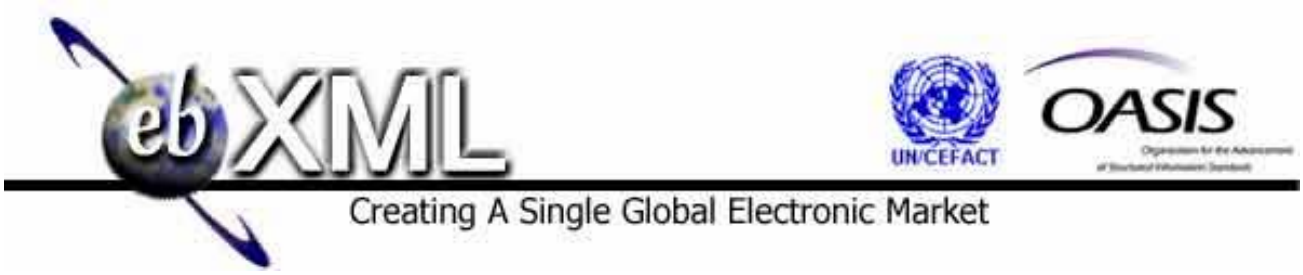

Automated Negotiation of Collaboration-Protocol Agreements Specification Version 0.01

OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee, Automated Negotiation Subteam

Date TBD

Status of this Document

This document specifies an ebXML SPECIFICATION for the eBusiness community.

Distribution of this document is unlimited.

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

This version:
URL TBD

Errata for this version:
URL TBD

***Previous version*:**
URL TBD

1 Automated Negotiation Subteam Members

2 Table of Contents

3 Introduction

3.1 Summary of Contents of Document

3.2 Definition and Scope of this Specification

SEE SECTION 7.6 OF EBCPP FOR IDEAS

3.3 Document Conventions

Terms in *Italics* are defined in Appendix G or in the glossary of the CPPA specification[ebCPP]. Terms listed in ***Bold Italics*** represent the element and/or attribute content of the XML *CPP, CPA*, or related definitions.

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

References to external documents are represented with BLOCK text enclosed in brackets, e.g. [RFC2396]. The references are listed in Section 14.

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, CPA, or NDDs* and not in others. Some of these elements define parameters or operating modes and SHOULD be implemented by all vendors. It might be appropriate to implement elective elements that represent major run-time functions, such as various alternative communication protocols or security functions, by means of plug-ins so that a given *Party* MAY acquire only the needed functions rather than having to install all of them.

By convention, values of [XML] attributes are generally enclosed in quotation marks; however those quotation marks are not part of the values themselves.

3.4 Versioning of the Specification, Schema, and Related Documents

3.5 Definitions

Technical terms related to the subject of this specification are defined in Appendix G. Technical terms related to Collaboration Protocol Profiles and Agreements and to the overall vocabulary of ebXML are defined in {ebCPP}.

3.6 Audience

One target audience for this specification is implementers of ebXML services and other

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

3.7 Assumptions

It is expected that the reader has an understanding of XML and is familiar with the ebXML CPPA specification[ebCPP].

3.8 Related Documents

Related documents include ebXML specifications on the following topics:

- ebXML Collaboration Protocol Profile and Agreement Specification[ebCPP]
- ebXML Business Process Specification Schema[ebBPSS]
- ebXML Message Service Specification[ebMS]

See Section 14 for the complete list of references.

4 Design Objectives

5 System Overview

5.1 What this Specification Does

Figure 1 is a high-level view of the negotiation process. Following are some details of the negotiation process illustrated in Figure 1.

- Initial inputs:
  - CPPs and the associated NDDs of two prospective partners or a CPA template and NDD that one partner provides to a prospective partner.
    - For the case of the CPA template and NDD, the CPA template might be generated by one of the parties, might be a copy of a CPA used by someone else that is almost exactly what is needed, or might be supplied by a third-party negotiation service.
  - Proposed Process Specification document (BPSS instance document)
    - The partners can negotiate about which BPSS instance document to use based on the name of the BPSS instance document (i.e. syntactic negotiation) but not over the details within a given BPSS instance document (semantic negotiation).
- The negotiation process starts with the two prospective partners exchanging NDDs or (for third-party negotiation) each prospective partner providing its NDD and CPP to the negotiation service. Alternatively, once party may provide a CPA template to the other party.
  - Which party can initially propose a CPA template?
    - The party who initiates contact with another party?
    - The party who is contacted by another party?
    - Either party?

    The team agreed that either party could propose a CPA template. However there is a potential race condition in which each proposes a CPA template. If "either party" is accepted as the answer, the negotiation specification will have to include a protocol for that resolves the race condition.
- Composition tool builds initial version of CPA from the two CPPs.
- If the initial CPA is complete (syntactically valid, usable, and agreed to by both parties), does it go into effect immediately or is human review and approval required? The former would be chosen if dynamic eCommerce is desired. The choice could be specified in the NDD. NCPAs could be provided for each alternative.
  - See "Responses to CPA Proposal"
- Negotiation of items requiring human input
  - May need to indicate in the NDD, what needs human input.
- Offer, counter-offer information is in business messages exchanged using negotiation business transactions defined in the NCPA.
- End of negotiation:
  - A successful result is a CPA that is ready to use, possibly subject to human approval.
  - An unsuccessful result means that agreement was not reached on some items in the CPA. Possibly, further human interaction could resolve the disagreement.

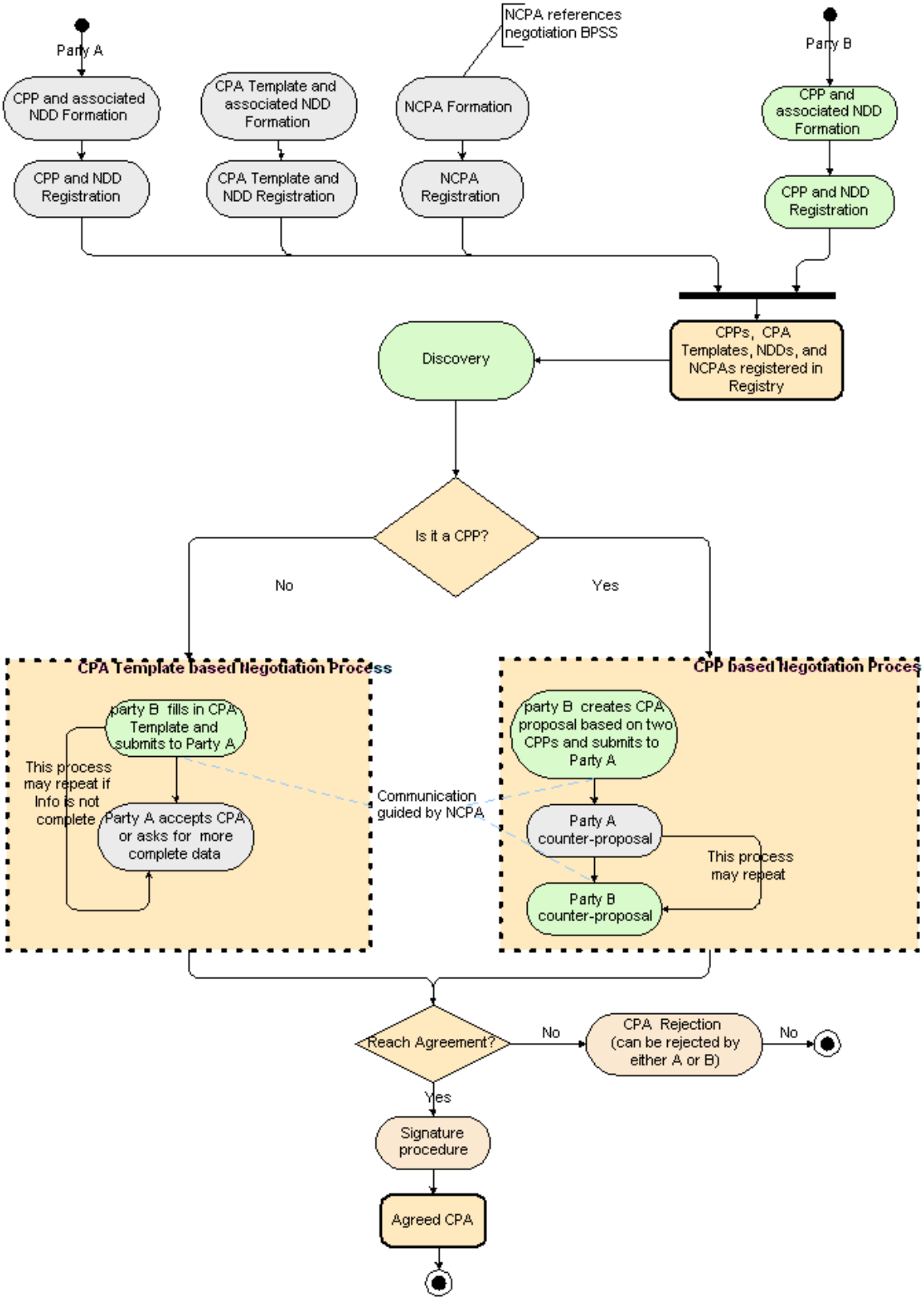

Figure 1, Negotiation Process

5.2 CPP Formation and Editing

These are pre-discovery steps that are out of scope for the negotiation specification, they are included here in the interest of completeness.

- CPP Template
  - Supplied with software installation (configured options)
  - Edited to reflect preferences
- NDD formation.
  - Although NDD formation is out of scope, the NDD schema is a key component of the specification.
- Tool for custom CPP formation
- Tool for NDD formation
- Service(s) for supplying CPPs or CPA templates
  - UDDI advertised, SOAP, ebXML, simple HTTP GET, and so on.
- ebXML registry submission (publication)
- Can a party publish both a CPP and a CPA template?
  In principle, a party should be able to publish both a CPP and a CPA template. However, this would lead to a problem that a given prospective trading partner might find either one. If a party intends that some prospective trading partners negotiate with a CPP while other are expected to accept a CPA template, then the party should probably publish only the CPP and decide whether to send a CPA template based on its knowledge of who the prospective trading partner is.

5.3 Discovery of CPPs

The discovery process is out of scope for the negotiation specification; it is included here in the interest of completeness.

- The minimum requirement is to be able to perform an HTTP GET of a CPP from a URL obtained by means outside the scope of this specification.
- UDDI ebXML Registry bootstrap.
- Search and retrieval in ebXML registry or similar registry.
- Well-known address as done in eCo framework.
- Should/can a registry have any further role(s), perhaps as value-added services?
  - Notification of CPP expirations?
  - Accept filled-out CPA templates?

5.4 Negotiation through an Intermediary

Negotiation through an intermediary is out of scope for this version of the specification if it requires a 3-party negotiation CPA. It may be possible to use an intermediary if the interactions between each Party and the intermediary are defined by a separate Negotiation CPA and a suitable BPSS instance document.

6 CPP and CPA Template Content

6.1 Negotiability

This section discusses how to express items that are negotiable in the *CPP* and *CPA* template prior to negotiation. The rules ensure that the negotiable *CPP* or *CPA* template can be validated by an XML parser while not appearing to constrain negotiability.

In general, since the negotiability details are provided in the *NDD*, it should be acceptable to include any arbitrary value or choice for a negotiable item in the pre-negotiation *CPP* or *CPA* template. In other words, the *NDD* overrides what is in the pre-negotiation *CPP* or *CPA* template for all negotiable items.

- Numerical values: Any valid value can be stated for a negotiable item in the pre-negotiation *CPP* or *CPA*.
- Cardinality: All acceptable choices that are to be negotiated must appear in the pre-negotiation *CPP* or *CPA* template.

THE ABOVE MATERIAL WILL BE EXTENDED TO ENCOMPASS ALL NEGOTIABILITY PATTERNS THAT ARE IDENTIFIED.

The following items in the CPP must be listed in preference order.

- ***PartyId*** elements under the same ***PartyInfo*** element.
- ***CanSend*** and ***CanReceive*** elements under the ***ServiceBinding*** element ***(NEED TO VERIFY THIS)***
- ***AccessAuthentication*** elements under the same ***TransportSender*** element
- ***EncryptionAlgorithm*** elements under the same ***TransportClientSecurity*** or ***TransportServerSecurity*** element.
- ***TransportProtocol*** elements under the same ***Transport*** element
- ***AnchorCertificate*** elements under the same ***Certificate*** element

7 CPA composition

The rules in this section apply to both composition of a CPA from two CPPs and (where appropriate) to the contents of a CPA template.

Appendix H contains a detailed discussion of CPA composition. ***Appendix H WILL BE DRAWN ON HEAVILY OR MOVED INTO THE NORMATIVE CHAPTERS OF THIS SPECIFICATION AND REORGANIZED AS NEEDED TO INDICATE WHAT IS NORMATIVE AND WHAT IS NON-NORMATIVE.***

- One party (or the intermediary) creates the initial draft of the CPA by CPA composition from the two CPPs.
- There is a possibility that both prospective trading partners might compose and send a draft CPA to each other. This race condition will have to be dealt with.
- A draft of a CPA composed from two CPPs is somewhat similar to a CPA template in that it is probably incomplete. However, the CPA template, by definition offers few choices to the other party whereas a draft composed form two CPPs may include a large number of negotiable items.
- It is likely that the process from the point that a CPA draft is composed from two CPPs will be very similar to the process for a CPA template except for the number of negotiable items in the two cases.
- The process of composing the CPA draft from two CPPs will often narrow down the amount of negotiation relative to the negotiation possibilities expressed in the NDDs. Many items that are potentially negotiable in the CPPs will be no longer negotiable after the CPA is composed. For example, there may be only one transport protocol that is common to the two parties. The negotiation process must evaluate the NDDs againt the composed CPA and not attempt to negotiate items for which the composition process fixed the result.
- It was noted during the Jan. 30, 2002 face to face meeting that it might not be necessary to create an XML document containing the composed CPA draft. The negotiation process could maintain the intersection of the two CPPs in an internal form and not complete the actual CPA document until the negotiation process has converged. However, some people preferred to start the negotiation by creating an initial draft CPA and providing it to both parties.
-

THIS SECTION WILL INCLUDE A DISCUSSION OF ERROR CONDITIONS THAT CAN BE DETECTED DURING THE CPA COMPOSITION PROCESS.

8 CPA Template

- A CPA template can be placed in a registry in place of a CPP when a party wishes to dictate all terms and conditions of the final CPA. The prospective trading partner would only have to fill in a minimal set of information, such as an endpoint address and a certificate to be ready to do business.
- With a CPA template, the accompanying NDD would be very simple but would indicate which elements and attributes need to be completed by the prospective trading partner. Having the NDD probably facilitates identifying the items to be negotiated or filled in compared with having to parse the CPA template to find those items.
- For a CPA template, it is likely that a party would not have multiple NDDs for the same template. Therefore, it may be appropriate to tie the NDD to the CPA template in the registry. Possibilities include:
  - Embedding the CPA template in the NDD
  - Importing the CPA template namespace and the template itself into the NDD.
- If party A discovers party B's CPP in a registry, Party B does not have party A's CPP. Party A could compose a CPA template using Party B's CPP, and present that template to Party B. This would save the extra steps for Party A to send its CPP to Party A and the exchange of NDDs. Note, however, that in this process, Party A is dominant. This might have a very different outcome than would result from a peer negotiation between Party A and Party B using two CPPs and two NDDs.

9 Negotiation CPA (NCPA)

The purpose of this chapter is to:

- Explain how to construct the Negotiation CPA such that it does not have to be negotiated;
- Explain the negotiation aspects of the NCPA. Principally, these aspects are the elements that define the interface between a CPA and the BPSS instance, i.e. the CollaborationRole, ***ProcessSpecification***, and ***Role*** elements.

The NCPA defines the interactions between two Parties who are negotiating the contents of a CPA. It identifies the BPSS instance document that defines the negotiation choreography. An example of an NCPA is in Appendix C.

The following are minimalist requirements that help avoid the need to negotiate the negotiation CPA.

- Use HTTP POST to send a proposed CPA to a URL.
- Synchronous response to a proposal. This avoids the need for the responder to know the URL for a response.
- *Messaging* using basic SOAP or W3C XML Protocol (when available). In this context, "basic" means that values or choices that have to be negotiated will either be omitted or will be given fixed values by this specification.
- ***THIS LIST WILL BE EXPANDED AS NEEDED.***

10 Pre-Conditions for Negotiation

This section discusses conditions that must be met before negotiation. If these conditions are not met, a successful outcome is unlikely. The discussions relate to CPPs or a CPA template as appropriate

The two partners must agree on what negotiation process to follow, i.e. what NCPA to use for negotiation. (The NCPA identifies the negotiation BPSS instance to be used.)

There must be a minimum level of matching (i.e. compatibility) between two CPPs.

- At least one transport protocol in common.
- There must be a minimum level of compatibility between at least one ***DocumentExchange*** element in each CPP ***(DETAILS TO BE DETERMINED)***.
- There must be at least one certificate authority (CA) in common between two CPPs. The CAs are identified in the certificates referred to by ***ArchorCertificateReference*** elements.
- ***THIS LIST WILL BE EXPANDED.***

See Section 6 for related information.

11 Negotiation Descriptor Document

11.1 Use of NDD

- An NDD could be placed in a registry along with the CPP. NDD and CPP would have to be connected by registry metadata. We do not want to include a link to the NDD in the CPP since there may be many NDDs, with different negotiation details, associated with one CPP.
- We believe that the recommended procedure should be not to include an NDD in the registry. Instead, one a party is discovered by a prospective trading partner, the NDDs should be exchanged in the opening step of the negotiation. This permits a party to send an NDD that it considers appropriate for the particular prospective trading partner.
- It should not be necessary to exchange revised NDDs after each negotiation step. The negotiation process can maintain the detailed state and compose an acceptable CPA at the end without repeated exchanges of NDDs. Appropriate state information can be exchanged in the negotiation messages.
- It might be desirable to exchange NDDs and/or a partially completed CPA occasionally as a checkpoint.
- It is suggested that in the first version of the specification, NDDs be exchanged only during initialization of the negotiation process. Based on initial experience, intermediate exchanges of NDDs could be added later.

11.2 Contents of NDD

The NDD must reference both the draft CPA (CPA template) and the CPPA Schema.

It is highly desirable to define the NDD in a sufficiently abstract fashion to be able to apply it to any kind of XML agreement. Doing so would mean that it would not be necessary to design a new NDD schema for each kind of document to be negotiated.

The NDD could consist of a variable length (cardinality 1 or more) set of [XPATH] statements, each of which refers to a negotiable element or attribute.

Under each such XPATH statement, the negotiability of the element or attribute would be defined by child elements. These child elements have to represent the negotiability characteristics of the element or attribute identified by the XPATH statement.Examples are:

- Cardinality (range of permitted cardinalities)
- For a numeric value, minimum, maximum, and negotiation step size
- For choices, XPATH statements, ID attribute values, qnames, element values, etc. which identify the specific choices within the document being negotiated. Examples in the CPA are certificates, delivery channels, transport protocols, and signature algorithms.

  NOTE: It is likely that an NDD expressed in this abstract manner would not be very readable. This is an opportunity for tool vendors to produce NDD composition tools. Such a tool would have a GUI that would tailor the view of the NDD to the specific kind of document to be negotiated. The tool would reference the schema of the document being

negotiated along with the NDD being constructed, which should supply it with sufficient information to make the views understandable by someone who is composing an NDD. This would enable that person to communicate with the tool in terms of the specifics of the document to be negotiated. The tool could then construct the NDD instance document in accord with the NDD schema.

12 Negotiation Protocol

12.1 BPSS Instance for Automated Negotiation

THIS SECTION IS AN EXPLANATION OF THE BPSS DEFINITION FOR AUTOMATED NEGOTIATION. ONE OR MORE FIGURES WILL BE USEFUL. THE FIGURES MIGHT BE SIMILAR TO THOSE IN BRIAN HAYES' "COLLABORATION PROTOCOL AGREEMENT SIMPLE NEGOTIATION BUSINESS PROCESS MODEL".

The choreography of the negotiation protocol MAY be defined by an instance document of the ebXML Business Process Specification Schema[ebBPSS]. The BPSS instance document for automated negotiation is in Appendix D.

A counter offer should be a requesting document in a new Business Transaction, not a response to an offer. To issue a counter offer, the recipient of an offer SHALL reply "counter-offer pending" and then issue the counter offer as a new Business Transaction. This avoids a race condition with respect to which Party sends the next message. It also avoids any need to for the two Parties to switch roles.

12.2 Offer and Counter Offer

12.2.1 Submission of Proposed CPA to One or Both Parties

- Protocol(s) for submission and CPAId conventions if ebXML MSG used.
- Lightweight PUT or POST of proposed CPA (to permit use with non-ebXML MSG transport MSHes.
- Response-to URLs?

12.2.2 Responses to CPA Proposal

This is an example of what might be specified.

- Accept with no changes
  - Accept
  - Accept and deploy (dynamic eCommerce)
- Accept with value changes only.
- Counterproposal:
  - Deleted elements,
  - Added elements
  - Re-ordered elements using an [XPATH]-based list of changes with status of required or preferred.
- Rejection: with reason(s) for rejection

12.2.3 Counterproposal Acceptance

12.2.4 Counterproposal Counter

12.2.5 Offer-Counter Offer Algorithm

- The offer-counter offer procedure must be designed to avoid infinite loops. The algorithm must converge rapidly to either success or failure. Some kind of forward progress indicator must be included.
- The convergence procedure must distinguish between an offer-counter offer loop over the same negotiable item and successive negotiations over different items.
- The NDD focuses the offers and counter offers on what is acceptable. Any offer or counter offer that is outside the limits defined in either NDD must be rejected.
- The algorithm generally should avoid backtracking over items for which the negotiation has converged. However there may be cases in which multiple negotiatable items interact. For such a case, backtracking might a necessary part of of converging the negotiation of the set of interacting items.

12.2.6 Counterproposal Rejection of Proposal or Counterproposal

12.3 Reasons for Rejection during Negotiation

The process of composing the CPA from CPPs will detect many problems before the negotiation process begins. Examples are mismatched Process Specification document and mismatched delivery channel requirements. These are elaborated in Section 7.

The rejection message includes reason, contact name, phone, and/or URL for further information.

Following are some reasons for rejection:

- CPP/CPA contents. Examples:
  - base CPP deprecated
  - signature on CPP failed validation
  - Signature on agreed CPA failed validation
    - CPA is not signed until it is agreed to.
  - proposed security too weak
  - proposed *Packaging* not supported
  - unable to support signals requested (Process Specification document)
- Business relationship
  - CPA unsupported without existing business relation
- Negotiation process
  - Too many counterproposals tried (no forward progress to convergence),
  - Proposed CPA previously received and not accepted.
- The current offer's "valid until" date has past.
- CPP/CPA format problems

- ♦ parsing error/data invalid
- Internal System Error

13 Negotiation Messages

A negotiation message includes information that controls the negotiation protocol along with (at least in some messages) the NDD and the CPA being negotiated.

Examples of protocol information are

- The date until this offer is valid.
- Requirements for signing the final CPA
- Error and exception information. See Section12.3.

14 References

VERSION NUMBERS AND URLS TBD

[ebBPSS] ebXML Business Process Specification Schema

[ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification, version 2.0.

[ebMS] ebXML Message Service Specification, version 2.0.

[RFC2119] Key Words for use in RFCs to indicate Requirement Levels, Internet Engineering Task Force RFC 2119, http://www.ietf.org/rfc/rfc2119.txt

[RFC2396] Uniform Resource Identifiers URI): General Syntax, Internet Engineering Task Force RFC 2396, http://www.ietf.org/rfc/rfc2396.txt

[SOAPATTACH] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs; Satish Thatte and Henrik Frystyk Nielsen, Microsoft, Published Oct 09 2000. http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211

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

[XMLENC] XML Encryption Syntax and Processing, Worldwide Web Consortium, http://www.w3.org/TR/2002/CR-xmlenc-core-20020304/

[XPATH] XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath

15 Conformance

16 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.

17 Contact Information

OTHERS TBD

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

Dale W. Moberg (Author)
Cyclone Commerce
8388 E. Hartford Drive
Scottsdale, AZ 85255
USA
Phone: 480-627-2648
email: mailto:dmoberg@cyclonecommerce.com

Notices

NEED TO DETERMINE OF UN/CEFACT HAS TO BE MENTIONED.

Portions of this document are copyright © 2002 OASIS.

Copyright (C) The Organization for the Advancement of Structured Information Standards [OASIS] 2002. *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 OASIS, except as needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it into languages other than English.

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

This document and the information contained herein is provided on an "AS IS" basis and OASIS 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.

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

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

OASIS has been notified of intellectual property rights claimed in regard to some or all of the contents of this specification. For more information consult the online list of claimed rights.

Appendix A XML Schema for Negotiation Descriptor Document

The XML Schema document for the NDD is available as a text file at:

Appendix B XML Schemas for Negotiation Messages

The XML Schemas for the negotiation messages are available in text form at:

THESE SCHEMAS SHOULD BE FOR COMPLETE EBXML MESSAGES INCLUDING THE EBXML MESSAGE SERVICE HEADERS.

Appendix C Negotiation CPA Example

The text file for this NCPA example is available at:

Appendix D BPSS Instance Document for Automated Negotiation

The text file for this example of the BPSS instance document for automated negotiation is available at:

Appendix E Example of NDD Instance Document

The text file for this example of an NDD instance document for automated negotiation is available at:

Appendix F Examples of Negotiation Message Instance Documents

The text files for the examples of negotiation message instance documents are available at:

Appendix G Glossary of Terms

This appendix contains definitions of terms created by this specification. For definitions of terms created by the CPPA Specification[ebCPP] and related terms that are part of the general ebXML vocabulary, see [ebCPP].

CPA Negotiation Process: The process by which a Collaboration Protocol Agreement (CPA) is formed based on information provided by two parties interested doing business The negotiation process is defined in a BPSS instance document.

CPA Template: A CPA template is a CPA with open fields. The schema for a CPA template is the normal CPP-CPA schema. The means of identifying open fields in the CPA template is defined in this specification.

Negotiation BPSS Instance Document: The representation of the negotiation-protocol process by means of an XML instance document that conforms to the ebXML Business Process Specification Schema specification.

Negotiation CPA (NCPA): The CPA that governs the negotiation process.

Negotiation Descriptor Document (NDD): A Negotiation Descriptor Document (NDD) describes what is negotiable in a CPP or a CPA template.

Negotiation Protocol: The negotiation process requires the exchange of data between both parties in the negotiation (and perhaps with a negotiation service). The format of these messages and the choreography of their exchanged is defined by a negotiation CPA and its corresponding BPSS instance document.

Negotiation Message: The negotiation protocol consists of exchanges of messages that contain the details of offers and counter offers. The specification defines the schema and semantics of each message.

Appendix H CPA Composition (Non-Normative)

THIS APPENDIX HAS BEEN COPIED FROM VERSION 2 OF THE CPPA SPECIFICATION. IT WILL BE RESTRUCTED AND SOME MATERIAL MOVED TO APPROPRIATE PLACES IN THE MAIN BODY OF THE SPECIFICATION.

H.1 Suggestions for Design of Computational Procedures

A quick inspection of the schemas for the top level elements, ***CollaborationProtocolProfile*** *(CPP)* and ***CollaborationProtocolAgreement*** *(CPA),* shows that a *CPA* can be viewed as a result of merging portions of the ***PartyInfo*** elements found in constituent *CPPs,* and then integrating these ***PartyInfo*** elements with other CPA sibling elements, such as those governing the *CPA* validity period.

Merging *CPPs* into *CPAs* is one way in which trading partners can arrive at a proposed or "draft" *CPA*. A draft *CPA* might also be formed from a *CPA* template. A *CPA* template represents one *Party's* proposed implementation of a *Business Process* that uses place-holding values for the identifying aspects of the other *Party*, such as ***PartyId*** or ***TransportEndpoint*** elements. To form a *CPA* from a *CPA* template, the placeholder values are replaced by the actual values for the other trading partner. The actual values could themselves be extracted from the other trading partner's *CPP,* if one is available, or they could be obtained from an administrator performing data entry functions.

We call objects draft *CPAs* to indicate their potential use as inputs to a *CPA* negotiation process in which a draft *CPA* is verified as suitable for both *Parties*, modified until a suitable *CPA* is found, or discovered to not be feasible until one side (or both) acquires additional software capabilities. In general, a draft *CPA* will constitute a proposal about an overall binding of a *Business Process* to a delivery implementation, while negotiation will be used to arrive at detailed values for parameters reflecting a final agreement. The *Negotiation Descriptor Document* provides both focus on what parameters can be negotiated as well as ranges or sets of acceptable values for those parameters.

In the remainder of this appendix, the goal will be to identify and describe the basic tasks that computational procedures for the assembly of the draft *CPA* would normally accomplish. While no normative specification is provided for an algorithm for *CPA* formation, some guidance for implementers is provided. 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.

Before enumerating and describing the basic tasks, it is worthwhile mentioning two basic reasons why we focus on the component tasks involved in *CPA* formation rather than attempt to provide an algorithm for CPA formation. These reasons provide some hints to implementers about ways in which they might customize their approaches to drafting *CPAs* from *CPPs.*

H.1.1 Variability in Inputs

User preferences provide one source of variability in the inputs to the *CPA* formation process. Let us suppose in this section that each of the *Parties* has made its *CPP* available to potential

collaborators. Normally one *Party* will have a desired *Business Collaboration* (defined in a ***ProcessSpecification*** document) to implement with its intended collaborator. So the information inputs will normally involve a user preference about intended *Business Collaboration* in addition to just the *CPPs*.

A *CPA* formation tool can have access to local user information not advertised in the *CPP* that can contribute to the *CPA* that is formed. A user can have chosen to only advertise those system capabilities that reflect capabilities that have not been deprecated. For example, a user can only advertise HTTP and omit FTP, even when capable of using FTP. The reason for omitting FTP might be concerns about the scalability of managing user accounts, directories, and passwords for FTP sessions. Despite not advertising an FTP capability, configuration software can 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 can, in this case, override the deprecation policy. Both tacit knowledge and detailed preference information account for variability in inputs into the *CPA* formation process.

H.1.2 Variable Stringency in Evaluating Proposed Agreements

The conditions for output of a *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* can propose a *CPA* that satisfies enough of the requirements for “a good enough” implementation. User input can be solicited to determine what is a good-enough implementation, and so can be as varied as there are user configuration options to express preferences. In practice, compromises can be made on security, reliable *Messaging*, levels of signals and acknowledgments, and other matters in order to find some acceptable means of doing business.

A *CPA* can support a fully interoperable configuration in which agreement has been reached on all technical levels needed for a *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 can 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* can reflect configured policies or expressly solicited user permission to ignore some shortcomings in configurations. A system might not be capable of responding in a *Business Collaboration* so as to support a specified ability to supply non-repudiation of receipt, but might still be acceptable for business reasons. A system might not be able to handle all the processing needed to support, for example, SOAP with Attachments[SOAPATTACH] and yet still be able to treat the multipart according to "multipart/mixed" handling and allow a *Business Collaboration* to take place. In fact, short of a failure to be able to transport data and a failure to be able to provide data relevant to the *Business Collaboration*, there are few features that might not be temporarily or indefinitely compromised about, given overriding business interests. This situation of "partial interoperability" is to be expected to persist for some time, and so interferes with formulating a "clean" algorithm for deciding on what is sufficient for interoperability.

H.2 CPA Formation Component Tasks

Technically viewed, a *CPA* provides "bindings" between *Business Collaboration* specifications (such as those defined within the ***ProcessSpecification***'s referenced documents) and those services and protocols that are used to implement these specifications. The implementation takes place at several levels and involves varied services at these levels. A *CPA* that arrives at a fully interoperable collaboration binding can be thought of as arriving at interoperable, application-to-application integration. *CPAs* can fall short of this goal and still be both 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 integration. Likewise, partial *CPAs* can leave significant system work to be done before a completely satisfactory application-to-application integration is realized. Even so, partial integration can be sufficient to allow collaboration, and to enjoy payoffs from increased levels of automation.

In practice, the *CPA* formation process can 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 the most important of these constituent tasks.

H.3 *CPA* Formation from *CPP*s: Context 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 ***ProcessSpecification*** between our side and our intended collaborator has been selected,
3. The ***Service, Action,*** and the specific ***Role*** elements that our software agent is to play in the *Business Collaboration* (with discussion soon restricted to ***BinaryCollaboration***s) are known, and
4. Finally, the capabilities that we have advertised in our *CPP* are known.

For vividness, we will develop our discussions using the RosettaNet™ PIP 3A4 BPSS instance document example and the *CPPs* of Company A and B that are found in full in appendices of [ebCPP] and that should also be available at the web site for the OASIS ebXML CPPA Technical Committee. For simplicity, we will assume that the information about capabilities is restricted to what is available in our agent's *CPP*, and in the *CPP* of our intended collaborator. We will suppose that we have taken on the viewpoint of Company A assembling a draft *CPA*. Please note that there is no guarantee that the same draft *CPA*s will be produced in the same order from differing viewpoints.

In general, the basic tasks consist of finding "matches" between our capabilities and our intended collaborator's capabilities at the various levels of the collaboration protocol stack and with

respect to the services supplied at these various levels. This stack, which need not be characterized in any detail, is at least distinguished by an application level and a *Messaging* transfer level. The application level is governed by a business process flow specification, such as [ebBPSS]. The *Messaging* transfer level will consist of a number of requirements and options concerning transfer protocols, security, *Packaging*, and *Messaging* patterns (such as various kinds of acknowledgment, error *Messages*, and the like.)

In actually assembling the tasks into a computational process, it will generally make sense to perform the tasks in a certain order. The overall order reflects the implicit structure of the *CPA*: first undertake those tasks to ensure that there is a match with respect to the *Business Collaboration* process. Without finding that the collaborators can participate in the same ***ProcessSpecification*** successfully, there is little point in working through implementation options. Then, examine the matches within the components of the bindings that have been announced for the *Business Collaboration* process, checking for the most indispensable "matches" first (***Transport***-related), and continuing checks on the other layers reflecting integrated interoperability at *Packaging*, security, signals and protocol patterns, and so on. With this basic overview in mind, let us proceed to consider the basic tasks in greater detail.

H.4 Business *Collaboration* Process Matching Tasks

Company A has announced within its *CPP*, at least one ***PartyInfo*** element. For current purposes, the most important initial focus is on all the sibling elements with the path ***/CollaborationProtocolProfile/PartyInfo/CollaborationRole***. Each element of this kind has a child, ***ProcessSpecification***. Our initial matching task (probably better viewed as a filtering task) is to select those nodes where the ***ProcessSpecification*** is one that we are interested in building a *CPA* for! Checking the attribute values allows us to select by comparing values in the ***name, xlink:href*** or ***uuid*** attributes. The definitive value for matching BPSS *Process Specifications* is the value found in the ***ProcessSpecification/@uuid*** attribute.

H.4.1 Matching *ProcessSpecification/Roles* and *Actions*: Initial Filtering and Selection

The previous task has essentially found two ***CollaborationRole*** node sets within our and our collaborator's *CPP* documents where the ***ProcessSpecification***s are identical, and equal to the value of interest given above. In other words, we have ***CollaborationRole***s with ***ProcessSpecification/@name***="PIP3A4RequestPurchaseOrder". It is convenient but not essential to use the ***name*** attribute in performing this selection.

We next proceed to filter these node sets. We have been given our ***Role*** element value for our participation in the ***ProcessSpecification***. For Company A, this Role has the name attribute with value "Buyer". Because we are here considering only ***BinaryCollaboration***s in BPSS terminology (or their equivalent in other flow languages), we are only interested in those ***CollaborationRole*** node sets within our collaborator's *CPP* that have a ***Role*** value equal to "Seller". So we assume we have narrowed our focus to ***CollaborationRole*** node sets in Company A's *CPP* with ***Role/@name***="Buyer" and in Company B's ***CollaborationRole*** node sets with ***Role/@name***="Seller".

For more general collaborations, such as in the ***MultiPartyCollaborations*** of [ebBPSS], we would need to know the list of roles available within the process, and keep track of that for each

of the ***CollaborationRole***s, the ***Role*** values chosen correspond correctly for the participants. We do not here discuss the matching/filtering task for collaborations involving more than two roles, as multiparty *CPA*s are not within scope for version 2.0 of [ebCPP].

H.5 Implementation Matching Tasks

After filtering the ***CollaborationRol***es with the desired ***ProcessSpecification***, we should find one ***CollaborationRole*** in our own *CPP* where we play the Buyer role and one CollaborationRole in our intended collaborator Company B's CPP where it plays the Seller role.

Our next task is to locate the specific candidate bindings relevant to *CPA* formation. There are bindings for ***Service*** and ***Action***s. For initial simplicity, we consider detailed matching tasks as they arise for a standard collaboration case involving a request ***Action***, followed by a response ***Action***. For version 2.0 of [ebCPP], most matching tasks will involve matching of referenced components of the *CPP*s ***ThisPartyActionBinding*** elements under ***CollaborationRole/ServiceBinding/CanSend/*** and under ***CollaborationRole/ServiceBinding/CanReceive***.

H.5.1 *Action* Correspondence and Selecting Correlative *PackageIds* and *ChannelIds*

In *CPP*s, under each of the elements ***CollaborationRole/ServiceBinding/CanSend*** and ***CollaborationRole/ServiceBinding/CanReceive*** are lists of ***ThisPartyActionBinding***s. For request-response collaboration patterns, we are interested in matches:

1. In the bindings of the requesting side's ***CanSend/ThisPartyActionBinding*** with the Responding side's ***CanReceive/ThisPartyActionBinding*** for the request ***Action***, and
2. In the bindings of the Responding side's ***CanSend/ThisPartyActionBinding*** with the requesting side's ***CanReceive/ThisPartyActionBinding*** for the response ***Action***.

These correlative bindings give us references to detailed components that need to match for a fully interoperable agreement. Case 1 pertains to the request. Case 2 pertains to the response.

For example, for Company A, we find under ***CanSend***:

```
<tp:ThisPartyActionBinding tp:action="Purchase Order Request Action"
tp:packageId="CompanyA_RequestPackage">
      <tp:BusinessTransactionCharacteristics ... />
      <tp:ActionContext tp:binaryCollaboration="Request Purchase Order"
      tp:businessTransactionActivity="Request Purchase Order"
      tp:requestOrResponseAction="Purchase Order Request Action"/>
      <tp:ChannelId>asyncChannelA1</tp:ChannelId>
</tp:ThisPartyActionBinding>
```

Correlative to this, for Company B, we find under ***CanReceive***:

```
<tp:ThisPartyActionBinding tp:action="Purchase Order Request Action"
tp:packageId="CompanyB_RequestPackage">
      <tp:BusinessTransactionCharacteristics ... />
      <tp:ActionContext tp:binaryCollaboration="Request Purchase Order"
      tp:businessTransactionActivity="Request Purchase Order"
      tp:requestOrResponseAction="Purchase Order Request Action"/>
      <tp:ChannelId>asyncChannelB1</tp:ChannelId>
```

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

The correlation of elements can normally (when we are dealing with BPSS ***BinaryCollaborations*** or their equivalents in other representations) be based on equality of the ***Action*** (or ***requestOrResponseAction***) values. More detailed correlation of elements can make use of more detailed testing and comparisons of the values in the ***ActionContex***t child elements of the relevant ***CanSend*** and ***CanReceive*** pairs.

In the preceding, we have illustrated the matching of ***CanSend*** and ***CanReceive*** for asynchronous bindings. All ***CanSend*** bindings that are siblings under a ***ServiceBinding*** element are asynchronous and make of use separate TCP connections that the ***CanSend*** side initiates on a listening TCP port. In order to represent binding details for synchronous sending, the convention is adopted whereby the ***CanSend*** element for a *Receiver* is placed under its ***CanReceive*** element. This is illustrated by:

```
<tp:CanSend>
   <tp:ThisPartyActionBinding
    tp:id="companyA_ABID6"
    tp:action="Purchase Order Request Action"
    tp:packageId="CompanyA_RequestPackage">
   <tp:BusinessTransactionCharacteristics
      tp:isNonRepudiationRequired="true"
      tp:isNonRepudiationReceiptRequired="true"
      tp:isConfidential="transient"
      tp:isAuthenticated="persistent"
      tp:isTamperProof="persistent"
      tp:isAuthorizationRequired="true"
      tp:timeToAcknowledgeReceipt="PT2H"
      tp:timeToPerform="P1D"/>
     <tp:ActionContext
      tp:binaryCollaboration="Request Purchase Order"
      tp:businessTransactionActivity="Request Purchase Order"
      tp:requestOrResponseAction="Purchase Order Request Action"/>
    <tp:ChannelId>syncChannelA1</tp:ChannelId>
    </tp:ThisPartyActionBinding>
    <tp:CanReceive>
        <tp:ThisPartyActionBinding
         tp:id="companyA_ABID7"
         tp:action="Purchase Order Confirmation Action"
         tp:packageId="CompanyA_SyncReplyPackage">
         <tp:BusinessTransactionCharacteristics
           tp:isNonRepudiationRequired="true"
           tp:isNonRepudiationReceiptRequired="true"
           tp:isConfidential="transient"
           tp:isAuthenticated="persistent"
           tp:isTamperProof="persistent"
           tp:isAuthorizationRequired="true"
           tp:timeToAcknowledgeReceipt="PT2H"
           tp:timeToPerform="P1D"/>
         <tp:ActionContext
          tp:binaryCollaboration="Request Purchase Order"
          tp:businessTransactionActivity="Request Purchase Order"
          tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
         <tp:ChannelId>syncChannelA1</tp:ChannelId>
        </tp:ThisPartyActionBinding>
    </tp:CanReceive>
    <tp:CanReceive>
       <tp:ThisPartyActionBinding
        tp:id="companyA_ABID8"
```

```
          tp:action="Exception"
          tp:packageId="CompanyA_ExceptionPackage">
         <tp:BusinessTransactionCharacteristics
           tp:isNonRepudiationRequired="true"
           tp:isNonRepudiationReceiptRequired="true"
           tp:isConfidential="transient"
           tp:isAuthenticated="persistent"
           tp:isTamperProof="persistent"
           tp:isAuthorizationRequired="true"
           tp:timeToAcknowledgeReceipt="PT2H"
           tp:timeToPerform="P1D"/>
       <tp:ChannelId>syncChannelA1</tp:ChannelId>
      </tp:ThisPartyActionBinding>
       </tp:CanReceive>
  </tp:CanSend>
```

This subordination will also carry over to the synchronous receiving side, in which its ***CanReceive*** element(s) is (are) under the ***CanSend*** element used to represent the initial sending of a request. An illustration from Company B's synchronous binding is:

```
<tp:CanReceive>
  <tp:ThisPartyActionBinding
  tp:id="companyB_ABID8"
  tp:action="Purchase Order Request Action"
  tp:packageId="CompanyB_SyncReplyPackage">
  <tp:BusinessTransactionCharacteristics
     tp:isNonRepudiationRequired="true"
     tp:isNonRepudiationReceiptRequired="true"
     tp:isConfidential="transient"
     tp:isAuthenticated="persistent"
     tp:isTamperProof="persistent"
     tp:isAuthorizationRequired="true"
     tp:timeToAcknowledgeReceipt="PT5M"
     tp:timeToPerform="PT5M"/>
  <tp:ActionContext
     tp:binaryCollaboration="Request Purchase Order"
     tp:businessTransactionActivity="Request Purchase Order"
     tp:requestOrResponseAction="Purchase Order Request Action"/>
    <tp:ChannelId>syncChannelB1</tp:ChannelId>
    </tp:ThisPartyActionBinding>
    <tp:CanSend>
      <tp:ThisPartyActionBinding
       tp:id="companyB_ABID6"
       tp:action="Purchase Order Confirmation Action"
       tp:packageId="CompanyB_ResponsePackage">
       <tp:BusinessTransactionCharacteristics
          tp:isNonRepudiationRequired="true"
          tp:isNonRepudiationReceiptRequired="true"
          tp:isConfidential="transient"
          tp:isAuthenticated="persistent"
          tp:isTamperProof="persistent"
          tp:isAuthorizationRequired="true"
          tp:timeToAcknowledgeReceipt="PT5M"
          tp:timeToPerform="PT5M"/>
       <tp:ActionContext
        tp:binaryCollaboration="Request Purchase Order"
        tp:businessTransactionActivity="Request Purchase Order"
        tp:requestOrResponseAction="Purchase Order Confirmation Action"/>
      <tp:ChannelId>syncChannelB1</tp:ChannelId>
     </tp:ThisPartyActionBinding>
    </tp:CanSend>
   <tp:CanSend>
```

```
      <tp:ThisPartyActionBinding
        tp:id="companyB_ABID7"
        tp:action="Exception"
        tp:packageId="CompanyB_ExceptionPackage">
      <tp:BusinessTransactionCharacteristics
        tp:isNonRepudiationRequired="true"
        tp:isNonRepudiationReceiptRequired="true"
        tp:isConfidential="transient"
        tp:isAuthenticated="persistent"
        tp:isTamperProof="persistent"
        tp:isAuthorizationRequired="true"
        tp:timeToAcknowledgeReceipt="PT5M"
        tp:timeToPerform="PT5M"/>
      <tp:ChannelId>syncChannelB1</tp:ChannelId>
       </tp:ThisPartyActionBinding>
      </tp:CanSend>
</tp:CanReceive>
```

H.5.2 Matching and Checking *DeliveryChannel* Details

Until now, most of the matching work has been undertaken to find pairs of correlative ***xxxActionBinding***, and so the matching has functioned as a filtering mechanism. Once in possession of pairs of correlative ***xxxActionBindings***, however, the work of checking for matches across the various dimensions of operation — transport, transport security, PKI compatibility for various tasks, agreement about *Messaging* characteristics (reliable *Messaging*, digital enveloping, signed acknowledgments (minimal non-repudiation of receipt), non-repudiation of origin, *Packaging* details, and more — begins.

Once in possession of the ***xxxActionBindings***, IDREFs provide references to the underlying components for comparison. For example, when comparing *Packaging* details, the request IDREFS are found at ***CanSend/ThisPartyActionBinding/@packageId*** and within the other *CPP* at ***CanReceive/ThisPartyActionBinding@packageId***. For Company A's request "Purchase Order Request Action," the *Packaging* IDREF is found in:

```
tp:packageId="CompanyA_RequestPackage"
```

and this IDREF value refers to:

```
<tp:Packaging tp:id="CompanyA_RequestPackage">
      <tp:ProcessingCapabilities tp:parse="true" tp:generate="true"/>
      <tp:CompositeList>
<tp:Composite
      tp:id="CompanyA_RequestMsg"
      tp:mimetype="multipart/related"
      tp:mimeparameters="type=text/xml;">
      <tp:Constituent tp:idref="CompanyA_MsgHdr"/>
      <tp:Constituent tp:idref="CompanyA_Request"/>
      </tp:Composite>
      </tp:CompositeList>
</tp:Packaging>
```

For Company A's request "Purchase Order Request Action", the delivery channel IDREF is found in:

```
<tp:ChannelId>asyncChannelA1</tp:ChannelId>
```

and this IDREF value refers to the element with this ID, namely:

```
<tp:DeliveryChannel tp:channelId="asyncChannelA1" tp:transportId="transportA1"
tp:docExchangeId="docExchangeA1">
<tp:MessagingCharacteristics
     tp:syncReplyMode="none"
     tp:ackRequested="always"
     tp:ackSignatureRequested="always"
     tp:duplicateElimination="always"/>
</tp:DeliveryChannel>
```

Two remaining crucial references for understanding the binding, are found in attributes of the ***DeliveryChannel***, namely: ***DeliveryChannel/@transportId*** and in the attribute ***DeliveryChannel/@docExchangeId.***

For Company A, for example, we find ***transportId***="transportA1" and ***docExchangeId***="docExchangeA1" are the IDREFs for the continuing binding information with the ***DeliveryChannel,*** "asyncChannelA1". Resolving these references, we obtain:

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

for ***transportID*** "transportA1" and

```
<tp:DocExchange tp:docExchangeId="docExchangeA1">
      <tp:ebXMLSenderBinding tp:version="2.0">
      <tp:ReliableMessaging>
      <tp:Retries>3</tp:Retries>
      <tp:RetryInterval>PT2H</tp:RetryInterval>
      <tp:MessageOrderSemantics>Guaranteed</tp:MessageOrderSemantics>
      </tp:ReliableMessaging>
      <tp:PersistDuration>P1D</tp:PersistDuration>
      <tp:SenderNonRepudiation>
      <tp:NonRepudiationProtocol>http://www.w3.org/2000/09/xmldsig#
      </tp:NonRepudiationProtocol>
```

```
        <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1
        </tp:HashFunction>
        <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-sha1
        </tp:SignatureAlgorithm>
        <tp:SigningCertificateRef tp:certId="CompanyA_SigningCert"/>
        </tp:SenderNonRepudiation>
        <tp:SenderDigitalEnvelope>
        <tp:DigitalEnvelopeProtocol
        tp:version="2.0">S/MIME</tp:DigitalEnvelopeProtocol>
        <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
        <tp:EncryptionSecurityDetailsRef
        tp:securityId="CompanyA_MessageSecurity"/>
        </tp:SenderDigitalEnvelope>
        </tp:ebXMLSenderBinding>
        <tp:ebXMLReceiverBinding tp:version="2.0">
        <tp:ReliableMessaging>
        <tp:Retries>3</tp:Retries>
        <tp:RetryInterval>PT2H</tp:RetryInterval>
        <tp:MessageOrderSemantics>Guaranteed</tp:MessageOrderSemantics>
        </tp:ReliableMessaging>
        <tp:PersistDuration>P1D</tp:PersistDuration>
        <tp:ReceiverNonRepudiation>
        <tp:NonRepudiationProtocol>http://www.w3.org/2000/09/xmldsig#
        </tp:NonRepudiationProtocol>
        <tp:HashFunction>http://www.w3.org/2000/09/xmldsig#sha1
        </tp:HashFunction>
        <tp:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-sha1
        </tp:SignatureAlgorithm>
        <tp:SigningSecurityDetailsRef
        tp:securityId="CompanyA_MessageSecurity"/>
        </tp:ReceiverNonRepudiation>
        <tp:ReceiverDigitalEnvelope>
        <tp:DigitalEnvelopeProtocol
        tp:version="2.0">S/MIME</tp:DigitalEnvelopeProtocol>
        <tp:EncryptionAlgorithm>DES-CBC</tp:EncryptionAlgorithm>
        <tp:EncryptionCertificateRef tp:certId="CompanyA_EncryptionCert"/>
        </tp:ReceiverDigitalEnvelope>
        </tp:ebXMLReceiverBinding>
</tp:DocExchange>
```

for the ***docExchangeId***, docExchangeA1.

There are, of course, other references, such as those to security-related capabilities, that will be important to resolve when checking detailed matching properties, but the four IDREFs (two for the sender and two for the *Receiver*) that have just been introduced are critical to the remainder of the match tests that will lead to the formation of draft *CPA*s. We will assume at this point that the reader can resolve IDREFs using the example *CPPs* and *CPAs* for Company A and B in the appendices, and will not exhibit them in the text in order to save space.

We next turn to a more in-depth treatment of the tests that are involved in finding the elements for a draft *CPA*.

The detailed tasks to be discussed in greater depth are:

1. Matching Channel ***MessagingCharacteristics***
2. Checking ***BusinessTransactionCharacteristics*** coherence with ***DeliveryChannel*** details
3. Matching ***Packaging***

4. Matching ***Transport*** and ***Transport***[***Receiver***|***Sender***]***Security***
5. Matching and checking ***DocExchange*** subtrees.

Because agreement about ***Transport*** is quite fundamental, we shall consider it first. Computational processes are likely to first find pairs that match on ***Transport*** details, and will ignore pairs failing to have matches at this level.

H.5.2.1 Matching *Transport*

Matching ***Transport*** first involves matching the ***Transport/TransportSender/TransportProtocol*** capabilities of the requester with the ***Transport/TransportReceiver/TransportProtocol*** capabilities found under the collaborator receiving the request. Several such matches can exist, and any of these matches can be used in forming a draft, provided other aspects match up satisfactorily. Each *CPP* is assumed to have listed its preferred transport protocols first (as determined by the listing of the Bindings that reference the ***Transport*** element, but different outcomes can result depending on which *CPP* is used first for searching for matches. In general, resolution of preference differences is left to a distinct phase of *CPA* negotiation, following proposal of a draft *CPA*. Negotiation can be performed by explicit ***Actions*** of users, but is expected to become increasingly automated.

Matching transport secondly involves matching the ***TransportSender/TransportProtocol*** capabilities of the responding collaborator with its ***TransportReceiver/TransportProtocol*** capabilities found under the collaborator receiving the response, which is typically the collaborator that has sent a request. Several such matches can exist, and any of these matches can be used in forming a draft. In one case, however, there may be no need for the second match on ***TransportProtocol.*** If we are using HTTP or some other protocol supporting synchronous replies and the ***DeliveryChannel*** has a ***MessagingCharacteristics*** child that has its ***syncReplyMode*** attribute with a value of "signalsAndResponse," then everything comes back synchronously, and there is no need to match on ***TransportProtocol*** for the response ***DeliveryChannel.***

If ***TransportSecurity*** is present, then there can be additional checks. First, ***TransportSender/TransportClientSecurity/TransportSecurityProtocol*** should be compatible with ***TransportReceiver/TransportServerSecurity/TransportSecurityProtocol***. Second, if either the ***TransportSender/TransportClientSecurity/ClientSecurityDetailsRef*** or ***TransportSender/TransportClientSecurity/ServerSecurityDetailsRef*** elements are present, and the IDREF references an element containing some ***AnchorCertificateRef,*** then an opportunity exists to check suitability of one *Party's* PKI trust of the certificates used in the ***TransportSecurityProtocol.*** For example, by resolving the IDREF value in ***TransportSender/TransportClientSecurity/ClientCertificateRef/@certId***, we can obtain the proposed client certificate to use for client-side authentication. By resolving the IDREFs from the ***AnchorCertificateRef,*** we become able to determine whether the proposed client certificate will "chain to a trusted root" on the server side's PKI. Similar remarks apply to checks on the validity of a server certificate found by resolving ***TransportReceiver/TransportServerSecurity/ServerCertificateRef*** . This server certificate can be checked against the CA trust anchors that are found by resolving ***TransportSender/TransportClientSecurity/ServerSecurityDetailsRef/@securityId***, and finding CA certificates (or CA certificate chains) in the ***KeyInfo*** elements under the Certificate element

obtained by resolving the IDREF found in ***AnchorCertificateRef@certId.***

When matches exist for the correlative ***Transport*** components, we then have discovered an interoperable solution at the transport level. If not, no *CPA* will be available, and a gap has been identified that will need to be remedied by whatever exception handling procedures are in place. Let us next consider other capabilities that need to match for "thicker" interoperable solutions.

H.5.2.2 Checking *BusinessTransactionCharacteristics* and *DeliveryChannel MessagingCharacteristics*

Under each of the correlative ***xxxActionBindings***, there is a child element of ***DeliveryChannel, MessagingCharacteristics,*** that has several attributes important in *CPA* formation tasks. The attributes having wider implications are ***syncReplyMode, ackRequested***, and ***ackSignatureRequested***; for the ***duplicateElimination*** and ***actor*** attributes, compatibility exists when the attributes that are found under the ***CanSend*** and ***CanReceive DeliveryChannels*** have the same values. As the element's name implies, all of these ***DeliveryChannel*** features pertain to the *Messaging* layer.

In addition, ***BusinessTransactionCharacteristics,*** found under ***ThisPartyActionBinding,*** contains attributes reflecting a variety of features pertaining to desired security and *Business Transaction* properties that are to be implemented by the agreed upon ***DeliveryChannels***. These properties may have implications on what capabilities are needed within more detailed components of the ***DeliveryChannel*** elements, such as in the ***Packaging*** element. When using a *BPSS* ***ProcessSpecification***, these properties may be specified within the ***BusinessTransaction.*** The properties of the ***BusinessTransactionCharacteristics*** element are, however, the ones that will be operative in the implementation of the ***BusinessTransaction,*** and may override the specified values found in the BPSS ***ProcessSpecification.*** Because the properties are diverse, the details that implement the properties can be spread over other elements referenced within the ***DeliveryChannel*** elements.

These attributes apply to either a request or a response delivery channel, but can impact either the *Sender* or *Receiver* (or both) in a channel. In addition, the attributes governing acknowledgments, for example, qualify the interrelation of ***DeliveryChanne***l elements by specifying behavior that is to occur that qualifies the contents of a return *Message*.

The most basic test for compatibility for any of the attributes in either ***MessagingCharacteristics*** or ***BusinessTransactionCharacteristics*** is that the attributes are equal in the sending *Party*'s ***DeliveryChannel*** referenced by ***CanSend/ThisPartyActionBinding/ChannelId*** and in the receiving *Party*'s ***DeliveryChannel*** referenced by ***CanReceive/ThisPartyActionBinding/ChannelId***. If they are unequal, and all bindings have been examined on both sides, a draft *CPA* will represent a compromise to some common set with respect to the functionality represented by the attributes.

In the following discussions, we will consider many of the attributes in the two ***xxxCharacteristics*** elements, and relate them to additional underlying implementational details, one of which is ***Packaging.***

From a high level, basic agreement in *Packaging* is a matter of compatibility of the generated *Packaging* on the sending side with the parsed *Packaging* on the receiving side. The basic *Packaging* check is, therefore, checking *Packaging* compatibility under the ***CanSend*** element of a sender ***Action*** with the *Packaging* under the ***CanReceive*** element of that same ***Action*** under the *Receiver* side.

For efficiency, representation of capabilities of parsing/handling *Packaging* can make use of both wildcards and repetition, and as needed these capabilities can also express open data formatting used on the generating side. For example, consider the ***SimplePart:***

```
<tp:SimplePart tp:id="IWild" tp:mimetype="*/*"/>
```

By wildcarding ***mimetype*** values, we represent our capability of accepting any data, and would match any specific MIME type. Also, consider a ***Constituent*** appearing within a ***Composite:***

```
      <tp:Constituent tp:idref="MsgHdr"/>
      <tp:Constituent minOccurs="0" maxOccurs="10" tp:idref="IWild"/>
```

This notation serves to capture the capability of handling any number of arbitrary MIME bodyparts within the ***Composite*** being defined. A *Packaging* capability such as this would obviously match numerous more specific generated *Packaging* schemes, as well as matching literally with a scheme of the same generality.

Certain more complex checks are needed for more complicated *Packaging* options pertaining to ***syncReplyMode***. These are discussed in the following.

syncReplyMode

The ***syncReplyMode*** attribute has a value other than "none" to indicate what parts of a *Message* should be returned in the reply of a transport capable of synchronous operation, such as HTTP. (We here use "synchronous" to mean "on the same TCP connection," which is one use of this term. We do not specify any waiting, notification, or blocking behavior on processes or threads that are involved, though presumably there is some computational activity that maintains the connection state and is above the TCP and socket layers.)

The possible implementations pertaining to various values of the ***syncReplyMode*** attributes are numerous, but we will try to indicate at least the main factors that are involved.

As will be seen, the ***Packaging*** element is important in specifying implementation details and compatibilities. But, because business-level signals may be involved, other ***xxxActionBindings*** may need examination in addition to the already selected bindings for the request and response. Also, the values of ***TransportReceiver/Endpoint/@type*** might need checking when producing draft *CPA*s.

Let us first begin with the cases in which responses*, Message Service Handler* signals and business signals return in some combination of a synchronous reply and other asynchronous *Message*(s). These various combinations will be discussed for the ***syncReplyMode*** values: "mshSignalsOnly," "signalsOnly," "responseOnly", and "signalsAndResponse".

By convention, synchronous replies are represented by subordinating ***CanSend*** or ***CanReceive*** elements under the ***CanReceive*** or ***CanSend*** elements that represent the initial request binding capabilities. For representing asynchronous requests, replies, or signals, the ***CanSend*** or ***CanReceive*** elements are all siblings and directly subordinate to the ***ServiceBinding***. Therefore, both asynchronous and synchronous capabilities can be grouped under a ***ServiceBindin***g in a *CPP*, and can still be unambiguously distinguished. In principle, increasing subordination (nesting) can indicate patterns of dialog more elaborate than request and response. Few use cases for this functionality are common at the time of this writing.

mshSignalsOnly

The request sender's ***DeliveryChannel*** (referenced by ***CanSend/ThisPartyActionBinding/ChannelId***) and the request *Receiver*'s ***DeliveryChannel*** (referenced by ***CanReceive/ThisPartyActionBinding/ChannelId***) both should have ***MessagingCharacteristics/@syncReplyMode*** value of "mshSignalsOnly"**.**

While a *Party* can explicitly identify a ***DeliveryChannel*** for the SOAP envelope with subordinate ***CanSend*** and ***CanReceive*** elements, and with them specialized bindings, these are typically omitted for ebXML *Messaging* software. It is presumed that each side can process a synchronous reply constructed in accordance with ebXML *Messaging*. The ***DeliveryChannel*** representation mechanism here serves as a placeholder for capturing other *Messaging* signal protocols that might emerge.

Currently, acknowledgments and signed acknowledgments, along with errors, are the primary *Message* Service signals that are included in the SOAP envelope. If Company A set ***syncReplyMode*** to ***mshSignalsOnly***, then Company B's correlative ***CanReceive/ThisPartyActionBinding/@packageId*** should contain a nested ***CanSend/ThisPartyActionBinding/@packageId*** for a *Message* without any business payload or signals. In addition, the ***CanSend/ThisPartyActionBinding/@packageId*** of Company B's response should resolve to *Packaging* format capable of returning the response (and possibly other constituents) asynchronously. The compatibility of the ***DeliveryChannel*** elements can be checked, as can the capability of Company A to receive that response payload, the signal payload(s), or responses bundled with signals as specified by the *Packaging* formats that are referenced through the relevant ***ThisPartyActionBinding*** element's ***packageId*** attribute values.

signalsOnly

The request sender's ***DeliveryChannel*** (referenced by its ***CanSend/ThisPartyActionBinding/ChannelId***) and the request *Receiver*'s ***DeliveryChannel*** (referenced by its ***CanReceive/ThisPartyActionBinding/ChannelId***) both should have ***MessagingCharacteristics/@syncReplyMode*** value of ***signalsOnly.***

If Company A sets ***syncReplyMode*** to "signalsOnly", then under Company B's correlative ***CanReceive*** element, there should be a nested ***CanSend/ThisPartyActionBinding*** whose ***packageId*** attribute's value resolves to a *Packaging* format appropriate for signals. For the ***CanSend/ThisPartyActionBinding/@packageId*** associated with Company B's business-level response, the attribute IDREF value should resolve to a *Packaging* format capable of returning

payloads and that omits business signals. This ***CanSend*** element will be a direct child of ***ServiceBinding***, a placement representing its asynchronous character. The original requesting *Party* will need to have a ***CanReceive/ThisPartyActionBinding*** that is compatible with the responding *Party*, and that is a direct child of its ***ServiceBinding*** element.

Using subordinate ***CanSend*** and subordinate ***CanReceive*** elements can be useful if the ***DeliveryChannel*** details for exception signals differ from those specified for request and response. signal bindings, for example, may differ by omitting ***ackRequested***, or possibly one of the security features (digital enveloping or non-repudiation of receipt) that are used for requests or responses. Just as with other tests on requests and responses, there can be checks for compatibility in ***Packaging, DocExchange, MessagingCharacteristics***, or ***BusinessTransactionCharacteristics*** referred to in the correlative subordinate ***CanSend*** and ***CanReceive DeliveryChannels***.

responseOnly
The request sender's ***DeliveryChannel*** (referenced by ***CanSend/ThisPartyActionBinding/ChannelId***) and the request *Receiver*'s ***DeliveryChannel*** (referenced by ***CanReceive/ThisPartyActionBinding/ChannelId***) both should have ***MessagingCharacteristics/@syncReplyMode*** value of "responseOnly".

If Company A sets ***syncReplyMode*** to "responseOnly", the ***CanSend/ThisPartyActionBinding/@packageId*** of Company B's response should resolve to a *Packaging* format capable of returning payloads, but omitting business signals. The ***CanSend/ThisPartyActionBinding*** element will be included as a child of the ***CanReceive*** element so the responder can indicate that it is a synchronous response.

There should be an independent way to return business-level error signals. So, there should be a ***ThisPartyActionBinding*** for any signal payload announced, and these bindings should be at the direct child of ***ServiceBinding*** level to represent their asynchronous flavor.

It is not too likely that ***ReceiptAcknowledgment*** and similar signals will be used when a response is returned synchronously. The motivation for using these signals is indicating positive forward progress, and this motivation will be undermined when a response is returned directly.

For the "responseOnly" case, including subordinate ***CanSend/ThisPartyActionBinding*** and ***CanReceive/ThisPartyActionBinding***, means that there can be checks for compatibility in ***Packaging, DocExchange, MessagingCharacteristics***, or ***BusinessTransactionCharacteristics.*** The ***syncReplyMode*** and ***ackRequested*** attributes here should be carefully considered because a "mshSignalsOnly" value here would mean that another round of synchronous *Messaging* will need to occur on the same connection. Incidentally, for ***Transport*** elements referenced under subordinate bindings, there need not be any ***Endpoint*** elements. If there are ***Endpoint*** elements, they may be ignored.

signalsAndResponse
The request sender's ***DeliveryChannel*** (referenced by ***CanSend/ThisPartyActionBinding/ChannelId***) and the request *Receiver*'s ***DeliveryChannel***

(referenced by ***CanReceive/ThisPartyActionBinding/ChannelId***) both should have ***MessagingCharacteristics/@syncReplyMode*** value of "signalsAndResponse".

If Company A sets ***syncReplyMode*** to "signalsAndResponse", the ***CanSend/ThisPartyActionBinding*** of Company B's response should be subordinate to Company B's ***CanReceive*** element. The *Packaging* format that is referenced should be capable of returning payloads and signals bundled together. If no asynchronous bindings exist for error signals, this will be the only defined ***DeliveryChannel*** agreed to for all aspects of *Message* exchange for the *Business Transaction*. However, it is likely that an asynchronous binding would normally be provided to send exception signals.

ackRequested and ackSignatureRequested

Checks on the ***ackRequested*** and ***ackSignatureRequested*** attributes within correlative ***DeliveryChannels*** (that is, correlative because referenced under one ***Action***'s ***CanSend*** and ***CanReceive*** elements) are primarily to see that the values of the corresponding attributes are the same.

However, there are some interactions of these attributes with other information items that need to be mentioned.

The principal use of the ***ackRequested*** attribute is within reliable *Messaging* configurations. If reliable messaging is to be configured, then checks on agreement in the correlative ***ReliableMessaging*** elements as found under ***DocExchange/ebXMLSenderBinding*** and ***DocExchange/ebXMLReceiverBinding*** are in order. Also, the value of the ***duplicateElimination*** attribute of ***MessagingCharacteristics*** should be checked for agreement. Draft *CPAs* may be formed by deliberately aligning values that are not equal along some of these dimensions. Downgrading may provide draft *CPAs* most likely to gain acceptance; so, for example, if ***duplicateElimination*** is "false" on the receiving side, aligning it to "false" on the sending side is most likely to produce a draft that succeeds.

The additional function of ***ackSignatureRequested*** is that it provides a "thin" implementation for non-repudiation of receipt. The basic check is for equality of attribute value, but additional constraints may need test and alignment. If no signal capable of implementing non-repudiation of receipt is found under the ***ServiceBinding***, then having an "always" value for ***ackSignatureRequested*** suggests aligning the ***BusinessTransactionCharacteristics*** attributes, ***isNonRepudiationReceiptRequired***, to be "true". However, if this is done, care should be taken to check that the ***BusinessTransactionCharacteristics*** attribute ***isIntelligibleCheckRequired*** is "false". This is because the *Messaging* implementation only deals with receipt in the sense of having received a byte stream off the wire (and persisting it so that it is available for further processing). It is not safe to presume that any syntactical or semantic checks on the data were performed.

H.5.2.3 *DocExchange* Checks for *BusinessTransactionCharacteristics*

When using *CPPs* and *CPAs* with ebXML *Messaging*, which is the most likely early deployment situation, there exists an opportunity to check agreement on ***BusinessTransactionCharacteristics*** attributes.

The following three attributes need to have equal values in the bindings for a request or for a response. No further discussion will be provided in this appendix on these "deadlines," except to say that a sophisticated proposed *CPA* generation tool might check on the coherence of the values chosen here with values for reliable *Messaging* parameters, existence of compatible ***ReceiptAcknowledgment*** or ***AcceptanceAcknowledgment*** bindings, and consistency with ***syncReplyMode*** internal configuration.

```
<attribute name="timeToAcknowledgeReceipt" type="duration"/>
<attribute name="timeToAcknowledgeAcceptance" type="duration"/>
<attribute name="timeToPerform" type="duration"/>
```

The remaining attributes involve a number of security related issues and will be the focus of the remaining discussion of ***BusinessTransactionCharacteristics*** attributes:

```
<attribute name="isNonRepudiationRequired" type="boolean"/>
<attribute name="isNonRepudiationReceiptRequired" type="boolean"/>
<attribute name="isIntelligibleCheckRequired" type="boolean"/>
<attribute name="isAuthenticated" type="tns:persistenceLevel.type"/>
<attribute name="isTamperProof" type="tns:persistenceLevel.type"/>
<attribute name="isAuthorizationRequired" type="boolean"/>
<attribute name="isConfidential" type="tns:persistenceLevel.type"/>
```

Here, the basic test is that for correlative ***DeliveryChannels***, the corresponding attributes have the same values. Again there are some interaction aspects with parts of the ***DeliveryChanne***l that motivate making some additional checks.

Previously, when discussing the ***MessagingCharacteristics*** attribute ***ackSignatureRequested***, it was pointed out that the *Messaging* implementation provides thin support for holding ***isNonRepudiationReceiptRequired*** "true" provided that the attribute ***isIntelligibleCheckRequired*** is "false". When both are "true", then there should exist a business signal with compatible ***Packaging*** and ***DeliveryChannel*** values. If the signal has been independently described within asynchronous ***CanSend*** and ***CanReceive*** elements, knowing the signal name (such as, "ReceiptAcknowlegment") may support a relatively simple search and test. However, if synchronous ***Transport***s are involved, some filters using ***syncReplyMode***s may be needed to discover an underlying support for a "thick" implementation of non-repudiation of receipt.

When non-repudiation of receipt is implemented by a business signal, then checks on signing certificate validity can involve the ***CollaborationRole/ApplicationCertificateRef*** and the ***CollaborationRole/ApplicationSecurityDetailsRef*** that provides a reference to the ***SecurityDetails*** element containing the list of ***TrustAnchors***. The certificate from the side signing the ***ReceiptAcknowledgment*** would be checked against the certificates referred to by the ***AnchorCertificateRef*** under ***TrustAnchors***.

The business signal will sometimes be conveyed as part of a *Message*. It remains true that the *Message* itself will still be sent through a *Message* Service Handler, and that the *Message* Service Handler can also sign the *Message* using the certificate found by resolving the IDREF found at

DocExchange/ebXMLSenderBinding/SenderNonRepudiation/SigningCertificateRef/@certId.

If a particular software component implements both *Message* Service Handler functionality and business-level security functionality, it is possible that the same certificate may be pointed to by ***ApplicationCertificateRef*** and ***SigningCertificateRef/@certId***. In other words, the distinction between *Message* Service Handler-level signing and application level signing is a logical one, and may not correspond with software component boundaries. Because the *Message* Service Handler signature is over the *Message*, the *Message* signature may be over an application-level signature. While this may be redundant for some system configurations, protocols may require both signatures to exist over the different regions.

Failure to validate a certificate may not prevent formation of a draft *CPA*. First, the sender's signing certificate can be a self-signed certificate. If so, a reference to this self-signed certificate may be added to the *Receiver*'s ***TrustAnchors/AnchorCertificateRef*** list. This proposal amounts to proposing to agree to a direct trust model, rather than a hierarchical model involving certificate authorities. Second, a proposal to add a trusted root may be made, again by appropriate revision of the ***TrustAnchors***.

When non-repudiation of receipt is implemented by the *Messaging* layer, the checks on PKI make use of elements under ***DocExchange***.

isNonRepudiationRequired
isAuthenticated
isAuthorizationRequired
isTamperProof

The ideas of authentication, authorization, non-repudiation and being "tamper proof" may be very distinct as business-level concepts, yet the implementation of these factors tend to use very similar technologies. Actually, prevention of tampering is not literally implemented. Instead, means are provided for detecting that tampering (or some accidental garbling) has occurred. Likewise, implementations of authorization usually are provided by implementations of access control (permitting or prohibiting a user in a role making use of a resource) and presentation of a token or credential to gain access, which may involve authentication as an initial step! Non-repudiation may build on all the previous functions, plus retaining information for supplying presumptive evidence of origination at some later time.

When checking whether ***isNonRepudiationRequired*** can be set to "True" for both *Parties*, check whether the signing certificate will be counted as valid at the *Receiver*.
The IDREF reference to the signing certificate is found in
DocExchange/ebXMLSenderBinding/SenderNonRepudiation/SigningCertificateRef/@certId.
The referenced certificate should be checked for validity with respect to the trust anchors obtained from ***TrustAnchors/AnchorCertificateRef*** elements under the ***SecurityDetails*** element referenced by the IDREF at
DocExchange/ebXMLReceiverBinding/ReceiverNonRepudiation/SigningSecurityDetailsRef/@securityId.

As previously noted, failure to validate a certificate does not prevent constructing a draft *CPA*. Either self-signed certificates or new trust anchors can be added to align the trust model on one

side with the other side's certificate.

In addition to checking the interoperability of the PKI infrastructures, checks on compatibility of values in the other attributes in ***DocExchange/ebXMLReceiverBinding/ReceiverNonRepudiation*** and in ***DocExchange/ebXMLSenderBinding/SenderNonRepudiation*** can be made. ***NonRepudiationProtocol, HashFunction,*** and ***SignatureAlgorithm*** values may be compatible even when not equal if knowledge of the protocol requirements allows fallback to a mandatory-to-implement value. So values here can be found equal, aligned, or negotiated to reach an agreement.

If ***isNonRepudiationRequired*** is "True", the ***isAuthenticated*** and ***isTamperProof*** should also be "True". This is because in implementing ***isNonRepudiationRequired*** by means of a digital signature, both authentication (with respect to the identity associated with the signing certificate) and tamper detection (with respect to the cryptographic hash of the signature) will be implemented as well. The converses need not be true because authentication and tamper detection might be accomplished without archiving information needed to support claims of non-repudiation.

isConfidential

The ***isConfidential*** attribute indicates properties variously distributed among levels of the application-to-application sending/receiving stacks.

isConfidential has possible values of "none", "transient", "persistent", and "transient-and-persistent". The "persistent" or "transient-and-persistent" values indicate that some digital enveloping function is present; a "transient" value indicates that confidentiality is applied at the transfer layer or below.

ebXML *Message* Service Specification, version 2.0[ebMS] does not have an "official" implementation for digital envelopes, and refers to the future XML Encryption specification[XMLENC] as its intended direction for that function. However, the XML Encryption specification is now a candidate recommendation, and is suitable for preliminary implementation.

Within the *CPA*, the ***DocExchange/ebXMLSenderBinding/SenderDigitalEnvelope*** and ***DocExchange/ebXMLReceiverBinding/ReceiverDigitalEnvelope*** can provide configuration details pertaining to security in accordance with [XMLENC]. Use of XML Encryption also will normally show up in the value of ***DigitalEnvelopeProtocol***, and can also appear within a ***NamespaceSupported*** element within ***Packaging.***

Currently, [ebMS] has only indicated a direction to eventually use XML Encryption, but has not mandated any digital envelope protocol. Digital enveloping may be done at the "application level," and will show up under MIME types within the ***Packaging*** element. PKI matching will make use of certificates supplied in ***ApplicationCertificateRef*** and ***ApplicationSecurityDetailsRef***. If other protocols are to be used, it would be safest to use extensions to the content model of ***DocExchange***, such as, ***XXXSenderBinding*** and

XXXReceiverBinding, and follow the pattern of the ebXML content models for ***DocExchange***. Future versions of [ebCPP] intend to make these extension semantics easier to use interoperably; currently, the extensions would be a multilateral extension within some trading community.

When checking whether ***isConfidential*** can be set to "persistent" or "transient-and-persistent" for both *Parties*, check whether the key-exchange certificate will be counted as valid at the sender. The IDREF reference to the ***SecurityDetails*** element is found in ***DocExchange/ebXMLSenderBinding/SenderDigitalEnvelope/EncryptionSecurityDetailsRef/@securityId.*** The trust anchor certificates obtained from ***TrustAnchors/AnchorCertificateRef*** elements under the ***SecurityDetails*** element will be used to test that the certificate referenced by ***DocExchange/ebXMLReceiverBinding/ReceiverDigitalEnvelope/EncryptionCertificateRef/@certId*** validates at the sender side.

As previously noted, failure to validate a certificate does not prevent constructing a draft *CPA*. Either self-signed certificates or new trust anchors can be added to align the trust model on one side with the other side's certificate.

In addition to the PKI-related checks and alignments, the elements ***EncryptionAlgorithm*** and ***DigitalEnvelopeProtocol*** should be checked for equality (or compatibility) and, if not compatible or equal, aligned to values that would work for an initial version of a proposed *CPA*. Preferences and alignment of these elements can be achieved in a subsequent negotiation phase.

Finally, it is possible that one side's ***DigitalEnvelope*** will be modeled using either the ***DocExchange/ebXMLSenderBinding/SenderDigitalEnvelope*** and ***DocExchange/ebXMLReceiverBinding/ReceiverDigitalEnvelope***, while the other side uses only ***Packaging*** to indicate use of, for example, S/MIME Digital Envelopes, because it receives an already enveloped payload from an application. In such a case, the PKI certificate validation check could require checking that a certificate described by ***DocExchange/ebXMLReceiverBinding/ReceiverDigitalEnvelope/EncryptionCertificateRef/@certId*** validates against the ***TrustAnchors*** found by resolving ***CollaborationRole/ApplicationSecurityDetailsRef***. This complication arises from the possibility that digital enveloping functionality can be spread over quite distinct portions of the stack in different software installations.

H.6 *CPA* Formation: Technical Details

When assembling a draft *CPA* from matching portions of two *CPP*s' ***PartyInfo*** elements, some additional constraints need to be observed.

First, as mentioned in section 9.11.1 of [ebCPP], software for producing draft *CPA*s needs to guarantee that ID values in one *CPP* are distinct from ID values in the other *CPP* so that no IDREF references collide when the *CPP*s are merged. The following ID values are potentially subject to collision:

- ***Certificates***
- ***SecurityDetails***
- ***SimplePart***

Packaging
DocExchange
Transport
DeliveryChannel
ThisPartyActionBinding

There are elements and complex type definitions containing IDREFs. Also some elements have attributes with IDREF values. These are:

PartyInfo
ActionBinding.type
ThisPartyActionBinding
OtherPartyActionBinding
OverrideMSHActionBinding
ChannelId
DeliveryChannel
Constituent
CertificateRef.type
AnchorCertificateRef
ApplicationCertificateRef
ClientCertificateRef
ServerCertificateRef
SigningCertificateRef
EncryptionCertificateRef
CertificateRef
SecurityDetailsRef.type

Second, when the ***CanSend*** and ***CanReceive*** binding information has been found to match (equal, correspond with, or be compatible with) the binding information under the other *Party's* ***CanReceive*** and ***CanSend*** elements, the IDREF references for the ***OtherPartyActionBinding*** are filled out in the *CPA*.

Third, for *CPA*s that are signed, the implementer is advised to review section 9.9.1.1 of [ebCPP] when using [XMLDSIG] for the signature technique. A proposed *CPA* need not have a signature.

Fourth, when a *CPA* is composed from two *CPP*s, see section 8.8 of [ebCPP] in which it is stated that all ***Comment*** elements from both *CPP*s SHALL be included in the *CPA* unless agreed to otherwise.

Fifth, several tests on *CPA* validity could be conducted on draft *CPA*s, but these tests are more critical for a negotiated *CPA* that is to be deployed and imported into run-time software components.

1. Expiration: Certificates used in signing a *CPA* can be checked to verify that they do not expire before the *CPA* expires, as given in the ***End*** element.

2. Certificate expiration: If a *CPA* lifetime exceeds the lifetime of certificates accepted for use in signing, key exchange or other security functions, then it would be advisable to make ***ds:KeyInfo*** refer to certificates, rather than to include them within the element by value.

3. Process-Specification references can be checked in accordance with the provisions of section 8.4.4 of [ebCPP] and its subsections.

Finally, a *CPA* has several elements whose values are not typically derived from either *CPP*s (and can need checking when using a *CPA* template as the basis for a draft *CPA*.) The ***Status***, ***Start***, ***End***, and possibly a ***ConversationConstraint***s element need to be added. The attributes,

CollaborationProtocolAgreement/@cpaid,
CollaborationProtocolAgreement/@version,
CollaborationProtocolAgreement/Status@value,
CollaborationProtocolAgreement/ConversationConstrain@invocationLimit, and
CollaborationProtocolAgreement/ConversationConstraint@concurrentConversations,

can also be supplied values as needed.