation uri' >
<!ELEMENT EMPTY>
<!ATTLIST Characteristics
  nonrepudiationOfOrigin CDATA #IMPLIED>
  nonrepudiationOfReceipt CDATA #IMPLIED>
  secureTransport CDATA #IMPLIED>
  confidentiality CDATA #IMPLIED>
  authenticated CDATA #IMPLIED>
  authorized CDATA #IMPLIED>
a-dtype NMTOKENS 'nonrepudiationOfOrigin boolean
  nonrepudiationOfReceipt boolean
  secureTransport boolean
  confidentiality boolean
  authenticated boolean
  authorized boolean' >
<!ELEMENT ServiceBinding (Packaging+, Override*)>
<!ATTLIST ServiceBinding
  channelId ID REF #REQUIRED>
  name CDATA #IMPLIED >
<!ELEMENT CPAType (Protocol , Type)>
<!ELEMENT Status EMPTY>
<!ATTLIST Status
  value (signed | agreed | proposed ) #REQUIRED >
<!ELEMENT Start (#PCDATA)>
<!ELEMENT Start e-dtype NMTOKEN #FIXED 'timeInstant' >
<!ELEMENT End (#PCDATA)>
<!ELEMENT End e-dtype NMTOKEN #FIXED 'timeInstant' >
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Type e-dtype NMTOKEN #FIXED 'string' >
<!ELEMENT ConversationConstraints EMPTY>
<!ATTLIST ConversationConstraints
  invocationLimit CDATA #IMPLIED>
  concurrentConversations CDATA #IMPLIED>
a-dtype NMTOKENS 'invocationLimit 1
  concurrentConversations 14' >
<!ELEMENT Override EMPTY>
<!ATTLIST Override
  action CDATA #REQUIRED>
  channelId ID #REQUIRED>
  xlink:href CDATA #IMPLIED>
  xlink:type (simple ) #FIXED 'simple' >
<!ELEMENT Role (#PCDATA)>
<!ATTLIST Role
  name CDATA #IMPLIED>
xlink:href CDATA #IMPLIED>
xlink:type (simple ) #FIXED 'simple' >
<!ELEMENT Packaging (ProcessingCapabilities , SimplePart+ , CompositeList?)++>
<!ELEMENT Comment ANY>
<!ELEMENT Composite (Constituent+)>  
<!ATTLIST Composite mimetype CDATA #REQUIRED
id ID #REQUIRED
mimeparameters CDATA #IMPLIED >
<!ELEMENT Constituent EMPTY>
<!ATTLIST Constituent idref IDREF #REQUIRED >
<!ELEMENT Encapsulation (Constituent)>  
<!ATTLIST Encapsulation mimetype CDATA #REQUIRED
id ID #REQUIRED
mimeparameters CDATA #IMPLIED >
<!ELEMENT CompositeList (Encapsulation | Composite)+>
<!ELEMENT XMLMetaDataInformation EMPTY>
<!ATTLIST XMLMetaDataInformation URI CDATA #IMPLIED
MetaDataDescriptionType (dtd | xsd ) #REQUIRED >
<!ELEMENT MimeHeader EMPTY>
<!ATTLIST MimeHeader HeaderName CDATA #REQUIRED >
<!ELEMENT MimeParameter EMPTY>
<!ATTLIST MimeParameter parameterAttribute CDATA #REQUIRED
parameterValue CDATA #IMPLIED >
<!ELEMENT SimplePart EMPTY>
<!ATTLIST SimplePart id ID #REQUIRED
mimetype CDATA #REQUIRED >
<!ELEMENT ProcessingCapabilities EMPTY>
<!ATTLIST ProcessingCapabilities parse CDATA #REQUIRED
generate CDATA #REQUIRED >
<!ELEMENT ds:Reference EMPTY>
Appendix D  XML Schema Document Corresponding to Complete CPA Definition (Normative)

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

```xml
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2000/10/XMLSchema-->
<xsd:schema xmlns:xlink = "http://www.w3.org/1999/xlink"
    xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
    xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema">
    <xsd:import namespace = "http://www.w3.org/1999/xlink" schemaLocation = "http://www.w3.org/1999/xlink"/>
    <xsd:element name = "CollaborationProtocolAgreement">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref = "CPAType" minOccurs = "0"/>
                <xsd:element ref = "Status"/>
                <xsd:element ref = "Start"/>
                <xsd:element ref = "Duration"/>
                <xsd:element ref = "ConversationConstraints" minOccurs = "0"/>
                <xsd:element ref = "PartyInfo" minOccurs = "0" maxOccurs = "unbounded"/>
                <xsd:element ref = "ds:Signature" minOccurs = "0"/>
                <xsd:element ref = "Comment" minOccurs = "0" maxOccurs = "unbounded"/>
            </xsd:sequence>
            <xsd:attribute name = "id" type = "xsd:string"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "CollaborationProtocolProfile">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref = "PartyInfo" maxOccurs = "unbounded"/>
                <xsd:element ref = "ds:Signature" minOccurs = "0"/>
                <xsd:element ref = "Comment" minOccurs = "0" maxOccurs = "unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "ReceivingProtocol">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:extension base = "xsd:string">
                    <xsd:attribute name = "version" type = "xsd:string"/>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "SendingProtocol">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:extension base = "xsd:string">
                    <xsd:attribute name = "version" type = "xsd:string"/>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "Protocol" type = "xsd:string"/>
    <xsd:element name = "CollaborationRole">
        <xsd:complexType>
            <xsd:sequence>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```
<xsd:element ref="ProcessSpecification"/>
<xsd:element ref="Role"/>
<xsd:element ref="CertificateRef" minOccurs="0"/>
<xsd:element ref="ServiceBinding" maxOccurs="unbounded"/>
</xsd:sequence>
<xsd:attribute name="roleId" use="required" type="xsd:ID"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="PartyInfo">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="PartyId" maxOccurs="unbounded"/>
<xsd:element ref="PartyRef"/>
<xsd:element ref="CollaborationRole" maxOccurs="unbounded"/>
<xsd:element ref="Certificate" maxOccurs="unbounded"/>
<xsd:element ref="DeliveryChannel" maxOccurs="unbounded"/>
<xsd:element ref="Transport" maxOccurs="unbounded"/>
<xsd:element ref="DocExchange" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="PartyId">
<xsd:complexType>
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="type" type="xsd:string"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
<xsd:element name="PartyRef">
<xsd:complexType>
<xsd:sequence/>
<xsd:attribute name="xlink:type" use="fixed" value="simple">
<xsd:simpleType>
<xsd:restriction base="xsd:NMTOKEN">
<xsd:enumeration value="simple"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:complexType>
</xsd:element>
<xsd:element name="DeliveryChannel">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="Characteristics"/>
</xsd:sequence>
<xsd:attribute name="channelId" use="required" type="xsd:ID"/>
<xsd:attribute name="transportId" use="required" type="xsd:IDREF"/>
<xsd:attribute name="docExchangeId" type="xsd:IDREF"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="Transport">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="SendingProtocol"/>
<xsd:element ref="ReceivingProtocol"/>
<xsd:element ref="Endpoint" maxOccurs="unbounded"/>
<xsd:element ref="TransportSecurity" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="transportId" use="required" type="xsd:ID"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="Endpoint">
<xsd:complexType>
<xsd:sequence/>
<xsd:attribute name="uri" use="required" type="xsd:uriReference"/>
<xsd:attribute name="type" use="default" value="allPurpose"/>
<xsd:restriction base="xsd:NMTOKEN"/>
<xsd:enumeration value = "login"/>
<xsd:enumeration value = "request"/>
<xsd:enumeration value = "response"/>
<xsd:enumeration value = "error"/>
<xsd:enumeration value = "allPurpose"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
<xsd:sequence/>
  <xsd:element ref = "Packaging" maxOccurs = "unbounded"/>
  <xsd:element ref = "Override" minOccurs = "0"/>
</xsd:sequence>
  <xsd:attribute name = "channelId" use = "required" type = "xsd:ID"/>
  <xsd:attribute name = "name" type = "xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>  

<xsd:element name = "Status">
  <xsd:complexType>
    <xsd:sequence/>
    <xsd:attribute name = "value" use = "required">
      <xsd:simpleType>
        <xsd:restriction base = "xsd:NMTOKEN">  
          <xsd:enumeration value = "signed"/>
          <xsd:enumeration value = "proposed"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
<xsd:element name = "Role">  
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base = "xsd:string">  
        <xsd:attribute name = "name" type = "xsd:string"/>
        <xsd:attribute name = "xlink:href" type = "xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
<xs:complexType name="SecurityRisks">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="ProcessingCapabilities"/>
    <xs:element ref="SimplePart" maxOccurs="unbounded"/>
    <xs:element ref="CompositeList" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Composite">
  <xs:sequence>
    <xs:element ref="Constituent" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="idref" use="required" type="xsd:string"/>
</xs:complexType>

<xs:complexType name="CompositeList">
  <xs:choice maxOccurs="unbounded">
    <xs:element ref="Encapsulation"/>
    <xs:element ref="Composite"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="XMLMetaDataInformation">
  <xs:sequence>
    <xs:element ref="Encapsulation"/>
    <xs:element ref="Composite"/>
  </xs:sequence>
</xs:complexType>
<xsd:complexType>
  <xsd:sequence/>
  <xsd:attribute name="parameterAttribute" use="required" type="xsd:string"/>
  <xsd:attribute name="parameterValue" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
Appendix E  Formats of Information in the CPP and CPA (Normative)

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

Formats of Character Strings

Protocol and Version Elements

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

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

Alphanumeric Strings

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

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

Numeric Strings

A numeric string is a signed or unsigned decimal integer in the range imposed by a 32-bit binary number, i.e. -2,147,483,648 to +2,417,483,647. Negative numbers MAY or MAY not be permitted in particular elements.
Appendix F  Composing a CPA from Two CPPs (Non-Normative)

Overview and Limitations

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

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

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

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

Variability in Inputs

User preferences provide one source of variability in the inputs to the CPA formation process.

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

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

Different Approaches

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

Variable Output “Satisficing” Policies

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

However, there can be interoperable configurations agreed to in a CPA in which not all aspects of a business collaboration match. Gaps MAY exist in packaging, security, signaling, reliable messaging and other areas and yet the systems can still transport the business data, and special means can be employed to handle the exceptions. In such situations, a CPA MAY reflect configured policies or expressly solicited user permission to ignore some shortcomings in configurations. A system might not be capable of responding in a business collaboration so as to support a recommended ability to supply nonrepudiation of receipt, but might still be acceptable for business reasons. A system might not be able to handle all the processing required to support "multipart/related" processing with a type value of "application/vnd.eb+xml," 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 Process, 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-Process (BP) specifications (as defined in the Process-Specification document) and those services and protocols that are used to implement these BP 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 BP to its implementing services and protocols can be thought of as arriving at interoperable, application-to-application integration. CPAs MAY fall short of this goal and still be useful and acceptable to the collaborating Parties. Certainly, if no matching data-transport capabilities can be discovered, a CPA would not provide much in the way of interoperable business-to-business integration. Likewise, partial CPAs will leave significant system work to be done before a completely satisfactory application-to-application integration is realized. Even so, partial integration MAY be sufficient to allow collaboration, and to enjoy payoffs from increased levels of automation.

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

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

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

1. an intended collaborator is known and the collaborator’s CPP has been retrieved,
2. the Business Process between us and our intended collaborator has been selected,
3. the specific role that our software agent is to play in the BP 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 Process 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.

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 BP. 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 BP that we have chosen. For simple binary collaboration cases, it is typically sufficient to find that our intended collaborator’s CollaborationRole set contains ProcessSpecification elements that we intend to implement and where the role is not identical to our role. For more general collaborations, we would need to know the list of roles available within the process, and keep track that for each of the
collaborators, the roles chosen instantiate those that have been specified within the *Process-Specification* document. Collaborations involving more than two roles are not discussed further.

### Matching Transport

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

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

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

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

and our seller has a Transport entry:

```xml
<Transport transportId = "sellid001">
  <SendingProtocol>HTTP</SendingProtocol>
  <ReceivingProtocol>
    HTTP
  </ReceivingProtocol>
  <Endpoint uri = "https://www.sellername.com/pos_here"
    type = "allPurpose"/>
  <TransportSecurity>
    <Protocol version = "1.0">TLS</Protocol>
    <CertificateRef certId = "certid002">Sellername</CertificateRef>
  </TransportSecurity>
</Transport>
```
A transport match for requests involves finding the initiator role or buyer has a **SendingProtocol** that matches one of our **ReceivingProtocols**. 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 **ReceivingProtocols**. So in the above example, “HTTP” again provides a match. When such matches exist, we then have discovered an interoperable solution at the transport level. If not, no **CPA** will be available, and a high-priority gap has been identified that will need to be remedied by whatever exception handling procedures are in place.

### Matching Transport Security

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

### Matching Document Packaging

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

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

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

1. the buyer asserting what packaging it can generate for its purchase order, and what packaging it can parse for its purchase order response **Messages**.
2. the seller asserting what packaging it can generate for its purchase order responses and what packaging it can parse for received purchase orders.

Matching by structural comparison would then involve comparing the packaging details of the purchase orders generated by the seller with the purchase orders parsable by the buyer. The comparison would seek to establish that the MIME types of the **SimpleParts** 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:

```xml
<ProcessingCapabilities parse = "true" generate = "true"/>
<SimplePart id = "P1" mimetype = "application/vnd.eb+xml"/>
<SimplePart id = "P2" mimetype = "application/po+xml"/>
<CompositeList>
  <Composite mimetype = "multipart/related" id = "P3"
    mimeparameters = "type=application/eb+xml">
    <Constituent idref = "P1"/>
    <Constituent idref = "P2"/>
  </Composite>
</CompositeList>
</Packaging>
```

However, it is to be expected that over time it might become possible to only 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.

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

## 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[MSSPEC] packaging is present. However, there are other sources that should be checked to confirm this match. The DeliveryChannel element has a subtree under the DocExchange element that, for the ebXMLBinding element, has a NameSpacesSupported element. XMLDsig capability should be found there. Likewise, a detailed check on this match should examine the information under the NonRepudiation element to check for compatibility in hash functions and algorithms.

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

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

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

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

**Other Considerations**

Handling Preferences among multiple matching capabilities involves

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