Collaboration Protocol Profile and Agreement Version 3.0

Committee Specification 01

24 September 2020

This version:
https://docs.oasis-open.org/ebcore/cppa/v3.0/cs01/cppa-v3.0-cs01.odt (Authoritative)
https://docs.oasis-open.org/ebcore/cppa/v3.0/cs01/cppa-v3.0-cs01.html
https://docs.oasis-open.org/ebcore/cppa/v3.0/cs01/cppa-v3.0-cs01.pdf

Previous version:
https://docs.oasis-open.org/ebcore/cppa/v3.0/csprd01/cppa-v3.0-csprd01.odt (Authoritative)
https://docs.oasis-open.org/ebcore/cppa/v3.0/csprd01/cppa-v3.0-csprd01.html
https://docs.oasis-open.org/ebcore/cppa/v3.0/csprd01/cppa-v3.0-csprd01.pdf

Latest version:
https://docs.oasis-open.org/ebcore/cppa/v3.0/cppa-v3.0.odt (Authoritative)
https://docs.oasis-open.org/ebcore/cppa/v3.0/cppa-v3.0.html
https://docs.oasis-open.org/ebcore/cppa/v3.0/cppa-v3.0.pdf

Technical Committee:
OASIS ebXML Core (ebCore) TC

Chairs:
Pim van der Eijk (pvde@sonnenglanz.net), Sonnenglanz Consulting
Sander Fieten (sander@fieten-it.com), Individual member

Editor:
Pim van der Eijk (pvde@sonnenglanz.net), Sonnenglanz Consulting

Additional artifacts:
This prose specification is one component of a Work Product that also includes:

- CPPA3 XML schema: https://docs.oasis-open.org/ebcore/cppa/v3.0/cs01/schema/cppa3.xsd
- Exception XML schema: https://docs.oasis-open.org/ebcore/cppa/v3.0/cs01/schema/exception.xsd
- Schema data dictionaries: https://docs.oasis-open.org/ebcore/cppa/v3.0/cs01/documentation/
- XML document samples: https://docs.oasis-open.org/ebcore/cppa/v3.0/cs01/samples/

Related work:
This specification replaces or supersedes:

- *Collaboration-Protocol Profile and Agreement Specification Version 2.0*. 23 September 2002. OASIS Standard. https://www.oasis-open.org/committees/download.php/204/ebcpp-2.0.pdf.
  Errata: https://www.oasis-open.org/committees/ebxml-cppa/documents/ebCPP-2_0-Errata.php.

This specification is related to:

- *Message Service Specification Version 2.0*. 01 April 2002. OASIS Standard. https://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf.
- *OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features*. OASIS Standard. Latest version: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms_core-3.0-spec.html.
- *OASIS ebXML Messaging Services Version 3.0: Part 2, Advanced Features*. Latest version: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/part2/201004/ebms-v3-part2.html.
- *AS4 Profile of ebMS 3.0 Version 1.0*. OASIS Standard. Latest version: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/profiles/AS4-profile/v1.0/AS4-profile-v1.0.html.

- *ebCore Agreement Update Specification Version 1.0.* Latest version: http://docs.oasis-open.org/ebcore/ebcore-au/v1.0/ebcore-au-v1.0.html.
- *SAML Conformance Clause for AS4/ebMS Version 1.0*. Latest version http://docs.oasis-open.org/ebxml-msg/ebms-v3.0-saml-conformance/v1.0/ebms-v3.0-saml-conformance-v1.0.html.

Declared XML namespace:

- http://docs.oasis-open.org/ebcore/ns/cppa/v3.0

Abstract:

The ebCore Collaboration Protocol Profile and Agreement (CPPA) specification enables efficient and effective configuration and deployment of B2B messaging systems by providing vendor-independent description formats for business and technical capabilities and associated parameters of an individual party (CPP) and for business data exchanges and associated configuration parameter settings that two parties have agreed to (CPA). Furthermore, it specifies default algorithms to automatically derive a CPA document from two parties' CPP documents and to match a CPA against an input CPP and an agreement discovery and registration protocol.

Status:

This document was last revised or approved by the OASIS ebXML Core (ebCore) TC on the above date. The level of approval is also listed above. Check the “Latest version” location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebcore#technical.

TC members should send comments on this specification to the TC’s email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the Technical Committee’s web page at https://www.oasis-open.org/committees/ebcore/.

This specification is provided under the RF on Limited Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this Work Product, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/ebcore/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

When referencing this Work Product the following citation format should be used:

[CPPA3]

Collaboration Protocol Profile and Agreement Version 3.0. Edited by Pim van der Eijk. 24 September 2020. OASIS Committee Specification 01. https://docs.oasis-open.org/ebcore/cppa/v3.0/cs01/cppa-v3.0-cs01.html. Latest version: https://docs.oasis-open.org/ebcore/cppa/v3.0/cppa-v3.0.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

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 section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, 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 OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specification, Candidate OASIS Standard, OASIS Standard, or Approved Errata).

[OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this deliverable.]

[OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this OASIS Standards Final Deliverable. OASIS may include such claims on its website, but disclaims any obligation to do so.]

[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 OASIS Standards Final Deliverable 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' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on 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 implementers or users of this OASIS Standards Final Deliverable, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.]

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

1 Introduction

1.1 Overview

To use B2B messaging protocols and communication networks to exchange documents or data, sender and receiver parties need to configure the communication parameters for their messaging systems and for their networks consistently. These parameters can be grouped in six categories:

- Parameters relating to a Sender, such as its Party Identifier, signing certificate(s), and the IP address (or address ranges) from which it connects and sends messages.
- Parameters relating to a Receiver, such as its Party Identifier, encryption certificate(s), server URL and the server IP address(es) at which it accepts connections and receives messages.
- Parameters relating to the supported business process(es), such as the business process name, version and identifier, party roles, message choreographies of services and their actions and quality of service characteristics of those actions.
- Parameters relating to the messaging protocol(s) used, such as selection of the (versions of) envelope formats to use, security or other quality of service configuration, transmission mode, and error handling.
- Parameters relating to pairs of Sender and Receiver parties, such as agreement identifiers.
- Parameters relating to the data and/or documents being exchanged, such as the (versions of) XML schema to be used for the exchanged documents, any message-level or payload-level properties and the way the data is being packaged.

Note that the same Party MAY be a Sender for some exchanges and a Receiver for other exchanges.

The version 3.0 ebCore Collaboration Protocol Profile and Agreement (CPPA) specification enables efficient and effective configuration and deployment of B2B messaging by providing definitions for:

- Collaboration Protocol Profile (CPP), an electronic document format that describes the business and technical capabilities of an individual party and associated parameters. A CPP covers both sending and receiving capabilities, message channel, transport and networking options, security tokens and trust anchors.
- Collaboration Protocol Agreement (CPA), a similar electronic document format that describes the business and technical exchanges that two Parties have agreed to and the associated configuration parameter settings for those exchanges, such as the selected channel, transport and networking options, security tokens and trust anchors.
- A default method to automatically construct a CPA document from two input CPPs using a method similar to unification, a concept from logic programming.
- A similar default method to automatically determine if a CPA “matches” an input CPP, in the sense of being a potential result of unifying the input CPP with another (possibly unknown) CPP.
- A metadata service location protocol, metadata service protocol and an agreement registration protocol.

This specification provides:

- A high-level overview in section 2 of the CPPA3 schema and the CPP and CPA document types defined in it. This overview provides an introduction to the more detailed documentation of the schema elements and types, which is embedded in the XML schema.
- A specification in section 3 of the default CPA formation algorithm.

- A specification in section 4 of the default CPA-CPP matching algorithm.
- A specification in section 5 of a metadata service location protocol, a metadata service protocol and an agreement registration protocol.
- A definition of CPPA3 conformance in section 6.

This prose specification is complemented by the normative CPPA3 XML schema, the agreement registration Exception XML schema, normative documentation embedded in those schemas and non-normative sample documents.

1.2 Version 3.0

This specification and the associated XML schema constitute version 3 of the ebXML CPPA. The previous version 2 of CPPA [ebCPPA] was approved as an OASIS Standard in 2002 and has been adopted by users and implementers since. The following list summarizes the main improvements and innovations introduced with this newer version of CPPA:

- The CPPA2 schema is complex, due to many levels of nesting and redundancy of structures. This makes editing CPPA2 documents error-prone and CPPA2 documents hard to read and to process. The CPPA3 schema allows the same information to be expressed much more succinctly and clearly. A CPPA3 document is much easier to read and to create and update manually or using automated tooling than a corresponding CPPA2 document. The same information can be expressed in a significantly smaller CPPA3 CPA document than in an CPPA2 CPA document.
- Whereas CPPA2 could only be used to configure the use of version 2 of the ebXML Messaging protocol [ebMS2], this version adds support for the newer ebXML Messaging specifications ebMS3 Core [EBMS3CORE], ebMS3 Advanced Features [EBMS3PART2], the AS4 profile [AS4] and the SAML Conformance Clause for AS4/ebMS [ebMS-saml-conformance].
- This version of CPPA also provides support for configuring EDIINT, including AS1, AS2 and AS3 [RFC3335, RFC4130, RFC4823], Message Disposition Notification [RFC3798] and the newer EDIINT Compression [RFC5402] and Features [RFC6017] specifications.
- This version of CPPA also provides support for the OASIS AMQP standard [amqp-core-messaging-v1.0, amqp-core-transport-v1.0, amqp-core-security-v1.0].
- In addition to enabling fine-grained configuration for messaging protocols, improved and native support is provided for largely pre-configured, heavily profiled use of such protocols in user communities, using “named channels” and “channel profiles”.
- XML Schema extensibility features are used extensively within the schema and also allow the schema to be extended. For example, a derivative of the CPPA3 XSD could add support for a new transport, for an additional messaging protocol, or for a new packaging format that are not natively supported in the current schema.
- CPPA3 enables fully automated formation of CPA documents from CPPs and defines a default formation algorithm that produces consistent and predictable result CPAs for compatible CPPs.
- CPPA3 enables fully automated matching of a CPA document to an input CPP, determining whether or not the CPA validly extends the CPP.
- For ebMS3 and AS4, CPPA3 supports a superset of P-Mode parameters and enables automated generation of P-Mode configuration data for AS4 products.
- CPPA3 supports network access control for IPv4 and IPv6 and exchange of network address information, needed by organizations that have policies that allow communication only from white-listed network addresses or deny communication from black-listed addresses.
- CPPA3 defines an authorization mechanism that provides fine-grained partner self-management to communities of arbitrary size. It can control formation of CPAs from CPPs, matching of CPAs to CPPs, and visibility of CPPs.

- CPPA3 provides a channel delegation feature, allowing parties to express outsourcing of message processing to third parties that act on their behalf.
- CPPA3 adds support for the WebSocket [RFC6455] transport and the SFTP subsystem of SSH2 [RFC4254], complementing the HTTP, SMTP, FTP transports already covered in CPPA2.
- CPPA3 provides specifications for metadata service location, metadata service and agreement registration.

This version 3 provides a superset of functionality of the earlier version 2. There is no backward compatibility in the data format. Software that only implements this version of the specification will not be able to process CPPA2 documents. Software that only implements the earlier CPPA2 will not be able to process CPPA3 documents.

CPPA2 and CPPA3 are structured documents and CPPA3 provides a superset of CPPA2 functionality. It is possible to convert (subsets of) existing CPPA2 documents to corresponding CPPA3 documents automatically or semi-automatically. This may help user communities using CPPA2 migrate to CPPA3.

1.3 Relation to ebXML

This specification is related to, and can be used with, other OASIS ebXML (electronic business XML) standards.

Among the B2B messaging protocols that this version of CPPA supports are the ebMS3 Core specification [EBMS3CORE], the ebMS3 part 2 Advanced Features specification [EBMS3PART2], the AS4 profile [AS4] and the SAML Conformance Clause for AS4/ebMS [ebMS-saml-conformance]. CPP and CPA, and this version of the CPPA specification, are related to the ebMS3 concept of “Processing Mode”, or P-Mode, as follows:

- A CPA records an agreement between two parties as to how messages must be processed, on both the sending and receiving sides. Both MSHs must be able to associate the same P-Mode with a message. The CPA provides this information in a machine-readable format to both parties.
- A single CPA document can configure a set of P-Modes.
- Whereas the P-Mode concept is an abstract concept, for which no open representation format has been standardized, CPP and CPA are non-proprietary XML document formats.
- The CPP and CPA schemas provide extensive support for re-use of parameters or parameter groups.
- This version of CPPA3 fixes 19 issues in the ebMS3 Core and AS4 specification related with the P-Mode concept or definition, listed in section 1.8.

Note that this version of CPPA is not tied to ebMS3 but also supports other messaging protocols. For example, it also supports the older version 2.0 of ebMS [ebMS2] and AS2 RFC4130 and other EDIINT protocols.

To enable parties to find other parties that are suitable business partners, or to enable synchronization of configurations and changes in configurations, CPPs MAY be stored in a repository and MAY be discovered using the capabilities of a registry such as the ebXML Registry [ebRS, ebRIM].

The business process underlying the exchange of messaging between parties MAY be described using the XML format defined in the OASIS ebBP [ebBP] standard. CPP and CPA documents MAY include references to an ebBP document and definitions in that document. That ebBP document MAY itself be stored in a repository such as the ebXML Registry.

1.4 Relation to ebCore Agreement Update

The ebCore Agreement Update specification [ebcore-au-v1.0] defines message exchanges and an XML schema to support the exchange of messaging service communication agreement update requests and

the associated responses to such requests. The specification also provides an Agreement Termination feature. Sections 4.3 and 4.5 of [ebcore-au-v1.0] describe how ebCore Agreement Update could be used with CPPA2 and CPPA3.

The CPPA3 Agreement Registration Service specified in section 5.4 of this specification complements the functionality of ebCore Agreement Update specification by providing a mechanism to deploy new agreements, which are not created as updates of existing agreements.

1.5 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

In this document, the abbreviations “CPPA2” and “CPPA3” are used as a general reference to, respectively, the version 2.0 and 3.0 of ebXML Collaboration Protocol Profiles and Agreements specifications, their XML schemas and XML documents that are valid for those schemas. The abbreviation “CPPA” wil refer to the ebXML Collaboration Protocol Profiles and Agreements specifications irrespective of version.

All notes and examples in this specification, all appendices and sections 1.3, 1.4, 1.8 and 2.3.2 are non-normative. All other text is normative.

1.6 Normative References

[amqp-core-messaging-v1.0] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0 Part 3: Messaging. 29 October 2012. OASIS Standard. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-messaging-v1.0-os.html.

[amqp-core-security-v1.0] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0 Part 5: Security. 29 October 2012. OASIS Standard. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-security-v1.0-os.html.

[amqp-core-transport-v1.0] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0 Part 2: Transport. 29 October 2012. OASIS Standard. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transport-v1.0-os.html.

[AS4-Profile] *AS4 Profile of ebMS 3.0 Version 1.0*. OASIS Standard, 23 January 2013. Edited by J. Durand and P. van der Eijk. http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/profiles/AS4-profile/v1.0/AS4-profile-v1.0.odt.

[BDX-Location-v1.0] Business Document Metadata Service Location Version 1.0. Edited by Dale Moberg and Pim van der Eijk. 01 August 2017. OASIS Standard. http://docs.oasis-open.org/bdxr/BDX-Location/v1.0/os/BDX-Location-v1.0-os.html. Latest version: http://docs.oasis-open.org/bdxr/BDX-Location/v1.0/BDX-Location-v1.0.html.

[ebBP] *ebXML Business Process Specification Schema Technical Specification v2.0.4*. Edited by J.J. Dubray, S. St. Amand, and M. Martin. OASIS Standard, 21 December 2006. http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/spec/ebxmlbp-v2.0.4-Spec-os-en.pdf

[ebCPPA] *Collaboration-Protocol Profile and Agreement Specification Version 2.0*, OASIS Standard, 23 September 2002. http://www.oasis-open.org/committees/download.php/204/ebcpp-2.0.pdf.

[ebMS2] *Message Service Specification*. Version 2.0. OASIS Standard, 1 April 2002. http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf.

[EBMS3CORE] *OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features*. Edited by P. Wenzel. OASIS Standard, 01 October 2007. Latest version: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms_core-3.0-spec.html

[EBMS3PART2] *OASIS ebXML Messaging Services Version 3.0: Part 2, Advanced Features*. Edited by J. Durand, S. Fieten and P. van der Eijk. OASIS Committee Specification 01. 19 May 2011. Latest version:

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/part2/201004/cs01/ebms-v3.0-part2-cs01.odt

[ebMS-saml-conformance] *SAML Conformance Clause for AS4/ebMS Version 1.0*. Edited by Ian Otto. 30 January 2014. OASIS Committee Specification 01. http://docs.oasis-open.org/ebxml-msg/ebms-v3.0-saml-conformance/v1.0/cs01/ebms-v3.0-saml-conformance-v1.0-cs01.html. Latest version: http://docs.oasis-open.org/ebxml-msg/ebms-v3.0-saml-conformance/v1.0/ebms-v3.0-saml-conformance-v1.0.html.

[IANA-HTTP] IANA Hypertext Transfer Protocol (HTTP) Parameters. https://www.iana.org/assignments/http-parameters/http-parameters.xhtml

[REC-XML] Extensible Markup Language (XML) 1.0 (Third Edition), T. Bray, J. Paoli, M. Sperberg-McQueen, E. Maler, F. Yergeau, Editors, W3C Recommendation, February 4, 2004, http://www.w3.org/TR/2004/REC-xml-20040204. Latest version available at http://www.w3.org/TR/REC-xml.

[REC-XML-NAMES] Namespaces in XML, T. Bray, D. Hollander, A. Layman, Editors, W3C Recommendation, January 14, 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114. Latest version available at http://www.w3.org/TR/REC-xml-names.

[RFC0959] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, DOI 10.17487/RFC0959, October 1985, http://www.rfc-editor.org/info/rfc959.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, http://www.rfc-editor.org/info/rfc2045.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, http://www.rfc-editor.org/info/rfc2119**.**

[RFC3335] Harding, T., Drummond, R., and C. Shih, "MIME-based Secure Peer-to-Peer Business Data Interchange over the Internet", RFC 3335, DOI 10.17487/RFC3335, September 2002, http://www.rfc-editor.org/info/rfc3335.

[RFC3798] Hansen, T., Ed., and G. Vaudreuil, Ed., "Message Disposition Notification", RFC 3798, DOI 10.17487/RFC3798, May 2004, http://www.rfc-editor.org/info/rfc3798.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, https://www.rfc-editor.org/info/rfc3986.

[RFC4130] Moberg, D. and R. Drummond, "MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2)", RFC 4130, DOI 10.17487/RFC4130, July 2005, http://www.rfc-editor.org/info/rfc4130.

[RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, https://www.rfc-editor.org/info/rfc4254.

[RFC4422] Melnikov, A., Ed., and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, https://www.rfc-editor.org/info/rfc4422.

[RFC4823] Harding, T. and R. Scott, "FTP Transport for Secure Peer-to-Peer Business Data Interchange over the Internet", RFC 4823, DOI 10.17487/RFC4823, April 2007, http://www.rfc-editor.org/info/rfc4823.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, http://www.rfc-editor.org/info/rfc5321.

[RFC5402] Harding, T., Ed., "Compressed Data within an Internet Electronic Data Interchange (EDI) Message", RFC 5402, DOI 10.17487/RFC5402, February 2010, http://www.rfc-editor.org/info/rfc5402.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, https://www.rfc-editor.org/info/rfc5246.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, http://www.rfc-editor.org/info/rfc5751.

[RFC6017] Meadors, K., Ed., "Electronic Data Interchange - Internet Integration (EDIINT) Features Header Field", RFC 6017, DOI 10.17487/RFC6017, September 2010, http://www.rfc-editor.org/info/rfc6017.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, https://www.rfc-editor.org/info/rfc6066.

[RFC6362] Meadors, K., Ed., "Multiple Attachments for Electronic Data Interchange - Internet Integration (EDIINT)", RFC 6362, DOI 10.17487/RFC6362, August 2011, http://www.rfc-editor.org/info/rfc6362.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, http://www.rfc-editor.org/info/rfc6455.

[RFC6931] Eastlake 3rd, D., "Additional XML Security Uniform Resource Identifiers (URIs)", RFC 6931, DOI 10.17487/RFC6931, April 2013, http://www.rfc-editor.org/info/rfc6931.

[RFC7230] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, http://www.rfc-editor.org/info/rfc7230.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, https://www.rfc-editor.org/info/rfc7234.

[RFC7235] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, https://www.rfc-editor.org/info/rfc7235.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, http://www.rfc-editor.org/info/rfc7540.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, https://www.rfc-editor.org/info/rfc8446.

[SAML-CORE-2.0-OS] *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*. Edited by S. Cantor et. al. OASIS Standard, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[SOAP11] Simple Object Access Protocol (SOAP) 1.1. Edited by D. Box, et al. W3C Note, 8 May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[SOAP12-MTOM] SOAP Message Transmission Optimization Mechanism, M. Gudgin, N. Mendelsohn, M. Nottingham, H. Ruellan, Editors, W3C Recommendation, January 25, 2005, http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/. Latest version available at http://www.w3.org/TR/soap12-mtom.

[SOAP12-PART1] SOAP Version 1.2 Part 1: Messaging Framework, M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. Frystyk Nielsen, Editors, W3C Recommendation, June 24, 2003, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/. Latest version available at http://www.w3.org/TR/soap12-part1/.

[SOAPATTACH] SOAP Messages with Attachments. Edited by J. Barton, et al. W3C Note, 2000. http://www.w3.org/TR/SOAP-attachments

[WSSecurityPolicy13] *WS-SecurityPolicy 1.3*. Edited by A. Nadalin et al. OASIS Standard, February 2009. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.doc

[WSS-SAML-Token-Profile-V1.1.1] *Web Services Security SAML Token Profile Version 1.1.1*. Edited by R. Monzillo et al. OASIS Standard, May 2012. http://docs.oasis-open.org/wss-m/wss/v1.1.1/os/wss-SAMLTokenProfile-v1.1.1-os.pdf

[WSSSMS] *OASIS Web Services Security: SOAP Message Security Version 1.1.1.* Edited by A. Nadalin, C. Kaler, R. Monzillo, P. Hallam-Baker and C. Milono. OASIS Standard, May 2012. http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-SOAPMessageSecurity-v1.1.1.doc

[WSSSWA] *OASIS Web Services Security: Web Services Security SOAP Message with Attachments (SwA) Profile Version 1.1.1*. Edited by F. Hirsch and C. Milono. OASIS Standard, May 2012. http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-SwAProfile-v1.1.1.doc

[WSSUsername] *Web Services Security Username Token Profile Version 1.1.1*. 18 May 2012. OASIS Standard. http://docs.oasis-open.org/wss-m/wss/v1.1.1/os/wss-UsernameTokenProfile-v1.1.1-os.html.

[WSSX509] *OASIS Web Services Security: Web Services Security X.509 Certificate Token Profile Version 1.1.1*. Edited by A. Nadalin, C. Kaler, R. Monzillo, P. Hallam-Baker and C. Milono. OASIS Standard, May 2012. http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-x509TokenProfile-v1.1.1.doc

[XKMS2] XML Key Management Specification (XKMS 2.0), P. Hallam-Baker, S. Mysore, Editors, W3C Recommendation, June 28, 2005, http://www.w3.org/TR/2005/REC-xkms2-20050628/. Latest version available at http://www.w3.org/TR/xkms2/.

[XML-C14N] Canonical XML Version 1.0, J. Boyer, Editor, W3C Recommendation, March 15, 2001, http://www.w3.org/TR/2001/REC-xml-c14n-20010315. Latest version available at http://www.w3.org/TR/xml-c14n.

[XMLDSIG-CORE] XML Signature Syntax and Processing (Second Edition), D. Eastlake, J. Reagle, D. Solo, F. Hirsch, T. Roessler, Editors, W3C Recommendation, June 10, 2008, http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/. Latest version available at http://www.w3.org/TR/xmldsig-core/.

[XMLDSIG-CORE1] XML Signature Syntax and Processing Version 1.1, D. Eastlake, J. Reagle, D. Solo, F. Hirsch, M. Nyström, T. Roessler, K. Yiu, Editors, W3C Recommendation, April 11, 2013, http://www.w3.org/TR/2013/REC-xmldsig-core1-20130411/. Latest version available at http://www.w3.org/TR/xmldsig-core1/.

[XMLENC-CORE] XML Encryption Syntax and Processing. W3C Recommendation 10 December 2002. Latest version: http://www.w3.org/TR/xmlenc-core/

[XMLENC-CORE1] XML Encryption Syntax and Processing Version 1.1, D. Eastlake, J. Reagle, F. Hirsch, T. Roessler, Editors, W3C Recommendation, April 11, 2013, http://www.w3.org/TR/2013/REC-xmlenc-core1-20130411/. Latest version available at http://www.w3.org/TR/xmlenc-core1/.

[XMLSCHEMA11-2] W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, D. Peterson, S. Gao, A. Malhotra, M. Sperberg-McQueen, H. Thompson, Paul V. Biron, Editors, W3C Recommendation, April 5, 2012, http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/. Latest version available at http://www.w3.org/TR/xmlschema11-2/.

[XOP] XML-binary Optimized Packaging, M. Gudgin, N. Mendelsohn, M. Nottingham, H. Ruellan, Editors, W3C Recommendation, January 25, 2005, http://www.w3.org/TR/2005/REC-xop10-20050125/. Latest version available at http://www.w3.org/TR/xop10/.

1.7 Non-Normative References

[as2-restart] T. Harding. “AS2 Restart for Very Large Messages”. Internet Draft, June, 2014. https://tools.ietf.org/html/draft-harding-as2-restart-07

[CIQ3] *Customer Information Quality Specifications Version 3.0 Name (xNL), Address (xAL), Name and Address (xNAL) and Party (xPIL)* OASIS Committee Specification 02. 20 September 2008. http://docs.oasis-open.org/ciq/v3.0/cs02/

[ebcore-au-v1.0] *ebCore Agreement Update Specification Version 1.0*. Edited by P. van der Eijk and T. Kramer. 19 September 2016. OASIS Committee Specification. Latest version: http://docs.oasis-open.org/ebcore/ebcore-au/v1.0/ebcore-au-v1.0.html.

[ebRIM] *OASIS ebXML RegRep Version 4.0. Part 1: Registry Information Model (ebRIM).* Edited by F. Najmi and N. Stojanovic. OASIS Standard, January 2012. http://docs.oasis-open.org/regrep/regrep-core/v4.0/os/regrep-core-rim-v4.0-os.odt

[ebRS] *OASIS ebXML RegRep Version 4.0 . Part 2: Services and Protocols. (ebRS)*. Edited by F. Najmi and N. Stojanovic. OASIS Standard, January 2012. http://docs.oasis-open.org/regrep/regrep-core/v4.0/os/regrep-core-rs-v4.0-os.odt

[ediint-fn] T. Harding. “Filename Preservation for EDIINT”. Internet Draft, March 2010. https://tools.ietf.org/html/draft-harding-ediint-filename-preservation-03

[ENTSOGAS4] ENTSOG AS4 Usage Profile for TSOs. https://entsog.eu/publications/common-data-exchange-solutions#AS4-USAGE-PROFILE

[PARTYIDTYPE] *ebCore Party Id Type Technical Specification Version 1.0,* April 2010. Edited by D. Moberg and P. van der Eijk. OASIS Committee Specification. Latest Version: http://docs.oasis-open.org/ebcore/PartyIdType/v1.0/PartyIdType-1.0.odt

[SBDH] UN/CEFACT Standard Business Document Header. Version 1.3. 2004-6-04. http://www.gs1.org/gs1-uncefact-xml-profiles-edi-xml-gdsn/sbdh-technical-specifications/1-3.

1.8 Issue References

The following lists of issues in ebMS3 specifications [EBMS3CORE, AS4] are addressed in this version of CPPA.

[EBXMLMSG12] D 3.6 D.3.6. PMode[1].Security.Sign.Attachment. https://issues.oasis-open.org/browse/EBXMLMSG-12

[EBXMLMSG13] D.3.6. and 5.2.2.12, external payloads. https://issues.oasis-open.org/browse/EBXMLMSG-13

[EBXMLMSG15] D3.4 Pmodes for error handling. https://issues.oasis-open.org/browse/EBXMLMSG-15

[EBXMLMSG32] Pmode parameter for reliable messaging protocol missing. https://issues.oasis-open.org/browse/EBXMLMSG-32

[EBXMLMSG33] Pmode parameter to express destination URL for asynchronous receipts. https://issues.oasis-open.org/browse/EBXMLMSG-33

[EBXMLMSG45] PMode parameter for Key Transport algorithm. https://issues.oasis-open.org/browse/EBXMLMSG-45

[EBXMLMSG49] Mask Generation Function. https://issues.oasis-open.org/browse/EBXMLMSG-49

[EBXMLMSG62] D.3.3 PMode[1].BusinessInfo.PayloadProfile[] https://issues.oasis-open.org/browse/EBXMLMSG-62

[EBXMLMSG63] D.3.3 PMode[1].BusinessInfo.PayloadProfile[], part cardinality. https://issues.oasis-open.org/browse/EBXMLMSG-63

[EBXMLMSG64] D.3.3 PMode[1].BusinessInfo.PayloadProfile[], schema name. https://issues.oasis-open.org/browse/EBXMLMSG-64

[EBXMLMSG69] P-Mode parameter missing for type of X509 Token Reference.
https://issues.oasis-open.org/browse/EBXMLMSG-69

[EBXMLMSG73] PMode[1].BusinessInfo.Service.
https://issues.oasis-open.org/browse/EBXMLMSG-73

[EBXMLMSG74] PMode[2].BusinessInfo.Action.
https://issues.oasis-open.org/browse/EBXMLMSG-74

[EBXMLMSG75] Property CompressionType is reserved.
https://issues.oasis-open.org/browse/EBXMLMSG-75

[EBXMLMSG80] Missing Digest, Nonce and Created in Pmode.*.Authorization.
https://issues.oasis-open.org/browse/EBXMLMSG-80

[EBXMLMSG89] D.2.1 PMode Notation. https://issues.oasis-open.org/browse/EBXMLMSG-89

[EBXMLMSG97] 7.11.2 X.509 tokens in Pull requests targeted to default role. https://issues.oasis-open.org/browse/EBXMLMSG-97

[EBXMLMSG98] AS4 5.2.3, securing pull requests.
https://issues.oasis-open.org/browse/EBXMLMSG-98

[EBXMLMSG105] *.Initiator.* and *.Responder.* in Pmode.
https://issues.oasis-open.org/browse/EBXMLMSG-105

[EBXMLMSG110] Overlap of PMode[1].ErrorHandling.Report.ProcessErrorNotifyProducer and PMode[1].ErrorHandling.Report.MissingReceiptNotifyProducer.
https://issues.oasis-open.org/browse/EBXMLMSG-110

1.9 Namespaces

In this specification, in XML examples, the following prefixes will be used for the specified namespaces.

Prefix	Namespace	Specification
cppa	http://docs.oasis-open.org/ebcore/ns/cppa/v3.0	This document and the CPPA3 XML schema
ds	http://www.w3.org/2000/09/xmldsig#	[XMLDSIG-CORE,XMLDSIG-CORE1]
ebbp	http://docs.oasis-open.org/ebxml-bp/ebbp-2.0	[ebBP]
exc	http://docs.oasis-open.org/ebcore/ns/exception/v3.0	This document and the Exception XML schema
eb3	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/	[EBMS3CORE]
xkms	http://www.w3.org/2002/03/xkms#	[XKMS]
xml	http://www.w3.org/XML/1998/namespace	[REC-XML]

Table 1: XML Namespaces

2 CPP and CPA

2.1 Profiles and Agreements

The CPPA3 specification is based on the concepts of collaboration protocol profiles and agreements.

- Collaboration Protocol Profile (CPP) is an electronic document format that describes the business and technical capabilities of an individual party and associated parameters. A CPP covers party information, sending and receiving capabilities, message channel, transport and networking options, security tokens and trust anchors.
- Collaboration Protocol Agreement (CPA) is a similar electronic document format that describes the business and technical exchanges that two Parties have agreed to and the associated configuration parameter settings for those exchanges, such as the selected channel, transport and networking options, security tokens and trust anchors.

An overview of the CPP and CPA schemas is provided in section 2.2. For the CPP and CPA schema and its detailed documentation, see section 2.3.

Two CPPs for two parties that can act in complementary business roles and have matching technical capabilities can be merged into a CPA that describes an agreed messaging configuration for the two parties using the combined data from the CPPs. Section 3 defines the CPPA3 default unification algorithm.

This specification does not assume that formation of a CPA is always, or in specific situations, a prerequisite for communication. Some messaging protocols or profiles, including but not limited to the ebMS3 SAML profile [ebMS-saml-conformance], are explicitly intended to enable communication without any prior agreement. For such protocols, a partner CPP provides all information needed for communication and formation of a CPA is not required.

2.2 Structures

2.2.1 Overview

The CPP and CPA document structures are very similar and contain elements for the following types of information:

- Document Metadata
- Party Information
- Service Specifications
- Service Bindings
- Property Sets
- Channels
- Channel Features
- Transports
- Payload Profiles
- Packaging
- PartyIdList

- Signatures

This section provides an overview of these elements. The full documentation is provided in the CPP and CPA schema documentation, which is introduced in section 2.3. Note that the `PartyIdList` element is only used in a CPP.

2.2.2 Document Metadata

A CPP contains a mandatory `ProfileInfo` element that provides metadata for the profile. The provided information includes the following elements:

- A `ProfileIdentifier` that identifies the profile.
- A natural language `Description` of the profile.
- An `ActivationDate` that specifies from which date and time the profile is valid.
- An `ExpirationDate` that specifies until which date and time the profile is valid.
- A `PhaseIn` interval that specifies the minimum amount time needed to phase in a new agreement based on the profile.

Except for `ProfileIdentifier`, all these elements are optional. In CPPs that involve Channel elements that use X.509 certificates, the `ActivationDate` and `ExpirationDate` MUST be consistent with the validity intervals of those certificates.

The following example shows a sample `ProfileInfo` structure from a CPP.

```
<cppa:ProfileInfo>
    <cppa:ProfileIdentifier>P4_cpp_named_channels</cppa:ProfileIdentifier>
    <cppa:Description xml:lang="en">Collaboration Protocol Profile for Party P4
      </cppa:Description>
    <cppa:ActivationDate>2016-01-01T00:00:00</cppa:ActivationDate>
    <cppa:ExpirationDate>2018-01-01T00:00:00</cppa:ExpirationDate>
    <cppa:PhaseIn>P7D</cppa:PhaseIn>
</cppa:ProfileInfo>
```

A CPA contains a mandatory `AgreementInfo` element that provides metadata about the agreement. Section 3.4.6 describes how an `AgreementInfo` element can be formed from two input `ProfileInfo` elements.

The following example shows a sample `AgreementInfo` structure from a CPA .

```
<cppa:AgreementInfo>
  <cppa:AgreementIdentifier>P1_cpp_P4_cpp_named_channels</cppa:AgreementIdentifier>
  <cppa:Description xml:lang="en">Agreement formed from P1_cpp_named_channels and
      P4_cpp_named_channels at 2016-10-02T13:25:40.762290</cppa:Description>
  <cppa:ProfileIdentifier>P1_cpp_named_channels</cppa:ProfileIdentifier>
  <cppa:ProfileIdentifier>P4_cpp_named_channels</cppa:ProfileIdentifier>
  <cppa:ActivationDate>2016-10-09T13:25:40.762322</cppa:ActivationDate>
  <cppa:ExpirationDate>2018-01-01T00:00:00</cppa:ExpirationDate>
</cppa:AgreementInfo>
```

2.2.3 Party Information

A CPP contains a `PartyInfo` element that provides information about the party. A CPA contains a `PartyInfo` element and a `CounterPartyInfo` element to provide information about the two parties. The provided information includes:

- Party Identifier values and types.
- Party Name.
- Contact Information.

- Optionally, certificates used in `Channel` or `Transport` elements. In CPPA3, X.509 certificates can be included, encoded as XML Signature `KeyInfo` elements. Another option is to select certificates using XKMS `LocateRequest` elements.
- Optionally, a `CertificateDefaults` element that specifies default certificates for particular certificate usage.
- Optionally, lists of trusted root certificates issued by trusted Certification Authorities. The `TrustAnchorSet` element contains a list of `Certificate` elements or references to `Certificate` elements.
- Optionally, lists of `CertificatePolicySet` elements that specify sets of X.509 certificate policies.
- Optionally, lists of `IDPRegistration` and `IDPRegistrationSet` elements that configure registrations with identity provider services.

Section 3.4.7 specifies how CPP `PartyInfo` elements are processed in CPA formation.

The following example contains a sample `PartyInfo` element. For readability all content other than `KeyName` is removed from the example `KeyInfo` structures.

```
<cppa:PartyInfo>
    <cppa:PartyName xml:lang="en">Party P1</cppa:PartyName>
    <cppa:PartyId
        type="urn:oasis:names:tc:ebcore:partyid-type:unregistered">P1</cppa:PartyId>
    <cppa:PartyContact>
        <cppa:Email>partnersupport@p1.example.com</cppa:Email>
    </cppa:PartyContact>
    <cppa:Certificate id="a_signing_cert">
        <ds:KeyInfo>
            <ds:KeyName>Signing Certificate for
                urn:oasis:names:tc:ebcore:partyid-type:unregistered:P1</ds:KeyName>
        </ds:KeyInfo>
    </cppa:Certificate>
    <cppa:Certificate id="a_encryption_cert">
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            xmlns:dsig11="http://www.w3.org/2009/xmldsig11#">
            <ds:KeyName>Encryption Certificate for
                urn:oasis:names:tc:ebcore:partyid-type:unregistered:P1</ds:KeyName>
        </ds:KeyInfo>
    </cppa:Certificate>
    <!-- A CA that P1 trusts. -->
    <cppa:Certificate id="ca_a">
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            xmlns:dsig11="http://www.w3.org/2009/xmldsig11#">
            <ds:KeyName>Certificate Signing
               urn:oasis:names:tc:ebcore:partyid-type:unregistered:A_CA</ds:KeyName>
        </ds:KeyInfo>
    </cppa:Certificate>
    <!-- Another CA that P1 trusts -->
    <cppa:Certificate id="ca_b">
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            xmlns:dsig11="http://www.w3.org/2009/xmldsig11#">
            <ds:KeyName>Certificate Signing
               urn:oasis:names:tc:ebcore:partyid-type:unregistered:B_CA</ds:KeyName>
        </ds:KeyInfo>
    </cppa:Certificate>
    <cppa:TrustAnchorSet id="trust_anchor_for_p1">
        <cppa:AnchorCertificateRef certId="ca_a"/>
        <cppa:AnchorCertificateRef certId="ca_b"/>
    </cppa:TrustAnchorSet>
</cppa:PartyInfo>
```

2.2.4 Service Specification

A `ServiceSpecification` provides information about service interactions between a Party acting in the `PartyRole` role and another Party acting in the `CounterPartyRole`. A `ServiceSpecification` MAY be annotated with the `uuid`, `name` and `version` attributes defined in [ebBP].

In a CPP:

- The `PartyRole` role relates to the Party described in the `PartyInfo` element.
- The `CounterPartyRole` role relates to an unspecified potential other Party with whom Party MAY interact.

In a CPA:

- The `PartyRole` role relates to the Party described in the `PartyInfo` element.
- The `CounterPartyRole` role relates to the Party described in the `CounterPartyInfo` element.

Section 3.4.8 describes how two `ServiceSpecification` elements for a corresponding role pair in two CPPs can be processed to produce a CPA `ServiceSpecification` element for that pair of roles.

2.2.5 Service Binding

A `ServiceSpecification` contains one or more `ServiceBinding` elements that provide bindings for service interactions between the two specified roles to a specific `Service`.

A given `Service` is associated with one or more `ActionBinding` elements that specify the specific business exchanges among Parties in the context of the `Service`. In an `ActionBinding`, the `action` attribute names the exchange. The directionality of the exchange is encoded in the `sendOrReceive` attribute.

- If the value is "*send*", the exchange is from the `PartyRole` role Party to the `CounterPartyRole` role Party.
- If the value is "*receive*", the exchange is from the `CounterPartyRole` Party to the `PartyRole` Party.

Note that whether the `PartyRole` role Party initiates the message exchange is not just dependent on the `sendOrReceive` attribute value, but also on the channel binding.

The following CPP example shows a `ServiceSpecification` for a "*Seller*", "*Buyer*" pair that contains two `ServiceBinding` elements, one for an "*OrderOnly*" `Service` and another for an "*Ordering*" `Service`. The first of these contains one receiving action, named "*SubmitOrder*". The second contains three actions, named "*SubmitOrder*", "*AcceptOrder*" and "*RejectOrder*".

```
<cppa:ServiceSpecification>
    <cppa:PartyRole name="Seller"/>
    <cppa:CounterPartyRole name="Buyer"/>
    <cppa:ServiceBinding>
        <cppa:Description xml:lang="en">A simple
             e-Commerce ordering service</cppa:Description>
        <cppa:Service>OrderOnly</cppa:Service>
        <cppa:ActionBinding id="BII03_001_R" action="SubmitOrder"
            sendOrReceive="receive" payloadProfileId="pp1">
            <cppa:ChannelId>ch3</cppa:ChannelId>
            <cppa:ChannelId>ch4</cppa:ChannelId>
        </cppa:ActionBinding>
    </cppa:ServiceBinding>
    <cppa:ServiceBinding>
        <cppa:Description xml:lang="en">A slightly more sophisticed
             e-Commerce ordering service</cppa:Description>
        <cppa:Service>Ordering</cppa:Service>
```

```
        <cppa:ActionBinding id="BII28_001_R" action="SubmitOrder"
            sendOrReceive="receive" payloadProfileId="pp1">
            <cppa:ChannelId>ch3</cppa:ChannelId>
            <cppa:ChannelId>ch4</cppa:ChannelId>
        </cppa:ActionBinding>
        <cppa:ActionBinding id="BII28_002_S" action="AcceptOrder"
            sendOrReceive="send" payloadProfileId="pp2">
            <cppa:ChannelId>ch1</cppa:ChannelId>
            <cppa:ChannelId>ch2</cppa:ChannelId>
        </cppa:ActionBinding>
        <cppa:ActionBinding id="BII28_003_S" action="RejectOrder"
            sendOrReceive="send" payloadProfileId="pp2">
            <cppa:ChannelId>ch1</cppa:ChannelId>
            <cppa:ChannelId>ch2</cppa:ChannelId>
        </cppa:ActionBinding>
    </cppa:ServiceBinding>
</cppa:ServiceSpecification>
```

An `ActionBinding` also provides bindings:

- To one or, in a CPP, multiple alternative messaging `Channel` definitions, using the `ChannelId` sub-element.
- To an optional payload profile, using the `payloadProfileId` attribute. This attribute enables payload validation by the receiving message handler.

In the shown example, the `ActionBinding` for "*SubmitOrder*" in "*OrderOnly*" is bound to an ordered list of cross-references to the channels "*ch3*" and "*ch4*".

To associate metadata with the action, an `ActionBinding` MAY specify one or more `Property` elements, or carry a `propertySetId` attribute that references a `PropertySet` element.

Section 3.4.9 describes how corresponding `ServiceBinding` elements in two CPPs can be processed to produce a CPA `ServiceBinding` element. Section 3.4.10 does this for `ActionBinding` elements.

2.2.6 PropertySet

A `PropertySet` element is a reusable container for a set of `Property` elements. It can be referenced from `ActionBinding` elements and facilitates reuse of property sets. The following example specifies a set containing two mandatory properties and one optional property.

```
<cppa:PropertySet id="four_corner">
    <cppa:Property name="finalRecipient" minOccurs="1" maxOccurs="1"/>
    <cppa:Property name="originalSender" minOccurs="1" maxOccurs="1"/>
    <cppa:Property name="trackingIdentifier" minOccurs="0" maxOccurs="1"/>
</cppa:PropertySet>
```

2.2.7 Channels

CPPA3 defines three types of channel definitions.

1. The first type of `Channel` configures a one-directional message exchange from one Message Service Handler to another Message Service Handler using a messaging protocol. The CPPA3 schema (see section 2.3) provides built-in, fine-grained support for a number of messaging protocols.
2. The `NamedChannel` mechanism allow parties to reference their own protocols or Usage Profiles using mutually agreed channel names. The `ChannelProfile` feature can be used to profile channels.
3. The `DelegationChannel` (see section 2.2.8) expresses messaging service delegation to third parties.

CPPA3 `ActionBinding` elements use their `ChannelId` child elements to specify channels to use for the exchange of business data. Those referenced `Channel` definitions MAY in turn refer to other channels to be used in conjunction with them. For example, a `Channel` MAY depend on another `Channel` to create a back-channel for it, or specify another `Channel` that is to be used to carry receipts for messages exchanged on it.

The following example shows a `NamedChannel` that can be bound to actions in service bindings. It covers an inbound channel, and specifies the trust anchor that the certificate of the counter party must chain to and the encryption certificate that the counter party must use to encrypt the message. The channel name is a symbolic reference to a particular type of channel, such as a usage profile. This reference must be shared and understood by the parties to be used. The fictitious value "*community-as4-profile.v1.0*" in the example can be thought of as a short hand for a definition of an AS4 community profile that has fixed values for various AS4 parameters.

```
<cppa:NamedChannel id="ch3" transport="tr3">
    <cppa:Description xml:lang="en">Current protocol, receiving mode
    </cppa:Description>
    <cppa:ChannelName>community-as4-profile.v1.0</cppa:ChannelName>
    <cppa:SigningTrustAnchorSetRef certId="trust_anchor_for_p1"/>
    <cppa:EncryptionCertificateRef certId="a_encryption_cert"/>
</cppa:NamedChannel>
```

Instead of a `NamedChannel`, a second use for `Channel` elements is to specify a channel that uses and configures a messaging protocol defined in the CPPA3 schema or in an extension of it. The following example provides an example of a complete `ebMS3Channel` element. It has sub-elements that configure WS-Security, AS4 Reception Awareness and Receipt and Error Handling.

```
<cppa:ebMS3Channel id="_52UD" transport="_BCXV" package="_ZHDC">
  <cppa:Description xml:lang="en">Channel formed from ch3 (Channel used to receive
   ebMS3 business messages) in P1_cpp and ch1 (Channel used to send UserMessages)
   in P4_cpp</cppa:Description>
  <cppa:SOAPVersion>1.2</cppa:SOAPVersion>
  <cppa:WSSecurityBinding>
    <cppa:WSSVersion>1.1</cppa:WSSVersion>
    <cppa:Signature>
      <cppa:SignatureAlgorithm>http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
         </cppa:SignatureAlgorithm>
      <cppa:DigestAlgorithm>http://www.w3.org/2001/04/xmlenc#sha256
         </cppa:DigestAlgorithm>
      <cppa:SigningCertificateRef certId="p4_signingcert"/>
    </cppa:Signature>
    <cppa:Encryption>
      <cppa:KeyEncryption>
        <cppa:EncryptionAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
         </cppa:EncryptionAlgorithm>
      </cppa:KeyEncryption>
      <cppa:EncryptionAlgorithm>http://www.w3.org/2009/xmlenc11#aes128-gcm
         </cppa:EncryptionAlgorithm>
      <cppa:EncryptionCertificateRef certId="a_encryption_cert"/>
    </cppa:Encryption>
  </cppa:WSSecurityBinding>
  <cppa:AS4ReceptionAwareness>
    <cppa:DuplicateHandling>
      <cppa:DuplicateElimination>true</cppa:DuplicateElimination>
    </cppa:DuplicateHandling>
    <cppa:RetryHandling>
      <cppa:Retries>2</cppa:Retries>
      <cppa:RetryInterval>PT2H</cppa:RetryInterval>
    </cppa:RetryHandling>
  </cppa:AS4ReceptionAwareness>
  <cppa:ErrorHandling>
    <cppa:DeliveryFailuresNotifyProducer>true</cppa:DeliveryFailuresNotifyProducer>
    <cppa:ProcessErrorNotifyConsumer>false</cppa:ProcessErrorNotifyConsumer>
    <cppa:ProcessErrorNotifyProducer>true</cppa:ProcessErrorNotifyProducer>
    <cppa:ReceiverErrorsReportChannelId>_XBNL</cppa:ReceiverErrorsReportChannelId>
  </cppa:ErrorHandling>
  <cppa:ReceiptHandling>
    <cppa:ReceiptChannelId>_XBNL</cppa:ReceiptChannelId>
```



```
  </cppa:ReceiptHandling>
</cppa:ebMS3Channel>
```

Note that this example channel has cross-references to another channel with identifier "*_XBNL*" that is used for reporting errors and confirming receipt of the message. That channel (not itself shown in the example) is also an `ebMS3Channel` and its security and other features can be specified for that channel separately.

All CPPA3 channels for messaging protocols have an optional `ChannelProfile` element that MAY be used to reference a profile of the channel. Like a `NamedChannel`, it avoids redundantly specifying values that are set by a profile but unlike a `NamedChannel` this element allows the profile values to be overridden. In the following example, this is used to specify support for two specific encryption algorithms, overriding any values that may be specified for the specified channel profile.

```
<cppa:ebMS3Channel id="b_ch_receive" transport="tr_receive" package="entsog_package">
  <cppa:Description xml:lang="en">Channel for incoming ENTSOG AS4 User
              Messages</cppa:Description>
  <cppa:ChannelProfile>http://www.entsog.eu/publications/as4#AS4-USAGE-PROFILE/v2.0/
UserMessageChannel</cppa:ChannelProfile>
  <cppa:WSSecurityBinding>
    <cppa:Encryption>
      <cppa:EncryptionAlgorithm>http://www.w3.org/2009/xmlenc11#aes128-gcm
            </cppa:EncryptionAlgorithm>
      <cppa:EncryptionAlgorithm>http://www.w3.org/2001/04/xmlenc#aes256-gcm
            </cppa:EncryptionAlgorithm>
      <cppa:EncryptionCertificateRef certId="_4UP74O"/>
    </cppa:Encryption>
  </cppa:WSSecurityBinding>
</cppa:ebMS3Channel>
```

A `Channel` MAY reference a specific `Transport` using its `transport` attribute. The example references the "*_BCXV*" transport. The allowed transports usually are channel-dependent. For example, the AS1, AS2 and AS3 channels MUST reference SMTP, HTTP and FTP transports, respectively and an ebMS 2.0 channel MUST reference an SMTP or HTTP transport. The CPPA3 schema documentation (referenced in section 2.3) explains in which situations a channel is and is not required to reference a transport.

A `Channel` MAY reference a specific `Package` using its `package` attribute. The example references the "*_ZHDC*" package. The degree to which packaging can (or needs to) be configured varies across messaging protocols and implementations.

Section 3.4.11 describes how corresponding `Channel` elements in two CPPs can be processed to produce a CPA `Channel` element.

The CPPA3 schema provides an abstract `Channel` element of type `ChannelType` from which the channel elements inherit. Extensions of CPPA3 can also use this mechanism to define additional channel elements.

2.2.8 Delegation

A third type of `Channel` in CPPA3 is the `DelegationChannel`. Delegation is a feature by which a Party expresses that the sending or receiving of messages for an action is performed by a third party, rather than by the Party itself. This MAY be used to express situations where the third Party performs actual outsourced business services for the delegating Party, but this is not REQUIRED. The delegation MAY also be limited to messaging only. In that case, there will need to be a separate and separately configured exchange of data between the delegating and delegated parties. The CPPA3 concept of `DelegationChannel` does not have any particular business or legal semantics.

In a CPP or CPA, the `PartyId` in a `DelegationChannel` expresses delegation from the Party in `PartyInfo`. In a CPA, delegation from the other Party in `CounterPartyInfo` is expressed using a `CounterPartyId` element. The following fragment from a CPA for `Party` A and `CounterParty` B shows a delegation from both parties, i.e. neither Party is directly involved in the configured exchange.

```
<cppa:DelegationChannel id="_QRIN">
  <cppa:PartyId
   type="urn:oasis:names:tc:ebcore:partyid-type:unregistered">C</cppa:PartyId>
  <cppa:CounterPartyId
   type="urn:oasis:names:tc:ebcore:partyid-type:unregistered">D</cppa:CounterPartyId>
</cppa:DelegationChannel>
```

Receivers SHOULD use delegation information expressed in `DelegationChannel` elements to determine whether third Party Senders are actually authorized to send business data on behalf of another Party. Senders SHOULD use the information to route messages.

Note that the configuration of the exchanges involving delegated Parties is not specified in the CPPA3 document that contains the `DelegationChannel`. That configuration MAY be specified in separate CPPA3 documents, but this is NOT REQUIRED.

2.2.9 Channel Features

Channels provide features for functionality such as security, reliable messaging, receipt handling and error handling. The CPPA3 schema (see section 2.3) provides document structures to configure these features. These structures MAY occur in two places:

- As child element of the channel definition element that uses it. In this case, the `id` attribute MUST NOT be present and the definition can not be reused.
- As child element of the `CPP` or `CPA` element. In this case, the `id` attribute MUST be provided. Channel definitions that use the feature do so by referencing it using its `id` value. This cross-reference mechanism allows multiple definitions to share a feature definition.

The security features allow:

- references to specific certificates to be used for specific purposes.
- references to specific trust anchors, to define constraints on the certificates of the other Party.

See section 3.4.20 for discussion on how these references are used in CPA formation.

As an example, the `ebMS3Channel` specified in the example in section 2.2.7 can be specified alternatively with reference to separate `ChannelFeature` elements, as follows:

```
<cppa:ebMS3Channel id="ch3" transport="tr3" package="swa1"
   securityBinding="wss_receiving_usermessage" reliableMessagingBinding="asrm_recv"
   errorHandling="eh_rcv" receiptHandling="rh_recv_um">
   <cppa:Description xml:lang="en">Channel used to receive ebMS3
         business messages</cppa:Description>
   <cppa:SOAPVersion>1.2</cppa:SOAPVersion>
</cppa:ebMS3Channel>
```

This example assumes a `WSSecurityBinding` channel feature element is provided in the document that has the "*wss_receiving_usermessage*" identifier. Similarly, features for reliable messaging, error and receipt handling are referenced.

The CPPA3 schema provides an abstract `ChannelFeature` element of type `ChannelFeatureType` from which the channel feature elements inherit. Extensions of CPPA3 can also use this mechanism to define additional channel features.

2.2.10 Transports

A `Channel` can be bound to a specific `Transport`. For TCP transports, endpoint URIs and security can be configured. Transport security mechanisms include:

- Username and Password based authentication.
- X.509 certificate-based client and server authentication.

- TLS configuration, including version and cipher suites.
- IPv4 and IPv6 address (range) specification, enabling network security configuration using firewalls.

Like `Channel` elements, `Transport` elements MAY reference specific certificates and trust anchors to match against in CPA formation. Using the optional `supportsIPv4` and `supportsIPv6` attributes, a `Transport` MAY indicate the Internet Protocol version supported by the `Transport`. The CPPA3 schema provides an abstract `Transport` element of type `TransportType` from which the transport elements inherit. Extensions of CPPA3 can use this mechanism to define additional transports. An example of a specific transport is `HTTPTransport`, which can be used by an `ebMS3Channel` element, as shown in the following example fragment:

```
<cppa:HTTPTransport id="_BCXV">
  <cppa:Description xml:lang="en">Transport formed from tr3 in P1_cpp and tr1
      in P4_cpp</cppa:Description>
  <cppa:ClientIPv4>127.0.0.1</cppa:ClientIPv4>
  <cppa:Endpoint>https://a.example.com:8080/as4handler</cppa:Endpoint>
  <cppa:TransportLayerSecurity>
    <cppa:TLSProtocol>1.2</cppa:TLSProtocol>
    <cppa:CipherSuite>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</cppa:CipherSuite>
  </cppa:TransportLayerSecurity>
</cppa:HTTPTransport>
```

2.2.11 Payload Profile

A `PayloadProfile` is a specification of the payload, or payloads, that MAY be exchanged in a particular `ActionBinding`. It provides the logical definition of the expected message content using one or multiple `PayloadPart` elements. A `PayloadPart` MAY specify the MIME content type of the payload, its cardinality, schema constraints and other constraints.

It is also possible to specify whether or not a payload part is expected to be signed or encrypted, and if yes, to configure the expected signing or encryption processing. Note that this is distinct from signing and encryption by the `Channel` or `Transport`.

```
<cppa:PayloadProfile id="pp1">
   <cppa:Description xml:lang="en">A UBL Order XML document and any number
      of PDF attachments</cppa:Description>
   <cppa:PayloadPart minOccurs="1" maxOccurs="1" requireSignature="true">
       <cppa:PartName>businessdocument</cppa:PartName>
       <cppa:MIMEContentType>application/xml</cppa:MIMEContentType>
       <cppa:Schema
         location="http://docs.oasis-open.org/ubl/os-UBL-2.1/xsd/maindoc/UBL-Order-
2.1.xsd"/>
       <cppa:Signature>
         <cppa:SignatureAlgorithm
       >http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256</cppa:SignatureAlgorithm>
         <cppa:SigningCertificateRef certId="asigningcert"/>
       </cppa:Signature>
   </cppa:PayloadPart>
   <cppa:PayloadPart minOccurs="0" maxOccurs="unbounded">
       <cppa:PartName>attachment</cppa:PartName>
       <cppa:MIMEContentType>application/pdf</cppa:MIMEContentType>
   </cppa:PayloadPart>
</cppa:PayloadProfile>
```

The `PayloadPart` element assumes Parties agree on a set of `PartName` values like the values "*businessdocument*" and "*attachment*" used in the example or URIs that define standard document types. These values are also used for cross-references from `Package` elements.

2.2.12 Packaging

A `Packaging` element specifies a physical packaging. The CPPA3 schema supports SOAP, MIME, SOAP-with-Attachments and MTOM and is extensible to other packaging types. The element MUST NOT

be used with protocols that do not have configurable packaging. Packaging elements MAY reference `PayloadPart` elements.

As an example, the following sample defines a SOAP with Attachments envelope structure in which both the "*businessdocument*" and "*attachment*" parts are packaged as separate MIME parts, and the attachment part follows the business document part.

```
<cppa:SOAPWithAttachmentsEnvelope id="swa1">
  <cppa:Description xml:lang="en">An ebMS3 SWA envelope, empty body all
  payload parts as separate MIME parts.</cppa:Description>
  <cppa:SimpleMIMEPart PartName="businessdocument"/>
  <cppa:SimpleMIMEPart PartName="attachment"/>
</cppa:SOAPWithAttachmentsEnvelope>
```

2.2.13 Authorization

In a CPP, parties MAY authorize counterparties at one of several layers. In formation and matching, authorization can express with which counterparties parties allow or deny CPAs to be formed from (see section 3.4.5) or matched against (see section 4.2.3). The annotations can also be used to control the visibility of CPPs, or parts of CPPs, to third parties, and to define views on CPPs for these third parties.

Authorization is handled using the `allowed` and `denied` properties that MAY annotate CPP structures and the `PartyIdList` and `PartyIdListRef` elements these properties refer to. A `PartyIdList` element can be referenced using the value of its `id` attribute and is a container for `PartyId` or `PartyIdListRef` elements. A `PartyIdListRef` is a reference to a `PartyIdList` element in the CPP or a list outside of it.

- The `allowed` property references a `PartyIdList` of identified parties with whom CPA formation and matching are allowed and to which visibility is provided. Its absence expresses that there is no restriction to a specific list of identified parties.
- The `denied` property references a `PartyIdList` of identified parties to whom CPA formation and matching are allowed and to which visibility is not provided. Its absence expresses that no specific list of identified parties is denied CPA formation.

The `allowed` and `denied` properties can be included at four levels: `CPP`, `ServiceSpecification`, `ServiceBinding`, and `ActionBinding`. If the `allowed` or `denied` properties are present a more than one level, the lower level property MUST express a further restriction. A lower level `allowed` MUST reference a subset of the parties referenced by the higher level `allowed` and a lower level `denied` MUST reference a superset of the parties referenced by the higher level `denied` .

2.2.14 Signatures

CPP and CPA documents MAY be signed by including an enveloped XML `Signature` in the document. For a CPP, the certificate used to sign the document SHOULD identify the natural or legal entity party to which the CPP relates. Alternatively, the certificate MAY identify some third party that is authorized to publish collaboration protocol profile information about the party.

A CPA MAY be signed one or more times. This allows signature by both Parties, or their authorized representatives, and an overall signature.

A CPP or CPA document is invalid if it is signed and one or more signatures are invalid.

CPP and CPA signatures MUST be valid as specified in the W3C XML Signature [XMLDSIG-CORE, XMLDSIG-CORE1] recommendations.

2.3 Schema

2.3.1 Normative Schema and Schema Documentation

The CPPA3 document structures are defined in the CPPA3 schema and its documentation. The normative Collaboration Protocol Profile and Agreement XML schema, which declares the http://docs.oasis-open.org/ebcore/ns/cppa/v3.0 namespace, is specified in:

schema/cppa3.xsd

The documentation of this schema is embedded in this schema document and is a normative part of this specification. The following two elements defined in the schema can be used as root elements for CPPA3 documents:

- `CPP` is the root element for Collaboration Protocol Profile documents.
- `CPA` is the root element for Collaboration Protocol Agreement documents.

The schema documentation provides:

- Extensive documentation on syntax and semantics of the structures in the CPPA3 schema.
- Explanation of differences to the previous version 2.0 of CPPA.
- For messaging protocols, explanation of the mapping of CPPA3 to configurations to those protocols.
- For ebMS3 and AS4, explanation of the mapping of CPPA3 structures to P-Mode parameters.

A valid CPPA3 document MUST be valid against the CPPA3 XML schema and MUST observe the definitions of syntax and semantics of the structures in the CPPA3 schema specified in the embedded schema documentation, including additional schema constraints not expressed in the XSD.

A valid CPPA3 CPP document MUST have a `CPP` root element.

A valid CPPA3 CPA document MUST have a `CPA` root element.

2.3.2 Alternative Documentation Format (Non-Normative)

A non-normative export in HTML format of this embedded documentation is available at:

documentation/cppa3.html

2.3.3 Normative References in XML Schema

The XML annotation documentation contains normative references to IETF, W3C and OASIS specifications. These references follow the same [*key*] syntax as this specification. All referenced normative specifications are included in section 1.6.

2.3.4 Schema Extensibility

The CPPA3 schema is designed to support a high degree of extensibility.

- The `Channel`, `ChannelFeature`, `Transport` and `Packaging` elements are abstract elements that can be substituted. In addition to the substitutions defined in the CPPA3 schema, other schemas can be designed that are based on the CPPA3 schema, but provide their own additional substitutions for these elements.
- The `Channel` elements have their own `ChannelExtension` elements. This makes it possible to define custom extension elements for an existing `Channel` that further configure the functionality of the `Channel`.

- The elements `CPPAExtension`, `PartyInfoExtension`, `ServiceSpecification-Extension`, `ServiceBindingExtension`, `ActionBindingExtension` enable future extension of those elements.

3 Forming a CPA from CPPs

3.1 Introduction

To be able to use CPP documents as inputs in the formation of a CPA, a method is needed that defines how the CPA is generated from the CPP documents and any other inputs in a consistent and predictable way. In principle, multiple such formation methods are conceivable. This section defines one such method, which can serve as a default CPPA3 CPA formation method. The method is identified using the following URI:

`http://docs.oasis-open.org/ebcore/cppa/v3.0/formation/default`

Note that defining this specific method in this specification does not preclude the definition and use of other methods to form a CPA from input CPPs, as long as those other methods use identifiers different from the `http://docs.oasis-open.org/ebcore/cppa/v3.0/formation/default` URI. Furthermore, there is no requirement that a CPA be created from two CPP input documents at all. A CPA MAY be formed directly using XML editing software, generated using templates, or any other tooling. A CPA MAY also be generated from inputs other than CPP documents.

This section provides a detailed specification of the default CPA formation method. All references to CPA formation in this section concern the CPPA3 default formation method. Since unification is used to do formation, this section will use the terms unification and formation interchangeably.

3.2 Default Formation Method

The default formation method defines how a version 3.0 CPA document is created from its inputs. The minimum inputs to the default CPA formation are:

- An ordered pair of valid version 3.0 CPP documents.
- A value representing the point in time at which the CPA is created.

In addition to these input parameters, implementations MAY offer further configuration options, such as:

- Inputs to, format, or even a specific value to use, for the `AgreementIdentifier` element content.
- User-provided `ActivationDate` or `ExpirationDate`. If unspecified, the default method computes and uses the earliest possible activation date and the latest possible expiration date and time based on the values specified in the CPP and CPA creation time.
- Constraints on, or values to use for, `Username` and `Password` elements.
- The URLs from which the source CPP documents can be retrieved, if available.
- Contextual information for delegation processing (see section 3.4.12)

CPA formation either fails or succeeds. The output of successful default CPA formation is a valid version 3.0 CPA document. If formation fails, no CPA is created. Some reasons for failure are discussed in section 3.4.

Advanced implementations of CPA Formation MAY apply additional validation beyond the minimum requirements specified in this specification. Examples of this are discussed in section 3.4.20. Such validation is out of scope for this specification. Its use could cause formation to fail in more situations if the additional validation fails. The XML output format of a successful formation is not different.

3.3 Equivalence

Different implementations of default CPA formation, or different runs of an implementation, MAY produce CPA XML documents that have non-identical canonical XML infosets [XML-C14N] for the same input CPPs. These distinct CPA documents are nevertheless *equivalent* for CPPA3 use cases because the differences only concern XML-internal aspects of the formed CPA document and have no impact on the information made available to messaging systems using the CPA for configuration:

- CPPs use XML ID/IDREF-valued elements and attributes for document-internal cross-referencing. Implementations SHOULD rename identifiers to prevent ID clashes when processing CPP structures. The mechanism to do so MAY involve prefixing the identifier values with distinct prefix values, but this is left to implementations. Different mechanisms may produce different result values.
- CPA formation assumes generation of XML identifiers for cross-referencing in the CPA. Different implementations MAY use any valid XML identifier format or convention.
- Use of the `CertificateDefaults` element is equivalent to explicitly listing the specified defaults in the elements where they are used.
- The elements `ServiceSpecification`, `Channel`, `ChannelFeature`, `Transport`, `PayloadProfile`, `Packaging`, as well as substitutions for those elements, can occur multiple times. The order of individual elements in these lists of elements is not meaningful.
- Natural language content of the optional `Description` elements is used for documentation purposes in the CPPA3 document only. An implementation of default CPA formation MAY process and/or generate `Description` elements, but this is NOT REQUIRED. For example, a `Description` could explain the source inputs used for the described element.
- `Property` elements can be included with `ActionBinding` elements or within referenced `PropertySet` elements. Similarly, `ChannelFeature` elements may be used in-line as `Channel` child elements or referenced using ID/IDREF elements. Implementations are NOT REQUIRED to preserve the choice for using in-line or referenced reusable properties or channel features in a (or in both) CPP(s) in the generated CPA.
- `TrustAnchorSet` elements can reference `Certificate` elements using `AnchorCertificateRef` or equivalently include such elements as child elements.

3.4 Operation

Default CPA formation method is a process of matching, merging and enhancing information in the input CPP documents. The default formation method is similar to, and inspired by, the concept of unification in logic programming:

- The formation of the whole is defined in terms of the formation of its parts.
- Formation succeeds or fails. If successful, formation has established that the parties can collaborate in complementary roles and that actions could be bound to a compatible unified `Channel`. If unsuccessful, the parties cannot collaborate in complementary roles or actions could not be bound to a compatible `Channel`.
- A CPA includes values that are present and match in both CPPs, but also combines information, taking information that is specified in one CPP but unspecified in the other or vice versa into a combined representation.
- Where two CPPs specify alternatives, formation involves determining which (if any) of multiple pairs of alternatives match.

Default CPA formation assumes a top-down, depth-first backtracking search strategy that evaluates goals in document order and produces results in the resulting order. With arbitrary input CPPs, there may be

multiple matching results. In default formation, the search strategy MUST guarantee at most one result output, equivalent to goal evaluation followed by a “cut” operator. With this constraint, CPA formation is a function that is not commutative for its two CPP arguments, because the order of the two CPP input parameters is relevant.

CPA formation treats parties as peers and uses information from either CPP, while checking for consistency if information is provided in both CPPs. For almost any parameter, a value specified for one party does not override, but is matched against, a value specified for the other party. Some exceptions to this are specified in sections 3.4.21 and 3.4.22.

CPA formation is only defined for valid input CPP documents. This means that implementations MUST validate the input CPP documents against the CPA schema. If one or both CPPs is invalid, the result of unification is undefined.

CPA formation MAY be used in conjunction with CPP preprocessors that add defaults associated with any `ChannelProfile` elements, or that substitute `NamedChannel` elements (see section 2.2.7).

CPP documents MAY be signed. CPA formation implementations MUST validate CPP signatures. Default CPA formation is undefined if any of the CPP document signatures is invalid.

When formation of complex elements involves unification of multiple child elements of different types, the output results for those child elements MUST be appended to output elements in an order that is compliant with the schema definition for the result element.

The remainder of this subsection defines common unification patterns (section 3.4.1), simple child elements (section 3.4.2) and complex child elements (section 3.4.3). From section 3.4.4 on, the formation formation method is described in a top-down fashion including special cases.

3.4.1 Common Formation Patterns

Formation for the `CPA`, `AgreementInfo`, `PartyInfo`, `ServiceSpecification`, `ServiceBinding`, `ActionBinding` and `Channel` elements will be discussed in sections 3.4.4 and further in more detail. In this section, common patterns in formation used for other elements are specified.

Formation operates on elements that have the same namespace name and local name and produces another element with that same namespace name and local name. For example, the process of formation MAY involve matching two `ebMS3Channel` elements in the `http://docs.oasis-open.org/ebcore/ns/cppa/v3.0` namespace which, if successful, produces an `ebMS3Channel` element in the output document in the same namespace. Note that:

- Unification of the `CPP` and `ProfileInfo` elements are exceptional as they do not have a CPA counterpart: the `CPP` elements are inputs into the `CPA` element and the `ProfileInfo` elements are inputs to the `AgreementInfo` element.
- Unification fails for any two elements that do not have the same namespace name or local name.

For simple elements, unification fails if the element content values do not match.

For simple elements with Boolean values, the unification MUST normalize the alternative notations (such as “1” and “true”) before matching.

For complex elements, unification fails if unification of one or more of the sub-elements fails. If one input element has an attribute other than `id`, `propertySetId`, `supportsIPv4` and `supportsIPv6`, the other input element MUST also have this attribute, and the two attribute values MUST be the same.

For the child elements of complex elements, two situations can be distinguished:

- The child element is a simple type. The unification is described in section 3.4.2.
- The child element is of a complex type. The unification is described in section 3.4.3.

3.4.2 Simple Child Elements

In forming the output element for a complex element that has a child element of a simple type, the content of the input elements is reused under the control of a number of parameters that are specified for the element in the context of the parent element:

- *Required*: This parameter expresses if the element is required in the output element. If true, the element is required to be specified in at least one of the two input CPPs. Formation fails if the element is not specified in any of the two input CPPs.
- *Strict*: If formation is strict, then the element MUST be present in both CPPs if it is present in one of the two CPPs, even if it is optional in the CPPA3 XML schema. Formation fails if the element is present in one CPP but not the other. A value N/A means that the element is mandatory in the schema, and therefore required in both CPPs.
- *Intersected*: This parameter applies to elements than can occur more than once as a child element of a node in a CPP. If true, then, if the element is present in both CPPs, the output CPA will contain the intersection of values from the two CPPs. If false, only the first matching element from the first CPP will be included. Whatever the value of the parameters, if the element is only specified in one of the inputs, the output will copy the input from that input. A value N/A for "Intersected" means that the child element only occurs once.

Element	Child Element	Required	Strict	Intersected
AMQPSecurity	SASLMechanism	True	False	False
AMQPTransport	SenderSettleMode	False	False	False
	ReceiverSettleMode	False	False	False
Addressing	Action	True	N/A	N/A
	Endpoint	True	False	N/A
	From	False		N/A
Compression	CompressionType	True	N/A	N/A
	CompressionDictionary	False	True	False
Duplicate-Handling	DuplicateElimination	True	False	N/A
ebMS3Channel	SOAPVersion	True	True	N/A
ebMS3Inferred-RoutingInput	MPCSuffix	False		N/A
	SOAPVersion	False		
Encryption	EncryptionAlgorithm	False	False	False
	EncryptAttachments	False	True	N/A
	EncryptExternalPayloads	False	True	N/A
	EncryptionCertifi-cateRef	False	False	N/A
ErrorHandling	DeliveryFailuresNoti-fyProducer	False	False	N/A
	ProcessErrorNotifyCon-sumer	False	False	N/A
	ProcessErrorNotifyPro-ducer	False	False	N/A

HTTPTransport	HTTPVersion	False	False	True
	ChunkedTransferEncoding	False	False	N/A
	ContentCoding	False	False	True
	Pipelining	False	False	N/A
KeyEncryption	EncryptionAlgorithm	False	False	False
NamedChannel	ChannelName	True	True	False
	SigningCertificateRef	True	True	N/A
	EncryptionCertifi-cateRef	True	True	N/A
PayloadPart	PartName	True	True	N/A
	MIMEContentType	False	False	N/A
	Schema	False	False	True
Receipt-Handling	ReceiptFormat	False	True	N/A
SAMLKey-Confirmed-SubjectToken	SAMLVersion	True	True	N/A
	KeyType	True	True	N/A
SFTPTransport	SSHCipher	False	False	True
Signature	SignatureFormat	False	False	False
	SignatureAlgorithm	False	False	False
	DigestAlgorithm	False	False	False
	CanonicalizationMethod	False	False	False
	SigningCertifi-cateRefType	False	False	N/A
	SignAttachments	False	True	N/A
	SignExternalPayloads	False	True	N/A
SMTPTransport	From	False	False	N/A
	To	False	False	N/A
	Subject	False	False	N/A
Splitting	CompressionAlgorithm	False	False	False
TCPTransport	ClientIPv4	False	False	True
	ClientIPv6	False	False	True
	Endpoint	True	False	N/A
Transport-Restart	RestartProtocol	True	True	N/A
Transport-LayerSecurity	StartTLS	False	True	N/A
	TLSProtocol	False	False	True
	ServerNameIndication-Required	False	False	N/A

	CipherSuite	False	False	True
	ClientCertificateRef	False	False	N/A
	ServerCertificateRef	False	False	N/A
User-Authentication	Digest	False	True	N/A
	Nonce	False	True	N/A
	Created	False	True	N/A
WSChannel	SOAPVersion	True	True	False
WSSecurity-Binding	WSSVersion	False	False	False
WSTransport	SubProtocol	False	False	True

Table 2: Simple Child Elements of Complex Elements

Table 2 defines these parameters for simple child elements of complex elements in the CPPA3 schema.

Note that for the `EncryptionCertificateRef`, `SigningCertificateRef`, `ClientCertificateRef` and `ServerCertificateRef` elements, additional constraints apply, described in section 3.4.20. Failure to meet those requirements will cause unification to fail.

3.4.3 Complex Child Elements

In forming the output element for a complex element that has a child element of a complex type, with the exception of the special cases discussed in sections 3.4.4 to 3.4.27, the following constraints apply:

- The complex child element MUST occur at most once;
- The complex child element MUST occur either in both inputs or in none of them;
- If the element occurs in both inputs, an element with the same namespace name and local name MUST be created in the output, if and only if its constituent parts can themselves be (recursively) unified.

3.4.4 CPA

The CPA element is formed on the basis of an ordered pair of CPP documents and the creation date and time, as follows:

- The `AgreementInfo` structure is formed from the `ProfileInfo` elements in the two `CPP` documents as described in section 3.4.6.
- The `PartyInfo` element is formed from the `PartyInfo` element of the first CPP document and the `CounterPartyInfo` element is formed the `PartyInfo` element of the second CPP document, as described in section 3.4.7.
- The `ServiceSpecification` is formed from the corresponding elements in the two CPPs, as described in section 3.4.8.
- At four levels, formation authorization elements in the CPPs control whether formation is allowed or not, as described in section 3.4.5.

As part of `ServiceSpecification` formation, `Channel, ChannelFeature`, `Transport`, `PayloadProfile`, and `Packaging` elements are processed. If successful, the result structures that are

directly or indirectly bound to actions in service specifications are added, with appropriate cross-reference identifiers, to the `CPA` element.

For `Channel` elements, a second and more specialized way for them (and any `Transport` elements that in turn depend on them) to become included in a CPA is via references from external payload definitions.

Formation fails for the CPA if any of the `AgreementInfo`, `PartyInfo`, `ServiceSpecification` formations fails.

3.4.5 Formation Authorization

As described in section 2.2.13, a CPP may express at four levels, `CPP`, `ServiceSpecification`, `ServiceBinding`, and `ActionBinding`. whether CPA formation is allowed or denied for specific sets of identified parties. These attributes MUST be taken into account when forming a CPA from two CPPs, as follows:

- If, at any of the four levels, one CPP specifies a set of allowed `PartyId` elements using its `allowed` attribute, then unification fails if one of the `PartyId` elements of the other CPP is not in this set.
- If, at any of the four levels, one CPP specifies a set of denied `PartyId` elements using its `denied` attribute, then unification fails if one of the `PartyId` elements of the other CPP is in this set.
- Authorization is only determined at a lower level if authorization at all higher levels succeeded.

3.4.6 AgreementInfo

The CPA `AgreementInfo` is formed using the `ProfileInfo` elements in the input `CPP` elements and the timestamp.

- A CPA has an `AgreementIdentifier`. Its value is left to implementations. Implementations MAY use some function over other values in the `ProfileInfo` or `PartyInfo` elements to compute it. Implementations SHOULD allow users to enforce specific format conventions and MAY allow the user to explicitly set the value to be used for the element.
- The `ProfileIdentifier` values in the `AgreementInfo` are copied from the `CPP`//`ProfileIdentifier` elements. If two `ProfileIdentifier` elements are present in the CPA, the first `ProfileIdentifier` MUST be taken from the CPP related to the party in and `PartyInfo` the second `ProfileIdentifier` MUST be taken from CPP related to the party in `CounterPartyInfo`.
- If an identified input `CPP` was retrieved from a particular URL and it is deemed useful to record this in the formed CPA, this URL MAY be encoded as the value of the `href` attribute of the relevant `ProfileIdentifier`.
- By default, the `ActivationDate` is set to the earliest possible date and time. If the formation request is processed at time *t*, the CPA can be activated at the earliest at the latest of the following six date/times:
    1. the request processing date and time *t.*
    2. *t* + the duration of the `PhaseIn` interval of the first CPP, if provided.
    3. *t* + the duration of the `PhaseIn` interval of the second CPP, if provided.
    4. the `ActivationDate` of the first CPP, if provided.
    5. the `ActivationDate` of the second CPP, if provided.

6. if provided, a user-requested activation date/time

- By default, the `ExpirationDate` is set to the earliest of:
  1. the `ExpirationDate` of the first CPP, if provided.
  2. the `ExpirationDate` of the second CPP, if provided.
  3. if provided, a user-requested expiration date/time.

If no `ExpirationDate` is provided in any of the CPPs and there is no user-requested expiration date/time, the CPA MUST NOT contain an `ExpirationDate` element. Its absence means the CPA is open-ended.

If the determined `ActivationDate` is later than the determined `ExpirationDate`, formation fails for the `AgreementInfo` element.

3.4.7 Party Information

Party Information in the CPA is created as follows:

- The `PartyInfo` is derived from the `PartyInfo` in the first argument CPP.
- The `CounterPartyInfo` is derived from the `PartyInfo` in the second argument CPP.

In both cases, the following types of subelement are simply copied:

- `PartyName`
- `PartyId`, retaining the order of elements in the source CPP.
- `PartyContact`

The CPA MUST include all `Certificate` and `TrustAnchorSet` elements and all under `PartyInfo` that are referenced from any `Channel` or `Transport` element required for a `ServiceBinding` included in the `CPA`.

`Certificate` and `TrustAnchorSet` elements and `IDPRegistration` and `IDPRegistrationSet` elements under `PartyInfo` that are not referenced from any `Channel` or `Transport` element required for a `ServiceBinding` SHOULD NOT be included in the `CPA`.

3.4.8 Service Specification

Within a CPP, a `ServiceSpecification` specifies the service interaction capability of the profiled party acting in a specific role R1 with potential unnamed other parties acting in a specific role R2. Both the `CPP` and `CPA` schema definitions allow from one to an unbounded number of `ServiceSpecification` child elements.

Default CPA formation specifies:

1. How a list of CPA `ServiceSpecification` elements is formed from the corresponding lists in the CPPs;
2. How individual CPA `ServiceSpecification` elements are formed from pairs of `ServiceSpecification` elements, one from each CPP.

As for (1), in default CPA formation, the list of `ServiceSpecification` elements in a CPA is derived from the lists of `ServiceSpecification` elements in the two CPPs as follows:

- For each `ServiceSpecification` in the first CPP, with `PartyRole/@name` R1 and `CounterPartyRole/@name` R2, a corresponding `ServiceSpecification` is searched for in

the second CPP with `PartyRole/@name` R2 and `CounterPartyRole/@name` R1. If any of the `uuid`, `name` and `version` attributes defined in [ebBP] is present on the element in one CPP, it MUST also be present on the corresponding element in the other CPP with an identical value.
These two `ServiceSpecification` elements constitute a potential match pair.
Formation of the CPA fails if none such potentially matching pairs of `ServiceSpecification` elements can be found for any pair of roles R1/R2.

- For any pair of potentially matching pairs of `ServiceSpecification` elements from the two CPPs identified in the previous step, form the unified CPA `ServiceSpecification` element, if formation is possible for those elements.
  Formation of the CPA fails if formation fails for all these potentially matching pairs of `ServiceSpecification` elements.

As for (2), a `ServiceSpecification` element in the CPA for complementary roles R1/R2 can be formed from a pair of `ServiceSpecification` elements, one from each of the two CPPs, as follows:

- For each `ServiceBinding` in the first `ServiceSpecification` with a `Service` S, find, in the second `ServiceSpecification`, the corresponding `ServiceBinding`, i.e. the `ServiceBinding` that covers the same `Service` S.
- Unify each pair of `ServiceBinding` following the specification in section 3.4.9. If unification succeeds, append the unified `ServiceBinding` as child to the `ServiceSpecification` element in the CPA.
- The CPA will contain a `ServiceSpecification` with a `Service` element with content S if and only if at least one of the child `ServiceBinding` element pairs is successfully unified. If none of the pairs of CPP `ServiceBinding` elements, for some `Service` S, can be formed into a CPA `ServiceBinding` element for that service, formation fails for the `ServiceSpecification` and no `ServiceSpecification` is created in the CPA for the <R1,R2> role pair.

3.4.9 Service Binding

A `ServiceBinding` contains a list of `ActionBinding` child elements. In default CPA formation, unification of the `ServiceBinding` consists of two steps:

1. Pairing `ActionBinding` child elements in the corresponding `ServiceBinding` element in the two CPPs.
2. For each pair, unify the two `ActionBinding` elements. This is described in section 3.4.10.

As for step (1), two `ActionBinding` elements are paired if:

- They have the same value for the mandatory `action` attribute.
- If the optional `replyTo` attribute is present in one of the two `ActionBinding` elements, it MUST be present in both `ActionBinding` elements. Furthermore, the values for this attribute MUST be the same value in both `ActionBinding` elements.
- If the value for the `sendOrReceive` attribute in the first element is "*send*", the value of the `sendOrReceive` attribute in the first element MUST be "*receive*" and vice versa.

In the formed CPA, an `ActionBinding` element with values for the `action`, `replyTo`, and `sendOrReceive` attributes copied from the input CPPs is present if and only if the paired elements can be unified.

Unification fails for a `ServiceBinding` if unification fails for a child pair of `ActionBinding` elements, of which at least one does not have the `use` attribute present, or has it present with value *required.*

3.4.10 Action Binding

Unification of a pair of `ActionBinding` elements in a `ServiceBinding` succeeds if and only if:

- At least one compatible pair of child `ChannelId` elements can be computed. Unification of a pair of channel elements identified using `ChannelId` elements is described in section 3.4.11.
- At least one pair of referenced payload profiles is compatible, as specified in section 3.4.17.
- Any pair of `Property` elements is compatible, as specified in section 3.4.19.

The order of child `ChannelId` elements in an `ActionBinding` element in a CPP is significant and expresses a preference, with earlier elements being preferred to later elements in the list. In a CPA, the number of `ChannelId` elements per `ActionBinding` MUST be exactly one. If multiple compatible pairings of `ChannelId` elements for the two `ActionBinding` elements exist, in default CPA formation the single pair that is selected for use in the CPA is the first pair in the ordered set of matching pairs formed by ordering first by the `ActionBinding` order in the first CPP and second by the `ActionBinding` order in the second CPP. This is the first matching pair that would be found in a top down, depth-first, left to right backtracking search strategy. Implementations MAY use this search strategy to avoid unnecessary channel matches. However, this decision is left to implementations. To handle situations where multiple pairs of `ActionBinding` elements reference the same candidate pairs of channel definitions, implementations MAY memoize the channel unification computation results to avoid redundant computation. This decision is also left to implementations.

An `ActionBinding` MAY reference payload profiles using the `PayloadProfileId` element. As with child `ChannelId` elements, the relative order expresses a preference, pairs are matched in depth-first second CPP order, and the first matching pair (if any) is included in the CPA.

These attributes MUST be either present in both or absent in both input elements. If present, the referenced payload profiles MUST unify as specified in section 3.4.17. If unification of the referenced profiles fails, unification of the referencing `ActionBinding` elements fails.

3.4.11 Channels

Channels are referenced and used in one of three ways:

1. As `ChannelId` elements in an `ActionBinding`. In this case, the channels are matched as described in section 3.4.10. The first successful unification of the first two matching input channels is used in the output CPA.
2. As channels referenced from the definitions of other channels, for use for a particular purpose. In this case the reference MUST be present in both channels being matched, for the same purpose, and there is only one pair of channel definitions to be matched. If the unified referencing channel is included in the output CPA, the unified referenced channel MUST be included as well so the dependency can be resolved in the result CPA.
3. As channel referenced from `ExternalPayload` elements.

The following paths from a referencing channel to a referenced channel MAY occur in CPP or CPA document:

- `TransportChannel` > `RequestChannelId`
- `ebMS2Channel` > `ErrorHandling` > `ReceiverErrorsReportChannelId`
- `ebMS3Channel` > `ErrorHandling` > `ReceiverErrorsReportChannelId`
- `AS1Channel` > `ReceiptHanding` > `ReceiptChannelId`
- `AS2Channel` > `ReceiptHanding` > `ReceiptChannelId`

- AS3Channel > ReceiptHanding > ReceiptChannelId
- ebMS2Channel > ReceiptHanding > ReceiptChannelId
- ebMS3Channel > ReceiptHanding > ReceiptChannelId
- ebMS3Channel > PullHandling > PullChannelId
- ebMS3Channel > Splitting > SourceChannelId
- ebMS3Channel > AlternateChannelId
- ebMS3Channel > @package > SimpleSOAPEnvelope > ExternalPayload > ChannelId
- ebMS2Channel > @package > SOAPWithAttachmentsEnvelope > ExternalPayload > ChannelId
- ebMS3Channel > @package > SOAPWithAttachmentsEnvelope > ExternalPayload > ChannelId
- WSChannel > @package > MTOMEnvelope > ExternalPayload > ChannelId

For each of these elements referencing channels, unification causes the unification of the referenced channels to be evaluated. If successful, the resulting unified channel is to be included in the result CPA and its identifier is used for cross-referencing. If unification fails, unification of the structure containing the referencing element also fails.

Unification of channels is specified in section 3.4.2 and 3.4.3 for most of their simple and complex subelements.

The CPPA3 schema defines the AS1Channel, AS2Channel, AS3Channel, ebMS2Channel, ebMS3Channel, NamedChannel, TransportChannelandWSChannel channel types. The table in section 3.4.2 specifies how the child elements are unified. Unification fails if any of these unifications fails.

Channel elements MAY carry the following attributes:

- transport: this element MUST be present on both channels if it is present on one. The values are cross-references to transport definitions in the source CPPs, which MUST be unified successfully in order for the channel unification to succeed. If unification succeeds, the resulting unified Transport MUST be included in the generated CPA and a transport attribute included on the generated channel that references it. Unification of channels is specified in section 3.4.2 and 3.4.3.
- asResponse: the occurrence of this attribute and its value (if present) MUST be the same for the two input channels, else unification fails. If successful, the attribute and its value are carried over to the output channel.
- package: this element MUST be present on both channels if it is present on one. The values are cross-references to package definitions, which MUST be unified successfully for the channel unification to succeed. If unification succeeds, the resulting unified Package MUST be included in the generated CPA and a package attribute included on the generated channel that references it. Unification of packages is specified in section 3.4.18.

3.4.12 Delegation Channels

When unifying ActionBinding elements in two CPPs for Parties P1 and P2 in which at least one of the two action bindings provides a DelegationChannel, three situations can be distinguished:

1. The DelegationChannel is in the CPP for the first Party P1. The Channel in the other CPP is of a type other than DelegationChannel.

2. The DelegationChannel is in the CPP for the second Party P2. The Channel in the other CPP is of a type other than DelegationChannel.

3. The action is bound to a DelegationChannel in both CPPs.

In the first situation, the DelegationChannel for the ActionBinding in the CPP for Party P1 targets a third Party P3 as PartyId. To enable unification of the CPP for P1 and P2, it is REQUIRED that P2 and P3 are known to be configured for communication for the action. This assumes contextual information is available during CPA formation. The mechanism using which this information is made available to the unification processor is out of scope for this specification. In the formed CPA for the CPPs for P1 and P2, a DelegationChannel is included that contains the P3 PartyId and (if provided) ProfileIdentifier from the CPP of P1.

The second situation is similar to the first situation. The DelegationChannel for the ActionBinding in the CPP for Party P2 targets a third Party P3 as PartyId. To enable unification of the CPP for P1 and P2, it is REQUIRED that P1 and P3 are known to be configured for communication for the action. As in the first situation, any mechanisms using which relevant contextual information is made available are out of scope for this specification. In the formed CPA for the CPPs for P1 and P2, a DelegationChannel is included that contains a CounterPartyId with content and type attribute value copied from the P3 PartyId in the CPP of P2 and (if provided) the provided ProfileIdentifier from the CPP of P2.

The third situation is a combination of the first two situations and subject to the combined specified conditions. If the conditions are met, the result CPA includes a DelegationChannel for a P3 PartyId to which the P1 Party in the CPA PartyInfo delegates messaging and a P4 CounterPartyId to which the P2 Party in the CPA CounterPartyInfo delegates messaging. For both third parties a specific ProfileIdentifier elements MAY be specified if provided in the source CPPs.

3.4.13 Property Sets

The CPPA3 concept of PropertySet provides the ability to reuse definitions of sets of Property elements in multiple ActionBinding channels using cross-references rather than embedding. For the purposes of unification, this is just a notational variant. Implementations of CPA formation MAY inline referenced properties in CPPs prior to unification, and MAY choose either inline copies or references in generated CPAs.

3.4.14 Channel Features

To CPPA3 concept of ChannelFeature provides the ability to reuse definitions of channel features in multiple channels using cross-references rather than embedding. For the purposes of unification, this is just a notational variant. Implementations of CPA formation MAY inline referenced features in CPPs prior to unification, and MAY choose either inline copies or references in generated CPAs.

3.4.15 MaxSize and FragmentSize

In CPPA3, the maximum size of the message can be specified at the Channel level. In the Splitting and Joining feature, the maximum size of a fragment can be specified.

In formation, the lowest specified value in either of the two input CPP structures is selected.

3.4.16 Transports

For Transport elements, unification needs to check compatibility of the optional supportsIPv4 and supportsIPv6 attributes. Unification fails if one Transport is only available using IPv4 and the other only using IPv6.

3.4.17 Payload Profiles and Parts

A `PayloadProfile` is a list of `PayloadPart` elements. For a `PayloadProfile` to unify:

- The lists of `PayloadPart` input elements MUST have the same length.
- The lists of `PayloadPart` input elements MUST unify pairwise, preserving document order, i.e. each `PayloadPart` in the first list MUST unify with the `PayloadPart` in the corresponding position in the second list. If unification succeeds for all parts, each result of unification is included in the output list of `PayloadPart` elements, again preserving document order.
- Unification of `PayloadPart` unify elements is described in section 3.4.2.

Unification fails for the `PayloadProfile` if any of these requirements is not met.

3.4.18 Package

The CPPA3 package elements reflect the nesting and ordering of the specified elements. To match these elements:

- The input elements MUST have the same namespace name and local name.
- The child elements lists MUST have the same length.
- For each child element at a position in the first input child list and child element at the same position in the second input child lists:
  - The child elements MUST have the same namespace name and local name.
  - The child elements MUST reference the same `PartName`.

If any of these requirements is not met, unification fails for the package.

3.4.19 Properties

The `ActionBinding` elements and the AMQP `ConnectionProperties`, `SessionProperties` and `LinkProperties` Elements MAY have one or multiple `Property` child elements. Lists of `Property` elements in two input CPPs unify if and only if:

- The lists are equal in length.
- For each `Property` child with a particular `name` in one input list, a `Property` child with the same `name` is present in the other input list.
- For each of candidate matching pair of elements, the values for `minOccurs` and `maxOccurs` are the same.

CPA formation for `ActionBinding` elements that reference `PropertySet` elements proceeds as if the `Property` children of the referenced set were included as child elements of the `ActionBinding`.

Note that unification of `Property` child elements differs from other child list unifications, for example, of `PayloadPart` input elements in `PayloadProfile` (see section 3.4.17), in that the order of child `Property` elements is not relevant.

Unification fails if any of these requirements is not met. If unification succeeds, the output CPA includes a copy of the lists of `Property` child elements.

3.4.20 Certificates and Trust Anchors

CPPA3 allows both direct references to certificates, using XML Signature `KeyInfo`, and indirect references using the XKMS `LocateRequest`. Parties can also specify `TrustAnchorSet` elements that

specify which trusted Certification Authorities and root certificates they accept as issuers of certificates, and specify certificate policies that policy and issuing certificate authorities MUST implement.

In case of direct references and of indirect references, CPPA3 unification includes validation of the intended use of X.509 certificates by parties against constraints on accepted Certification Authorities issuing such certificates by their counter parties. This validation can be enforced for certificates used for signing, encryption and authentication, at both the transport and the messaging layers.

- For a particular channel that uses signing, a receiving party MAY specify a `SigningTrustAnchorSetRef` and/or one or more `SigningCertificatePolicy` identifiers. A sending party MAY include a `SigningCertificateRef` for the channel. If both situations apply, it MUST be validated that the referenced signing certificate has been issued by a Certification Authority (CA) root certificate included in the referenced `TrustAnchorSet` for the receiver (if present) and that the issuing CA and any policy CAs implement at least one policy referenced by a `SigningCertificatePolicySetRef` (if present).
- For a particular channel that uses encryption, a sending party MAY specify an `EncryptionTrustAnchorSetRef` and/or one or more or a `EncryptionCertificate-Policy` references. A receiving party MAY specify an `EncryptionCertificateRef` for the channel. If both situations apply, it MUST be validated that the referenced encryption certificate has been issued by a Certification Authority included in the referenced `TrustAnchorSet` for the sender (if present) and that the issuing CA and any policy CAs implement at least one policy referenced by a `EncryptionCertificatePolicySetRef` (if present).
- For a TLS secured transport, the TLS client party MAY specify a `ClientCertificateRef`. The server party MAY specify a `ClientTrustAnchorSetRef` and/or one or more `Client-CertificatePolicy` identifiers for a particular exchange. If both situations apply, it MUST be validated that the client certificate has been issued by a Certification Authority included in the referenced `TrustAnchorSet` for the server (if present) and that the issuing CA and any policy CAs implement at least one policy referenced by a `ClientCertificatePolicy-SetRef` (if present).
- For a TLS secured transport, the TLS server party MAY specify a `ServerCertificateRef`. The client party MAY specify a `ServerTrustAnchorSetRef` and/or one or more `Server-CertificatePolicy` identifiers for a particular exchange. If both situations apply, it MUST be validated that the server certificate has been issued by a Certification Authority included in the referenced `TrustAnchorSet` for the client (if present) and that the issuing CA and any policy CAs implement at least one policy referenced using `ServerCertificatePolicy-SetRef` (if present).

Implementations of the default CPA formation method MUST validate that a specific leaf certificate of a signing, encryption, TLS client or server authentication type, if presented, has been issued by one of the Certification Authorities in a referenced `TrustAnchorSet` if present for that certificate type. As the `KeyInfo` structures in the CPP include the full certificate path for the certificate, this is a simple match on CPP XML structures. Any certificate references that successfully pass these tests MUST be added to the result CPA. These CPA formation time tests obviate the need to include trust anchor set references are not needed in the result CPA.

If in a CPP a party expresses a requirement on the receiving side of a channel that the sender counter party provides a specific leaf signing certificate for the sender side of a channel using a `Signing-CertificateRequired` with a true value, unification fails if the sender does not specify such a certificate using a `SigningCertificateRef` element. Similar checks MUST be performed for the `EncryptionCertificateRequired` and `EncryptionCertificateRef` elements, the `ClientCertificateRequired` and `ClientCertificateRef` elements, and the `ServerCertificateRequired` and `ServerCertificateRef` elements.

For signing, encryption, client and server TLS, if an input CPP element specifies a trust anchor set, the other CPP element does not specify a leaf certificate and all other constraints are met, then unification MUST include the trust anchor set reference in the result CPA.

Details of the handling of XKMS `LocateRequest` elements in CPA formation are left to implementations and/or to CPPA3 usage profiles. Indirect references involving XKMS `LocateRequest` elements in any of the CPPs MAY be evaluated and replaced by the located `KeyInfo`, during CPA formation, in contexts where an indirect reference cannot be used in the CPA. The latter would be the case if a messaging protocol, or an implementation of a messaging protocol, does not support runtime XKMS request processing for a channel or transport and/or requires end entity certificates to be statically configured. Any such retrieved certificate MUST be validated against any specified trust anchor or certificate policies, as specified above.

Implementations of default CPA formation MAY implement additional validation of the specified certificates, both the end entity certificates, any intermediate CA certificates and ultimately the used root CA certificates. This validation MAY involve certificate validation such as CRL or OCSP checks. However, such checks are normally already done in messaging systems. Furthermore, any checks made during CPA formation would only be done at that initial point in time, whereas the validity of certificates has to be confirmed throughout the lifetime of the agreement. These additional checks are therefore NOT REQUIRED for the formation of the CPA.

3.4.21 Reliable Messaging

CPA formation handles two reliable messaging elements asymmetrically:

- `PersistDuration` is used by the receiver in support of duplicate elimination. For this reason, the `PersistDuration` included in the CPA is taken from the Receiver CPP. Any value specified in the Sender CPP is ignored.
- `RetryHandling` is a parameter that is relevant for the Sender. For this reason, the `RetryHandling` included in the CPA is taken from the Sender. Any value specified in Receiver CPP is ignored.

3.4.22 MPC

In CPPA3, the ebMS3 MPC, if specified, is set as value of the `mpc` attribute at the `ebMS3Channel` element. When used with Pull mode, the Sender acts as Pull server. The Pull server MAY want to determine the MPC name from which the messages are served, if different from the default MPC. In CPA formation, the Receiver Party therefore MUST NOT specify any MPC name. CPA formation fails if the party specifies an MPC name.

3.4.23 User Authentication

The `UserAuthentication` element is used for configuring user authentication in `WSSecurityBinding` and `TCPTransport`. The `Username` and `Password` elements MUST be set in the generated CPA. The values to be used are left to implementations.

3.4.24 Elements to Sign or Encrypt

The `SignElements` and `EncryptElements` elements in `WSSecurityBinding` configure which elements to sign or encrypt by specifying a series of XPath `Expression` elements.

- The cardinality of the element MUST be the same in both input sources. That is, either both have the element or neither of them has the element.
- The content of the element, if present, is a list of `Expression` elements. Both input lists MUST be the same in length.

- The lists of `Expression` input elements MUST unify pairwise, preserving document order, i.e each `Expression` in the first list MUST unify with the `Expression` in the corresponding position in the second list and each result of unification is included in the output list.

Two `Expression` elements match if they have the same XPath content.

3.4.25 WS-Addressing From

When unifying two Web Services `Addressing` elements, the WS-Addressing `From` element of the Sender CPP `Addressing` element, if present, is copied to CPA `Addressing` element. Any element on the Receiver CPP `Addressing` element is ignored.

3.4.26 RestartInterval and JoinInterval

When unifying the two `TransportRestart` elements, the Receiver `RestartInterval` element, if present, is copied to the CPA `TransportRestart` element. This element MUST NOT be specified in the Sender CPP.

Similarly, when unifying the two `Splitting` elements, the Receiver `JoinInterval` element, if present, is copied to the CPA `Splitting` element. This element MUST NOT be specified in the Sender CPP.

3.4.27 SAML Tokens

The CPPA3 schema provides an abstract `SAMLToken` element and a `SAMLKeyConfirmed-SubjectToken` substitution element. This supports configuration of ebMS3 messaging systems for use with the ebMS3 SAML Conformance clause specification [ebMS-saml-conformance], which profiles the use of ebMS3 and AS4 with the WS-Security SAML Token Profile [WSS-SAML-Token-Profile-V1.1.1]. That specification supports use of WS-Security using symmetric or asymmetric holder-of-key SAML tokens to authenticate ebMS3 Senders to Receivers.

If a Party uses SAML in any channel in a CPP or CPA, it:

- MUST include one or more `IDPRegistration` elements in its `PartyInfo`. These represent registrations of the party with a particular identity provider service. An `IDPRegistration` element is uniquely identified using its `ProviderID` element value.
- MUST include one or more `IDPRegistrationSet` elements in its `PartyInfo`. These represent identified sets of IDP registrations, for reference from key-confirmed SAML tokens.
- Any `WSSecurityBinding` element that uses holder-of-key SAML Authentication MUST include a `SAMLKeyConfirmedSubjectToken` element.
- This `SAMLKeyConfirmedSubjectToken` element MUST reference, using a `IDPRegistrationSetRef`, a set of IDP registrations. One IDP registration in the referenced set of IDPs set MUST be used to authenticate sender and to encrypt any symmetric keys.

The use of SAML is aimed at supporting dynamic many-to-few communication. Therefore, formation of agreements is typically assumed not to be used or desired. Instead, the CPPs are expected to provide sufficient information to allow the communication to be initiated dynamically. However, this specification defines CPA unification for the SAML feature in CPPA3 if only because it allows ahead-of-time determination of the compatibility of Sender and Receiver configurations.

In forming a CPA from a CPP, the `IDPRegistration` and `IDPRegistrationSet` elements MUST be copied to the result CPA.

When unifying `SAMLKeyConfirmedSubjectToken` elements in two CPPs:

- The `SAMLVersion` and `KeyType` elements MUST be unified following the simple child element unification pattern.

- At least one `IDPRegistration` in Sender's applicable `IDPRegistrationSet` MUST also be included in Receiver's applicable `IDPRegistrationSet`.
- It MUST be verified that each of Receiver's `SAMLAttribute` elemens that has a value `required` for its `use` attribute is also specified as being `required` for Sender.

If unification is successful:

- The `ProviderID` of the first matching IDP registration MUST be included as child element in the result CPA.
- The `SigningCertificateRef`, if specified for Sender, is included in the CPA result structure. This only applies in situations using an asymmetric proof key type.

4 Matching a CPA to a CPP

4.1 Introduction

In the process of configuring a new communication partner or service, or updating an updating configuration, a party may be presented with a CPA that is claimed to provide the configuration for the new or updated partner or service and that is claimed to be compatible with party's CPP. This CPA MAY be created from the party's CPP and another CPP using the default CPA formation method defined in section 3, using some other automated method or using some manual process. The CPA MAY also be created using just one CPP or without any CPP as input. The presented CPA may be provided by the communication partner, by a third party community management service, or obtained otherwise.

To be able to use the presented CPA, it is necessary to verify that the CPA is consistent with a party's configuration. The CPA is consistent with and “matches” an input CPP, if it can be a potential result of unifying the input CPP with another (possibly unknown) CPP. It this situation, it can be said to validly select data from the CPP (where the CPP offers multiple options among which one can be selected) and to extend it with other data provided by the other CPP.

The section defines a method to validate this consistency by systematically matching the CPA against a CPP representing the party's configuration. The method is identified using the following URI:

`http://docs.oasis-open.org/ebcore/cppa/v3.0/matching/default`

Note that defining this specific method in this specification does not preclude the definition and use of other methods to match a CPP against a CPA, as long as those other methods use identifiers different from the `http://docs.oasis-open.org/ebcore/cppa/v3.0/matching/default` URI.

The inputs to the method are a valid CPPA3 CPP document and a valid CPPA3 CPA document. The output is an indication of the success or failure of the match of the two documents. To facilitate trouble-shooting in case of match failure, implementations MAY provide logs or other information that help users pinpoint the source of any non-matches.

The method follows the input CPA structure, linking its parts to corresponding parts in the input CPP. This is because the CPP may provide a lot of data on channels, services, transports etc. that are not used in the CPA, whereas in the CPA every channel, service, transport etc. must be matched against some corresponding structure in the CPP. The specification describes which and how data in the CPP MUST match data in CPA.

The matching method is used in the CPPA3 Agreement Registration service, specified in section 5.4.

4.2 Operation

4.2.1 Party Information

A CPP contains a `PartyInfo` element that provides information about the party. A CPA contains a `PartyInfo` element and a `CounterPartyInfo` element. Therefore the first step in matching is to map the CPP `PartyInfo` element against one of these structures by checking that:

- The `PartyName` elements have the same textual content.
- The `PartyId` elements (element content and value of the `type` attribute, if present) in the CPA are a subset of the `PartyId` elements in the CPP.

The CPA fails to match the CPP if the CPP `PartyInfo` element cannot be matched against either of these elements.

If the CPP `PartyInfo` element matches the `CounterPartyInfo` element in the CPA rather than the `PartyInfo` element in the CPA, all subsequent matches of `ServiceSpecification` MUST swap `PartyRole` and `CounterPartyRole` elements accordingly and all matches of `ActionBinding` MUST swap the "`send`" and "`receive`" values of the `sendOrReceive` attribute.

4.2.2 Profile and Agreement Information

If the `AgreementInfo` element in the CPA contains two `ProfileIdentifier` elements, the value of the `ProfileIdentifier` element in the CPP MUST match the first of these element if the CPP relates to the `PartyInfo` party in the CPA or the second if it relates to the `CounterPartyInfo` party in the CPA.

If the CPP and the CPA both contain an `ActivationDate` element, the value in the CPA MUST NOT precede the value in the CPA.

If the CPP and the CPA both contain an `ExpirationDate` element, the value in the CPA MUST NOT follow the value in the CPA.

4.2.3 Match Authorization

Similar to the authorization checks performed in formation as described in section 3.4.5, the authorization information that can be expressed in a CPP using the `allowed` and `denied` attributes at the four levels they occur at MUST be checked against the party identifiers of the other party occurring in the CPA.

4.2.4 Service Specifications

For every `ServiceSpecification` in the CPA, there MUST be a matching `ServiceSpecification` in the CPP. Two `ServiceSpecification` elements match if their `PartyRole` and `CounterPartyRole` element values match, possibly swapped as explained in 4.2.1, and if every `ServiceBinding` in the CPA matches a `ServiceBinding` in the CPP.

Note that the CPP may have `ServiceSpecification` elements for role pairs that are not in the CPA. These are ignored.

4.2.5 Service Bindings

A `ServiceBinding` in the CPA matches a `ServiceBinding` in the CPP if:

- their `Service` child elements have matching element content and (if present) `type` attribute values.
- For every `ActionBinding` child element of the `ServiceBinding` element in the CPA there is an `ActionBinding` child element of the `ServiceBinding` element in the in CPA that has the same values for the `action` attribute and corresponding value of `sendOrReceive` attribute (possibly swapped as explained in 4.2.1) of which at least one referenced channel matches the channel referenced in the CPA. And of which at least one referenced payload profile matches the payload profile referenced in the CPA.
- For every required `ActionBinding` child element in the CPP, a corresponding `ActionBinding` child element is present in the CPP.

Note that in a in the CPP, in a `ServiceSpecification` for a pair of roles there may be `ServiceBinding` elements for services that are present in the `ServiceSpecification` for the matching pair of roles in the CPA. These are ignored.

4.2.6 Complex Elements

A Channel element and other complex schema elements in the CPA match a corresponding element in the CPP if they have the same element tag including namespace, if all its complex and simple child elements match recursively and if all shared references to certificates reference the same certificate and all referenced channels and transports.

Exceptionally, no match is required for some reliable messaging elements for reasons explained in 3.4.21.

4.2.7 Simple Elements

A simple child element in the CPA matches the CPP if either:

- there is no simple child element of the same type in the corresponding parent element in the CPP.
- there is such at least one such corresponding simple child element that has the same element content.

4.2.8 Certificate References, Certificates and PKI

A certificate reference child element in the CPA matches the CPP if either:

- there is no corresponding certificate reference child element in the CPP, or
- there is such a corresponding certificate reference element and the referenced certificate structures have identical content for the leaf certificate `X509Certificate` element (if present in both) or the same `KeyName`.

If the CPA includes a reference of a particular type to a certificate that is not in the CPP (such as a `SigningCertificateRef`), and if the the CPP has reference to a trust anchor set for that type (such as a `SigningTrustAnchorSetRef`), then the referenced certificate MUST be comptable with the specified trust anchor sets, as is specified for unification in section 3.4.20.

4.2.9 Channels

For matching, CPA channels can be divided in two groups: delegation channels and all other types of channels.

If there is a `DelegationChannel` in a CPA, one or both parties delegate(s) message processing to a third party, evidenced by the presence of `PartyId` and/or `CounterPartyId` child elements in the `DelegationChannel` element, as explained in section 2.2.8. Therefore, depending on the presence of these subelements, a corresponding `DelegationChannel` element, with identical party identifier content, has to be present in the CPP.

If the CPA channel is not a delegation channel, the complex element matching of section 4.2.6 applies.

4.2.10 Channel References

A child element in the CPA that references a channel is valid if a channel reference of the same type is present in the CPP as a child of the corresponding parent element, and if the channel referenced from the CPA matches the channel referenced from the CPP.

5 CPPA3 Discovery and Registration

5.1 Introduction

A metadata service for business interactions provides information about what kinds of data transactions, and what kinds of enabling technologies for those transactions, are available for specific business process participants. This section specifies both a metadata service based on CPPA3 CPP documents and a service to locate such services based on their party identifiers. It also specifies a CPA registration service.

The goal of a Dynamic Delegation Discovery System (DDDS) application for metadata service discovery is to find URLs of specific types of metadata services, by using a Domain Name System (DNS) query string that represents an identity of a person or organization. Section 5.2 describes how the OASIS Business Document Metadata Service Location [BDX-Location-v1.0] can be used to retrieve one or more URLs of CPPA3-based metadata services from the DNS.

The CPPA3 CPP is an electronic document format that describes the business and technical capabilities of an individual party and associated parameters. A CPP covers both sending and receiving capabilities, message channel, transport and networking options, security tokens and trust anchors. Section 5.3 describes a simple HTTP-based metadata service protocol that uses CPPA3 CPP as metadata data format.

To share and deploy a new agreement, parties MAY use the CPA registration service described in section 5.4.

5.2 CPPA3 Metadata Service Location

The goal of a Dynamic Delegation Discovery System (DDDS) application for metadata service discovery is to find URLs of specific types of metadata services, by using a DNS query string that represents an identity of a party. In other words, the goal is to retrieve one or more URLs from the DNS that will enable finding out more about an entity's enabled metadata services. This functionality is provided by the OASIS Business Document Metadata Service Location [BDX-Location-v1.0] specification, which defines a mechanism to retrieve locations of metadata services for parties, published as U-NAPTR DNS records, using DNS query strings that identify those parties. Implementations of CPPA3 Metadata Service Location MUST conform to the OASIS Business Document Metadata Service Location [BDX-Location-v1.0] specification. This specification does not constrain the format for DNS query strings for use with Business Document Metadata Service Location. The specification describes a number of options, including mechanisms for non-DNS Participant Identifiers and for use of service provider domains.

When used to locate CPPA3 CPP documents, the published service location URI MUST be a Uniform Resource Locator (URL), i.e. a URI [RFC3986] that, in addition to identifying a CPPA3 CPP resource, provide a means of locating the CPPA3 CPP resource by describing its primary access mechanism (e.g., its network "location"). The published URL MUST directly identify and locate the CPPA3 CPP metadata resource, i.e. it MUST NOT be limited to the domain name of the metadata service.

To select only CPPA3 CPP documents, the *service* field of the U-NAPTR resource record MUST be set to the value "*meta:cppa3*". The use of a pre-defined value for the *service* field allows additional U-NAPTR records for the same DNS query string to be published that use other values for the *service* field, if metadata resources in additional formats (out of scope for this specification) are to supported.

When used in combination with the metadata service retrieval protocol described in section 5.3, the URL scheme MUST be "*http*" or "*https*" (case insensitive).

5.3 CPPA3 Metadata Service

5.3.1 Introduction

This section describes a simple (secure) HTTP binding for CPPA3-based metadata services. Metadata service resources are provided using HTTP server functionality. Metadata service resources are requested using HTTP client requests.

This specification does not preclude definition or use of other, more sophisticated metadata services. In particular CPPA3 CPPs MAY be stored in a repository and MAY be discovered using the capabilities of a registry such as the ebXML Registry [ebRS, ebRIM]

5.3.2 HTTP Protocol Binding

Implementations MUST support HTTP version 1.1 [RFC7230] or 2.0 [RFC7540]. Use of transport layer security version 1.2 or higher [RFC5246, RFC8446] is RECOMMENDED.

Server implementations MUST support use of the HTTP GET method to provide access to resources using CPPA3 CPP documents. Use and semantics of HTTP methods other than the GET method is out of scope for this specification.

Servers MAY require client authentication and authorization as described in section 5.3.5.

Implementations MAY implement caching using the associated header fields that control cache behavior or indicate cacheable response messages described in [RFC7234].

For requests to existing resources, for successfully authenticated client consumers (if authentication is used) that are allowed access the CPPA3 CPP document or parts thereof (if authorization is used), the HTTP 200 OK status code MUST be used and a representation of the target resource MUST be returned. The response *Content-Type* header MUST be set to the value “text/xml” (case-insensitive) and the "charset" parameter MUST be set.

5.3.3 CPPA3 CPP Resources

Metadata resources MUST be identified and located using Uniform Resource Locators. The URL scheme MUST be “*http*” or “*https*” (case-insenstive). No specific additional constraints are placed on the URL format.

Implementations or user communities MAY adopt specific URL format conventions.

Metadata resources MUST be valid CPP XML documents conforming to the CPPA3 XML schema.

When used in conjunction with the metadata service location protocol defined in section 5.2, the URI retrieved from the DNS MUST be used to locate the resource.

5.3.4 Character Encoding

As a consequence of Section 4.3.3 of [REC-XML], "Character Encoding in Entities", which requires XML processors to support both the UTF-8 and UTF-16 character encodings, this protocol mandates that clients support both UTF-8 and UTF-16 character encodings.

To improve interoperability, the "charset" parameter of *Content-Type* HTTP header field MUST be used to determine the correct character encoding of the message.

A CPPA3 resource MUST be serialized using either UTF-8 or UTF-16 character encoding.

A CPPA3 resource MUST indicate the correct character encoding, using the "charset" parameter (case-insensitive).

A client MUST ignore, if present, the encoding declaration of the CPPA3 resource's XML declaration.

5.3.5 Access Control

The metadata service provider MAY require authentication of the requesting client consumer to access all or some of its metadata resources.

Specific authentication mechanisms are out of scope for this specification, but MAY include HTTP authentication [RFC7235] or TLS client authentication [RFC5246, RFC8446]. When using HTTP authentication, the HTTP 401 Unauthorized status code MUST be returned if the request has not been applied because it lacks valid authentication credentials for the target resource.

The metadata service provider MAY also return different content for all or some of its metadata resources, depending on whether or not the requesting consumer is authenticated and, if the consumer is authenticated, depending on the identity of the consumer.

As described in section 2.2.13, a CPPA3 CPP, and particular substructures of a CPPA3 CPP, MAY be annotated using the `allowed` and `denied` authorization attributes.

If at least one of the `allowed` and `denied` authorization attributes is present in a particular CPPA3 CPP resource, at some structural level in the CPP, if the consumer is successfully authenticated to the metadata service, and if the authentication consumer is identified as or representing a particular party, then the metadata service provider:

- MUST deny access to the resource if a toplevel `allowed` attribute is present and the identified or represented party is not a member of the referenced party identifier list.
- MUST deny access to the resource if a toplevel `denied` attribute is present and the identified or represented party is a member of the referenced party identifier list.

If the consumer is successfully authenticated to the metadata service, if the authenticated consumer is identified as or representing a particular party, and if either of the following situations apply:

- No `allowed` and `denied` authorization attributes are present at the toplevel.
- An `allowed` or `denied` authorization attribute is present at the toplevel and the identified or represented party is allowed access.

Then the CPPA3 metadata service provide MUST return a CPPA3 CPP for the requested resource that is derived from the resource as stored on the server by recursively walking the CPP XML tree structure, including sub-structure nodes at levels below the toplevel only if:

- No `allowed` and `denied` authorization attributes are present at the node.
- An `allowed` authorization attribute is present at the node and the identified or represented party is member of the referenced list.
- A `denied` authorization attribute is present at the node and the identified or represented party is not a member of the referenced list.

Depending on policy, a metadata service provider MAY provide access to unauthenticated, anonymous consumer clients. For CPP resources, and for any substructure of such resources, such anynomous consumers SHOULD be denied access if the (sub) resource has either an `allowed` and `denied` attribute.

If either no authentication was requested, or the consumer client is successfully authenticated, but is not authorized to access the CPPA3 CPP resource, the 403 (Forbidden) status code SHOULD be returned to indicate that the server understood the request but refuses to authorize it. Alternatively, the 404 (Not Found) status code MAY be used, if the origin server is not willing to disclose that the resource exists.

The HTTP 404 (Not Found) status code MUST also be used if the requested resource does not exist.

5.3.6 Using the Location Service with the Metadata Service

The output of the location service MAY be used as input for the metadata service, as shown in the following diagram.

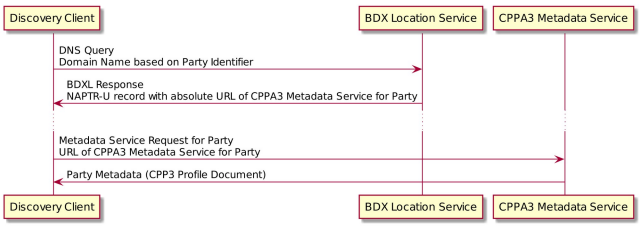

Figure 1: Location Service and Metadata Service

However, this is NOT REQUIRED if the URL for the Metadata Service for a party is shared with counterparties through some other mechanism.

5.4 CPPA3 Agreement Registration Service

5.4.1 Introduction

The CPPA3 Agreement Registration Service is a service using which two parties can agree bilaterally on an agreement encoded as a CPPA3 Collaboration Protocol Agreement (CPA) XML document. This section specifies the service and three actions. It also specifies an XML format using which reasons for rejection can be communicated.

The capability of a party to use the service can be expressed in CPPA3 Profile and Agreement XML documents. The service can be executed using a channel compatible with requester and responder.

5.4.2 Agreement Registration Service Flow

The overall flow of CPPA3 Agreement Registration is displayed in Figure 2.

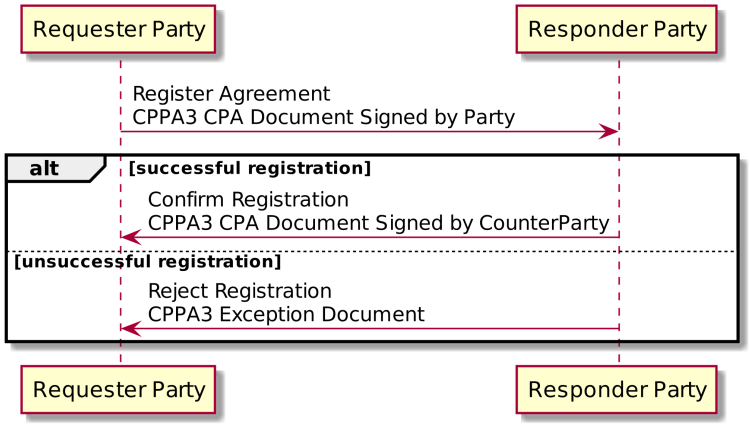

Figure 2: Agreement Registration Service

The counterparty accepts and confirms the proposal by signing the agreement and invoking the Confirm Agreement action on party's Agreement Registration service. The counterparty alternatively rejects the proposal and notifies party using the Reject Registration action.

The exchange consists of the following steps:

1. A party makes a registration request by signing a CPPA3 CPA agreement document and invoking the Register Agreement action on counterparty's Agreement Registration service. Party commits to the proposed agreement by signing its content at document level using an enveloped XML Signature [XMLDSIG-CORE, XMLDSIG-CORE1]. To facilitate matching of the CPA against a CPP of counterparty, the `ProfileIdentifier` of that CPP SHOULD be included in the presented CPA.
2. If the counterparty accepts the request, it returns a CPPA3 CPA that is identifical to the Agreement that was proposed in the request, except that it contains an enveloped XML Signature signed by the counterparty rather than the party. At this stage, the Agreement is accepted and both parties have a version of the agreement that is signed by the other party
3. If the counterparty does not accept the request, it communicates this to party by returning CPPA3 Exception document. This document explains the reasons for rejection in a structured XML format.

The Agreement Registration flow is similar to, but different from the ebCore Agreement Update flow [ebcore-au-v1.0]. For discussion, see section 5.6.

5.4.3 Agreement Registration in CPPA3 Profiles and Agreements

As any service, a party can indicate its capability to use the agreement registration service, in a requesting and/or responding role, in a CPPA3 Collaboration Protocol Profile (CPP) XML document. Two parties can configure exchange of Agreement Registration actions in a CPPA3 Collaboration Protocol Agreement (CPA) XML document.

For interoperability, this section specifies common CPPA3 structures and fixed values for CPPA3 elements and attributes for use with the CPPA3 Agreement Registration service. The following values MUST be used in a ServiceSpecification:

- The `PartyRole` (or `CounterPartyRole`) of the party making the registration request is set to *http://docs.oasis-open.org/ebcore/ns/cppa/v3.0/roles/requester*
- The `CounterPartyRole` (or `PartyRole`) of the party to which the registration request is made is set to *http://docs.oasis-open.org/ebcore/ns/cppa/v3.0/roles/responder*
- In the `ServiceBinding`, the `Service` element is set to *http://docs.oasis-open.org/ebcore/ns/AgreementRegistration/v1.0* The element does not have a type attribute.
- The `ServiceBinding` has three `ActionBinding` elements, corresponding to the three exchanges in Figure 2.
- The `ActionBinding` for the requesting action has a value *RegisterAgreement* for its `action` attribute.
- The `ActionBinding` for the responding action confirming the agreement registration has a value *ConfirmRegistration* for its `action` attribute.
- The `ActionBinding` for the alternative responding action rejection the agreement registration has a value *RejectRegistration* for its `action` attribute.

The following CPPA3 fragment illustrates the use of these values.

```
<cppa:ServiceSpecification>
    <cppa:PartyRole
        name="http://docs.oasis-open.org/ebcore/ns/cppa/v3.0/roles/requester"/>
    <cppa:CounterPartyRole
        name="http://docs.oasis-open.org/ebcore/ns/cppa/v3.0/roles/responder"/>
    <cppa:ServiceBinding>
        <cppa:Description xml:lang="en">The CPPA3 Agreement Registration service,
           in initiating mode</cppa:Description>
```

```
        <cppa:Service
    >http://docs.oasis-open.org/ebcore/ns/AgreementRegistration/v1.0</cppa:Service>
        <cppa:ActionBinding id="ab_1_s"
            action="RegisterAgreement" sendOrReceive="send" >
            <!-- One or more cppa:ChannelId elements -->
            <cppa:PayloadProfileId>pp_cppa3_cpa_proposed</cppa:PayloadProfileId>
        </cppa:ActionBinding>
        <cppa:ActionBinding id="ab_2_r" action="ConfirmRegistration"
            sendOrReceive="receive" replyTo="ab_1_s">
            <!-- One or more cppa:ChannelId elements -->
            <cppa:PayloadProfileId>pp_cppa3_cpa_accepted</cppa:PayloadProfileId>
        </cppa:ActionBinding>
        <cppa:ActionBinding id="ab_3_r" action="RejectRegistration"
            sendOrReceive="receive"  replyTo="ab_1_s">
            <!-- One or more cppa:ChannelId elements -->
            <cppa:PayloadProfileId>pp_cppa3_exception</cppa:PayloadProfileId>
        </cppa:ActionBinding>
    </cppa:ServiceBinding>
</cppa:ServiceSpecification>
```

In a CPPA3 CPP document, this fragment specifies the capability of the CPP party to make Agreement Registration requests and receive confirmations and rejections. To specify the ability to receive Agreement Registration requests and respond with confirmations or rejections, the values for `PartyRole` and `CounterPartyRole` are reversed.

In a CPPA3 CPA document, the fragment specifies the capability of the party in `PartyInfo` to make the request and receive the two types of responses and the capability of the party in `CounterPartyInfo` to receive the request and send the two types of responses. To specify the reverse roles for the `PartyInfo` and `CounterPartyInfo` parties, the values for `PartyRole` and `CounterPartyRole` are reversed.

If a single party is able to both make Agreement Registration requests (and receive responses to them) and receive requests (and respond to them), two `ServiceSpecification` elements are required, with mirrored substructures and values.

As indicated in the example using the XML comment, this specification does not provide a specific channel binding for the Agreement Registration actions. The common functionality of channels (section 2.2.7) and compatibility of channels (see section 3.4.11) applies.

As with any CPPA3 action, Agreement Registration actions MAY be bound to a delegation channel if a party outsources the service to a service provider (see sections 2.2.8 and 3.4.12). Such a service provider may provide all messaging for a party (i.e. both Agreement Registration messaging and other business messaging), or it may be a special-purpose matchmaking service provider that is not involved in other messaging.

The following CPPA3 fragment shows two `PayloadProfile` elements that can be used for the `RegisterAgreement` and `ConfirmRegistration` action values. Both express that these actions involve one business document `PayloadPart`, which is a CPPA3 `CPA` element.

```
<cppa:PayloadProfile id="pp_cppa3_cpa_proposed">
    <cppa:Description xml:lang="en">A version 3.0 CPPA3 CPA
      document</cppa:Description>
    <cppa:PayloadPart minOccurs="1" maxOccurs="1">
        <cppa:PartName>businessdocument</cppa:PartName>
        <cppa:Schema element="CPA"
            namespace="http://docs.oasis-open.org/ebcore/ns/cppa/v3.0" />
        <cppa:Signature>
            <cppa:SigningCertificateRef certId="party_a_signing_certificate"/>
        </cppa:Signature>
     </cppa:PayloadPart>
</cppa:PayloadProfile>

<cppa:PayloadProfile id="pp_cppa3_cpa_accepted">
    <cppa:Description xml:lang="en">A version 3.0 CPPA3 CPA
      document</cppa:Description>
    <cppa:PayloadPart minOccurs="1" maxOccurs="1">
        <cppa:PartName>businessdocument</cppa:PartName>
```



```
        <cppa:Schema element="CPA"
            namespace="http://docs.oasis-open.org/ebcore/ns/cppa/v3.0" />
        <cppa:Signature>
            <cppa:SigningTrustAnchorSetRef certId="a_party_trust_anchors"/>
        </cppa:Signature>
    </cppa:PayloadPart>
</cppa:PayloadProfile>
```

As the Agreement Registration service REQUIRES the use of XML Signature, the `PayloadPart` contains a `Signature` element. For outbound use, this element references the signing `Certificate` for the requesting party using an XML identifier reference. For inbound use, it references a `TrustAnchorSet` of certification authority certificates that the `Certificate` of the counterparty MUST chain to (see section 3.4.20) using another XML identifier reference.

5.4.4 Exception Document

The CPPA3 Agreement Registration Exception document structure is defined in the CPPA3 schema and its documentation. The normative Exception XML schema, which declares the *http://docs.oasis-open.org/ebcore/ns/cppa/v3.0* namespace, is specified in:

schema/exception.xsd

The documentation of this schema is embedded in this schema document and is a normative part of this specification. A valid CPPA3 Agreement Registration Exception document MUST be valid against the CPPA3 Agreement Registration Exception XML schema and MUST observe the definitions of syntax and semantics of the structures in the CPPA3 schema specified in the embedded schema documentation, including additional schema constraints not expressed in the XSD.

In a CPPA3 document, the use of the *Exception* document can be expressed by linking the action to a `PayloadProfile` that involves on `PayloadPart` XML document that is rooted by a CPPA3 `Exception` element.

```
<cppa:PayloadProfile id="pp_cppa3_exception">
    <cppa:Description xml:lang="en">An exception document reporting
      registration is not accepted</cppa:Description>
    <cppa:PayloadPart minOccurs="1" maxOccurs="1">
        <cppa:PartName>businessdocument</cppa:PartName>
        <cppa:Schema element="Exception"
           namespace="http://docs.oasis-open.org/ebcore/ns/exception/v3.0" />
    </cppa:PayloadPart>
</cppa:PayloadProfile>
```

The `Exception` document MAY include any number of `Error` elements to explain the reasons why the proposed Agreement was rejected. An has required `errorCode`, `shortDescription` and `severity` attributes. The allowed values for these attributes are presented in the following table:

Error Code	Short Description	Severity	Description or Semantics
AR:0001	InvalidRequest	Failure	The presented CPA is invalid. For example, it is not well-formed or not valid against the CPPA3 XML schema.
AR:0002	RequestRejected	Failure	The request is rejected for some other reason.
AR:0003	ProcessingError	Failure	An error occurred processing the request.
AR:0004	ServiceUnavailable	Failure	The request cannot be processed because the Agreement Registration service is not available.
AR:0005	ProfileNotFound	Failure	A Profile is referenced in `ProfileIdentifier` and either:

Error Code	Short Description	Severity	Description or Semantics
			1. The request is authorized but the profile could not be found. 2. The request is not authorized and Receiver is not willing to disclose that the Profile exists.
AR:0006	AuthorizationFailed	Failure	The request could not be authorized
AR:0007	NoMatch	Failure	The presented `CPA` does not match the referenced `CPP` (if a specific CPP is referenced using a `ProfileIdentifier`), or does not otherwise match receiver's capabilities.

Table 3: Agreement Registration Errors

5.5 Combining the Location, Metadata and Registration Services

The CPPA3 Metadata Service Location (see section 5.2), Metadata Service (see section 5.3) and Agreement Registration (see section 5.4) services can all be used independently or in combination. They are shown in use in combination in Figure 3.

- Party queries a BDX Location Service using a domain name corresponding to the party identifier for CounterParty.
- The BDX location service returns the URL of the metadata service for CounterParty.
- Party then queries the CPPA3 metadata service using the obtained URL.
- The CPPA3 metadata service returns a CPPA3 CPP document in response.
- Party supports the CPPA3 Agreement Registration service as requester. On inspection of the CPPA3 CPP document, party finds that CounterParty supports the CPPA3 Agreement Registration service as responder. Party constructs a CPPA3 Agreement document covering the services it wants to engage in with CounterParty. It MAY use CPPA3 Agreement Formation algorithm (see section 3) to construct this CPA document. The Agreement Registration service MAY itself be one of the services to be included in the Agreement.
- Party invokes the *RegisterAgreement* action on CounterParty's Agreement Registration service. This exchange may be involve a previously configured channel, one or two delegated agreement registration service providers, or a channel that does not require prior agreement.
- Counterparty inspects and accepts the proposed agreement. It MAY do this by using the Agreement Matching algorithm (see section 4) to verify that the presented CPA matches its CPP.
- At this stage, both parties need to deploy the agreed Agreement. How this is done is out of scope for this specification, as it may involve proprietary interfaces of messaging products, service management task, network managent tasks (e.g. firewall rules), or actions by service providers. Note that the `ActivationDate` of the Agreement MAY be in the future.
- Once the Agreement is activated on both sides, parties are able to use services that the agreement enables.

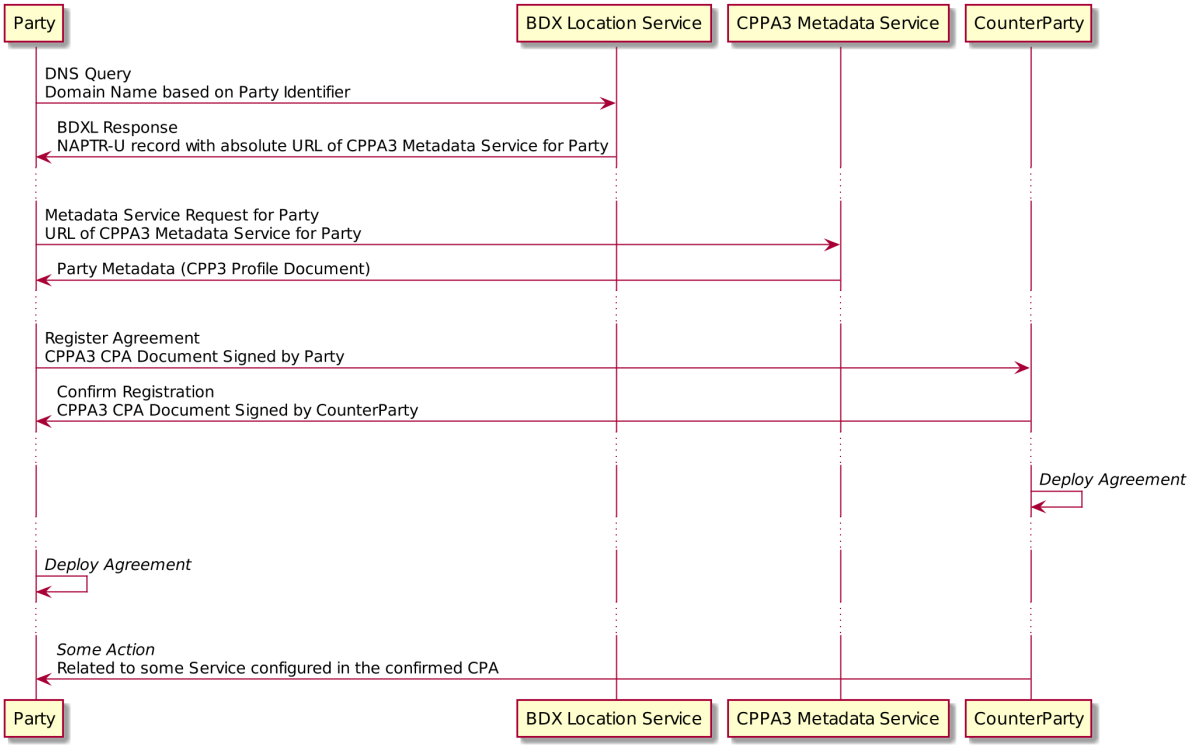

Figure 3 Combined use of CPPA3 Discovery and Registration Services

5.6 Relation to ebCore Agreement Update

The CPPA3 Agreement Registration Service provides a mechanism to deploy an agreement that is not based on an existing, previously deployed agreement using the CPPA3 CPA format. The ebCore Agreement Update specification [ebcore-au-v1.0] is a simpler mechanism that allows parties to update agreements by only exchanging changes. This is particularly useful for routine configuration changes such as updates of X.509 certificates.

Agreement Update also provides an agreement termination mechanism. This is useful to finish an agreement before the value set for its `ExpirationDate`.

6 Conformance

6.1 Conformance Targets

This specification defines 14 Conformance Targets:

- CPPA3 CPP Schema Conformance (applies to documents)
- CPPA3 CPA Schema Conformance (applies to documents)
- Default CPA Formation Conformance (applies to software implementations or services)
- Default CPA Matching Conformance (applies to software implementations or services)
- CPPA3 CPP Consumer Conformance (applies to B2B messaging software implementations or services)
- CPPA3 CPP Producer Conformance (applies to B2B messaging software implementations or services)
- CPPA3 CPA Consumer Conformance (applies to B2B message software implementations or services)
- CPPA3 CPA Producer Conformance (applies to B2B message software implementations or services)
- CPPA3 Metadata Service Location Client Conformance
- CPPA3 Metadata Service Location Server Conformance
- CPPA3 Metadata Service Client Conformance
- CPPA3 Metadata Service Client Conformance
- CPPA3 Agreement Registration Sender Conformance
- CPPA3 Agreement Registration Receiver Conformance

Note that these conformance targets are intended to cover common anticipated CPPA3 uses only and are not exclusive of other potential targets.

6.2 CPPA3 CPP Schema Conformance

In order for a CPPA3 `CPP` document to claim Core CPPA3 CPP Schema Conformance, the CPPA3 `CPP` document MUST be a valid `CPP` document as defined in section 2.3.1.

6.3 CPPA3 CPA Schema Conformance

In order for a CPPA3 `CPA` document to claim Core CPPA3 CPA Schema Conformance, the CPPA3 `CPA` document MUST be valid as defined in section 2.3.1.

6.4 Default CPA Formation Conformance

In order to claim Default CPA Formation Conformance, a software implementation or service that forms a CPPA3 `CPA` document from two CPPA3 `CPP` documents using the `http://docs.oasis-open.org/ebcore/cppa/v3.0/formation/default` formation method MUST follow the definition of default CPA formation specified in section 3.

All generated CPA documents MUST conform at “CPPA3 CPA Schema Conformance” level.

6.5 Default CPA-CPP Matching Conformance

In order to claim Default CPA-CPP Matching Conformance, a software implementation or service that matches a CPPA3 CPA document against a CPP document using the http://docs.oasis-open.org/ebcore/cppa/v3.0/matching/default formation method MUST follow the definition of default CPA-CPP matching specified in section 4.

6.6 CPPA3 CPP Consumer Conformance

In order to claim CPPA3 CPP Consumer Conformance, a B2B messaging software implementation or service MUST provide an interface that consumes a valid CPPA3 CPP document representing B2B messaging configuration parameters for a partner of a party and (possibly subject to additional constraints and possibly using additional parameters) updates the party's B2B partner configuration accordingly.

The software implementation or service MAY be restricted to a subset of CPPA3 CPP documents and MUST reject CPP documents using unsupported CPPA3 features. For example, a messaging product that only implements the AS2 messaging protocol, but not the AS3 protocol, could offer a CPPA3-based CPP (or CPA) import module to support partner configuration management. This product MUST support the CPPA3 AS2Channel and HTTPTransport elements as they are key to using AS2 but is NOT REQUIRED to support the CPPA3 AS3Channel and FTPTransport elements as these elements do not apply to AS2.

6.7 CPPA3 CPP Producer Conformance

In order to claim CPPA3 CPP Producer Conformance, a B2B messaging software implementation or service MUST provide an interface that (possibly subject to additional constraints and possibly using additional input parameters) produces a CPPA3 CPP document representing an export of a party's B2B messaging configuration. The software implementation or service MAY be restricted to a subset of CPPA3 CPP functionality, reflecting the feature set of the implementation that may not implement all functionality supported in CPPA3.

All generated CPP documents MUST conform at “CPPA3 CPP Schema Conformance” level.

6.8 CPPA3 CPA Consumer Conformance

In order to claim CPPA3 CPA Consumer Conformance, a B2B messaging software implementation or service MUST provide an interface that consumes one or more valid CPPA3 CPA documents representing agreed B2B messaging configuration parameters for a party and one or more of its partners and (possibly subject to additional constraints and possibly using additional input parameters) updates the party's B2B partner configuration accordingly.

The software implementation or service MAY be restricted to a subset of CPPA3 CPA documents and MUST reject documents using unsupported CPPA3 features. For an example, see above in section 6.6, CPPA3 CPP Consumer Conformance.

6.9 CPPA3 CPA Producer Conformance

In order to claim CPPA3 CPA Producer Conformance, a B2B messaging software implementation or service MUST provide an interface that (possibly subject to additional constraints and possibly using additional input parameter) produces one or more CPPA3 CPA documents representing an export of a party's B2B messaging partner configuration the party agreed with one or more partners.

All generated CPA documents MUST conform at “CPPA3 CPA Schema Conformance” level.

6.10 CPPA3 Metadata Service Location Client Conformance

In order to claim CPPA3 Metadata Service Location Client Conformance, an implementation MUST conform to the CPPA3 Metatadata Service Location protocol as specified in section 5.2 as a client.

6.11 CPPA3 Metadata Service Location Server Conformance

In order to claim CPPA3 Metadata Service Location Client Conformance, an implementation MUST conform to the CPPA3 Metatadata Service Location protocol as specified in section 5.2 as a server.

6.12 CPPA3 Metadata Service Client Conformance

In order to claim CPPA3 Metadata Service Location Client Conformance, an implementation MUST conform to the CPPA3 Metatadata Service protocol as specified in section 5.3 as a client.

6.13 CPPA3 Metadata Service Server Conformance

In order to claim CPPA3 Metadata Service Client Conformance, an implementation MUST conform to the CPPA3 Metatadata Service protocol as specified in section 5.3 as a server.

6.14 CPPA3 Agreement Registration Sender Conformance

In order to claim CPPA3 Agreement Registration Sender Conformance, an implementation MUST conform to the CPPA3 Agreement Registration protocol as specified in section 5.4 as a Sender.

6.15 CPPA3 Agreement Registration Receiver Conformance

In order to claim CPPA3 Agreement Registration Receiver Conformance, an implementation MUST conform to the CPPA3 Agreement Registration protocol as specified in section 5.4 as a Receiver.

Appendix A Examples

This specification comes with some sample CPP and CPA documents illustrating CPPA3 features. XML excerpts are edited for readability.

Appendix A.1 ebMS3 Channels and Channel Profiles

The following set of examples covers two parties in the energy sector that exchange EDIGAS XML documents. The examples relate to data exchanges involving Transmission System Operators in the European gas sector using the ENTSOG AS4 Usage Profile for TSOs [ENTSOGAS4].

- CPP for A
- CPP for B
- CPA for A-B

These CPP and CPA examples illustrate the use of a number of CPPA3 features, in particular:

- Service specifications and bindings of services and actions.
- Payload profiles and part properties.
- Support for the ebMS3 messaging protocol and the AS4 profile.
- The ability to define and reference channel profiles that provide defaults that can be overridden and can be expanded to full channel configurations.

In the referenced usage profile, for business services, party role values use values defined in an industry-specific code list. The following excerpt shows a party can perform the *ZSH* role to exchange messages with parties performing the *ZSO* role. Service values are similarly drawn from a service code list, which includes the *A09* value. The usage profile mandates a fixed *http://edigas.org/service* value of the service *type* attribute. Within services, in this instance all exchanges are profiled to use the AS4 default value for *action.* This is a choice made for the community. Alternatively, the action could be set to a value that reflects a specific business action. In the examples, the various business exchanges are differentiated using references to different payload profiles.

```
<cppa:ServiceSpecification>
  <cppa:PartyRole name="ZSH"/>
  <cppa:CounterPartyRole name="ZSO"/>
  <cppa:ServiceBinding>
    <cppa:Service type="http://edigas.org/service">A09</cppa:Service>
    <cppa:ActionBinding sendOrReceive="send"
        action="http://docs.oasis-open.org/ebxml-msg/as4/200902/action" id="ab_4_1">
      <cppa:ChannelId>ch_send</cppa:ChannelId>
      <cppa:PayloadProfileId>pp_AL3</cppa:PayloadProfileId>
    </cppa:ActionBinding>
  </cppa:ServiceBinding>
  <cppa:ServiceBinding>
    <cppa:Service type="http://edigas.org/service">A06</cppa:Service>
    <cppa:ActionBinding sendOrReceive="send"
        action="http://docs.oasis-open.org/ebxml-msg/as4/200902/action" id="ab_5_1">
      <cppa:ChannelId>ch_send</cppa:ChannelId>
      <cppa:PayloadProfileId>pp_ANN</cppa:PayloadProfileId>
    </cppa:ActionBinding>
    <!-- other action bindings in A06 omitted for brevity -->
  </cppa:ServiceBinding>
  <!-- other service bindings for this role pair omitted for brevity -->
</cppa:ServiceSpecification>
```

The payload profile referenced in the first shown action binding specifies that the exchange includes one business document payload part that has the *application/xml* MIME content type. This part has one mandatory property called *EDIGASDocumentType* which has a fixed value *AL3*.

```
<cppa:PayloadProfile id="pp_AL3">
  <cppa:PayloadPart maxOccurs="1" minOccurs="1">
    <cppa:PartName>businessdocument</cppa:PartName>
    <cppa:MIMEContentType>application/xml</cppa:MIMEContentType>
    <cppa:Property maxOccurs="1" minOccurs="1" name="EDIGASDocumentType"
       value="AL3"/>
  </cppa:PayloadPart>
</cppa:PayloadProfile>
```

References to XML schemas are allowed in CPPA3 payload profiles, but not included in the example.

In addition to business services the samples implement the ebMS3 “test” service defined in section 5.2.2 of ebMS3 [EBMS3CORE] and the ebCore Certificate Update service defined in the [ebcore-au-v1.0]. These services are not shown in the excerpts in this section.

In CPPA3 documents, actions are bound to channels. In the samples, they are bound to `ebMS3Channel` elements. The CPPA3 schema allows simple references to predefined and agreed channel profiles, or fine grained configuration of all features of the channel type, and combinations of the two.

A simple reference to a profile is shown in the following excerpt. It assumes parties have a shared understanding of *http://www.entsog.eu/AS4-USAGE-PROFILE/v3/UserMessageChannel* as an identifier of a particular usage profile.

```
<cppa:ebMS3Channel id="ch_receive" transport="tr_receive">
  <cppa:ChannelProfile>http://www.entsog.eu/AS4-USAGE-PROFILE/v3/UserMessageChannel
   </cppa:ChannelProfile>
</cppa:ebMS3Channel>
```

It is also possible to reference a channel profile but to override values specified in it. In the following example, a party references the same channel profile, but expresses that it supports two compression algorithms for the channel. The order indicates a preference for the *application/brotli* algorithm, but *application/gzip* is also supported for compatibility. (Note that the AS4 standard and the ENTSOG reference usage profile only support *application/gzip*. At the time of writing any support for Brotli compression would be a proprietary extension.) Due to how CPA formation works in CPPA3, the Brotli option could only be selected if the counter party also supports it.

```
<cppa:ebMS3Channel id="ch_receive" transport="tr_receive">
  <cppa:ChannelProfile>http://www.entsog.eu/AS4-USAGE-PROFILE/v3/UserMessageChannel
    </cppa:ChannelProfile>
  <cppa:Compression>
    <cppa:CompressionAlgorithm>application/brotli</cppa:CompressionAlgorithm>
    <cppa:CompressionAlgorithm>application/gzip</cppa:CompressionAlgorithm>
  </cppa:Compression>
</cppa:ebMS3Channel>
```

CPPA3 also allows for fine-grained specification of ebMS3 channels, which includes setting all individual processing mode parameters. The channel profile feature and the fine-grained configuration can be combined in CPPA3 processors by expanding the channel definitions to include defaults set for the channel profile. This expansion needs to also account for the fact that some channels rely on other channels (e.g. for receipts and errors), so configurations for those channels need to be added. The CPA included in the example illustrates this for receipt and error channels, and results in two `ebMS3Channel` elements.

```
<cppa:ebMS3Channel id="_SCFE" transport="_M7ID">
  <cppa:Description xml:lang="en">Channel formed from a_ch_receive (Channel for incoming
ENTSOG AS4 v3 User Messages) in ENTSOG AS4 Profile for Registered Network User 12 and
b_ch_send (Channel for outgoing ENTSOG AS4 v3 User Messages) in ENTSOG AS4 Profile for
Exchange Trader 14</cppa:Description>
  <cppa:ChannelProfile>http://www.entsog.eu/AS4-USAGE-PROFILE/v3/UserMessageChannel
     </cppa:ChannelProfile>
  <cppa:SOAPVersion>1.2</cppa:SOAPVersion>
  <cppa:WSSecurityBinding>
    <cppa:WSSVersion>1.1</cppa:WSSVersion>
    <cppa:Signature>
```



```
      <cppa:SignatureAlgorithm>https://www.w3.org/2001/04/xmldsig-more#rsa-sha256
         </cppa:SignatureAlgorithm>
      <cppa:DigestAlgorithm
            >http://www.w3.org/2001/04/xmlenc#sha256</cppa:DigestAlgorithm>
      <cppa:SigningCertificateRef certId="_A4RHGV"/>
      <cppa:SignElements>
        <cppa:Expression>{http://www.w3.org/2003/05/soap-envelope}Body
            </cppa:Expression>
        <cppa:Expression>{http://www.w3.org/2003/05/soap-envelope}Header/
          {http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/}Messaging
        </cppa:Expression>
      </cppa:SignElements>
      <cppa:SignAttachments>true</cppa:SignAttachments>
    </cppa:Signature>
    <cppa:Encryption>
      <cppa:KeyEncryption>
        <cppa:EncryptionAlgorithm> http://www.w3.org/2009/xmlenc11#rsa-oaep
          </cppa:EncryptionAlgorithm>
        <cppa:MaskGenerationFunction>http://www.w3.org/2009/xmlenc11#mgf1sha256
         </cppa:MaskGenerationFunction>
        <cppa:DigestAlgorithm>http://www.w3.org/2001/04/xmlenc#sha256
          </cppa:DigestAlgorithm>
      </cppa:KeyEncryption>
      <cppa:EncryptionAlgorithm>http://www.w3.org/2009/xmlenc11#aes128-gcm
         </cppa:EncryptionAlgorithm>
      <cppa:EncryptAttachments>true</cppa:EncryptAttachments>
      <cppa:EncryptionCertificateRef certId="_7DDK72"/>
    </cppa:Encryption>
  </cppa:WSSecurityBinding>
  <cppa:AS4ReceptionAwareness>
    <cppa:DuplicateHandling>
      <cppa:DuplicateElimination>true</cppa:DuplicateElimination>
      <cppa:PersistDuration>P10D</cppa:PersistDuration>
    </cppa:DuplicateHandling>
    <cppa:RetryHandling>
      <cppa:Retries>15</cppa:Retries>
      <cppa:ExponentialBackoff>
        <cppa:Ceiling>PT1H</cppa:Ceiling>
      </cppa:ExponentialBackoff>
      <cppa:RetryInterval>PT30S</cppa:RetryInterval>
    </cppa:RetryHandling>
  </cppa:AS4ReceptionAwareness>
  <cppa:ErrorHandling>
    <cppa:DeliveryFailuresNotifyProducer>true</cppa:DeliveryFailuresNotifyProducer>
    <cppa:ProcessErrorNotifyConsumer>true</cppa:ProcessErrorNotifyConsumer>
    <cppa:ProcessErrorNotifyProducer>true</cppa:ProcessErrorNotifyProducer>
    <cppa:ReceiverErrorsReportChannelId>_MOYD</cppa:ReceiverErrorsReportChannelId>
  </cppa:ErrorHandling>
  <cppa:ReceiptHandling>
    <cppa:ReceiptChannelId>_MOYD</cppa:ReceiptChannelId>
  </cppa:ReceiptHandling>
  <cppa:Compression>
    <cppa:CompressionAlgorithm>application/gzip</cppa:CompressionAlgorithm>
  </cppa:Compression>
</cppa:ebMS3Channel>
<cppa:ebMS3Channel id="_MOYD" asResponse="true">
  <cppa:ChannelProfile>http://www.entsog.eu/AS4-USAGE-PROFILE/v3/
    SignalMessageChannel</cppa:ChannelProfile>
  <cppa:SOAPVersion>1.2</cppa:SOAPVersion>
  <cppa:WSSecurityBinding>
    <cppa:WSSVersion>1.1</cppa:WSSVersion>
    <cppa:Signature>
      <cppa:SignatureAlgorithm>https://www.w3.org/2001/04/xmldsig-more#rsa-sha256
      </cppa:SignatureAlgorithm>
        <cppa:DigestAlgorithm>http://www.w3.org/2001/04/xmlenc#sha256
           </cppa:DigestAlgorithm>
      <cppa:SigningCertificateRef certId="_LPNSIF"/>
      <cppa:SignElements>
        <cppa:Expression>{http://www.w3.org/2003/05/soap-envelope}Body
          </cppa:Expression>
```



```
        <cppa:Expression>{http://www.w3.org/2003/05/soap-envelope}Header/
        {http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/}Messaging
           </cppa:Expression>
      </cppa:SignElements>
    </cppa:Signature>
  </cppa:WSSecurityBinding>
</cppa:ebMS3Channel>
```

Appendix A.2 EDIINT AS2, Delegation, Payload Versions

The following set of CPP and CPA examples covers two parties, A and B, which can act as Buyer and Seller, respectively.

- CPP for Party A, Buyer
- CPP for Party B, Seller
- CPA for Parties A and B

These CPP and CPA samples illustrate the use of a number of CPPA3 features, in particular:

- the support for EDIINT AS2,
- automated selection of compatible channels and payload profiles in unification and matching,
- payload and protocol versioning,
- finegrained X.509 policy and PKI support,
- the concept of channel delegation, which enables support of so-called three-corner and four-corner topologies.

As shown in its CPP, Party A can participate as a Buyer in an *OrderingBilling* service. It can send UBL Orders and receive OrderResponse documents. For both, it supports both the 2.1 (preferred) and the 2.0 versions of the UBL standard schemas. This information is described in `PayloadProfile` elements, that are referenced from the OrderBilling `ActionBinding` elements.

```
<cppa:ServiceSpecification>
    <cppa:PartyRole name="Buyer"/>
    <cppa:CounterPartyRole name="Seller"/>
    <cppa:ServiceBinding>
        <cppa:Service>OrderingBilling</cppa:Service>
        <cppa:ActionBinding id="ab1" action="SubmitOrder" sendOrReceive="send">
           <cppa:ChannelId>ch1</cppa:ChannelId>
           <cppa:ChannelId>ch5</cppa:ChannelId>
           <cppa:PayloadProfileId>order2.1</cppa:PayloadProfileId>
           <cppa:PayloadProfileId>order2.0</cppa:PayloadProfileId>
        </cppa:ActionBinding>
        <cppa:ActionBinding id="ab2" action="ConfirmOrder" sendOrReceive="receive">
           <cppa:ChannelId>ch3</cppa:ChannelId>
           <cppa:ChannelId>ch6</cppa:ChannelId>
           <cppa:PayloadProfileId>orderresponse2.1</cppa:PayloadProfileId>
           <cppa:PayloadProfileId>orderresponse2.0</cppa:PayloadProfileId>
        </cppa:ActionBinding>
    </cppa:ServiceBinding>
</cppa:ServiceSpecification>
```

As shown in its CPP, Party B can also participate in the *OrderingBilling* service, in the complementary role of Seller. B is capable of receiving (respectively, sending) the actions that A is capable of sending (respectively, receiving). Unlike A, it only supports the version 2.0 schema of UBL. In CPA formation, the only compatible option is therefore this 2.0 schema.

Party A operates a B2B messaging server that implements EDIINT AS2. By linking the `ActionBinding` elements to `AS2Channel` configurations, A expresses its capability to send and receive the UBL documents in the OrderingBilling service using signed EDIINT AS2 messages. Note that the binding of message exchange to channels is done at the level of actions. This means that A could use AS2 for only a

subset of its exchanges and some other protocol for other exchanges, or use different AS2 configurations (which, for example, could differ on whether or not features like signing or encryption are used) for different exchanges.

The leaf signing certificate used by A in outbound signed AS2 messages is included in its CPP and derived CPAs and referenced from the `AS2Channel` using a `SigningCertificateRef` element. This allows recipients to validate that the correct certificate is used. For messaging, there are no hard-wired dependencies on or assumptions about PKI in CPPA3.

```
<cppa:AS2Channel id="ch1" transport="tr1">
    <cppa:Signature>
        <cppa:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1
          </cppa:SignatureAlgorithm>
        <cppa:DigestAlgorithm>http://www.w3.org/2000/09/xmldsig#sha1
          </cppa:DigestAlgorithm>
        <cppa:SigningCertificateRef certId="asigningcert"/>
    </cppa:Signature>
    <cppa:ReceiptHandling>
        <cppa:ReceiptChannelId>ch2</cppa:ReceiptChannelId>
    </cppa:ReceiptHandling>
</cppa:AS2Channel>
```

Party A's `AS2Channel` configurations for sending and receiving business documents include a `ReceiptHandling` element that configures signed asynchronous MDNs. There are no MDNs for MDNs, so the `AS2Channel` configurations for MDNs do not themselves include any `Receipt-Handling` subelements.

In its AS2 channels, Party A expects inbound MDNs to be signed by the counterparty. If A wanted, it could use a `SigningTrustAnchorSetRef` to express that, for inbound MDNs, it only accepts certificates from a particular Certification Authority (or list of such CAs). In that case, default CPA formation would check that A's certificate chains to one of those trust anchors, as explained in section 3.4.20, and unification would fail if there is no matching CA root certificate.This is not the case for A in the sample CPP. A does however express, using a `SigningCertificatePolicySetRef`, that it expects certificates to meet a particular certificate policy (in this case one identified using the *2.5.29.32.0* policy OID). These constraints are also enforced in CPA formation.

```
<cppa:Signature>
    <cppa:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1
        </cppa:SignatureAlgorithm>
    <cppa:DigestAlgorithm>http://www.w3.org/2000/09/xmldsig#sha1
        </cppa:DigestAlgorithm>
    <cppa:SigningCertificatePolicySetRef setId="all_issuance_policies"/>
</cppa:Signature>
```

Note that, as these constraints are expressed for individual channels, a party can express different constraints for particular services. E.g. for one service, a particular PKI or policy set may be mandatory, whereas no or other constraints may apply for another service.

In addition to operating its own AS2 server, Party A is also registered with a third party service provider C, to which it is able to delegate both inbound and outbound messaging. This means that C may send UBL Orders to other parties on behalf of A and that other Seller parties may send UBL OrderResponse documents intended for A to C. If it wanted, A could offer even more alternative communication options, e.g. using messaging protocols other than AS2, or using other alternative service providers than C. The CPPA3 data model is very flexible and allows for fine-grained configurations.

Party B does not operate is own AS2 messaging server, but fortunately it is also registered with C. It can receive from C the documents that A can send to C and can send to C the documents that A can receive from C, meaning that a complete document exchange for the *OrderingBilling* service is possible between the two parties using the delegated communication. This is sometimes referred to as a “three corner model”, and is offered by e-commerce trading platform hubs.

If B would delegate messaging not to C, but to another service provider D, then communication between A and B could still be possible if C and D are connected for the OrderingBilling documents. This is sometimes referred to as the “four corner model”. The configuration of the message exchange between C

and D is not itself described in the CPPs of A and B. So if, for example, C and D today are using AS2 for their communication but are migrating to AS4, neither A nor B needs to update its profile and the agreement between A and B also does not need to be changed.

Appendix A.3 Extensibility

The following XML schema and XML instance sample sets illustrate extensibility of the CPPA3 schema.

The first example set illustrates transport extensibility. The schema defines an `AvianCarrier` transport element.

- XML schema illustrating transport extensibility
- Sample XML instance using the extension element

The second example set illustrates extensibility using an extension `PartyAddress` element that instantiates the optional `PartyInfoExtension` element. The XML schema also imports the xAL schema that is part of the OASIS Customer Information Quality Specifications Version 3.0 [CIQ3].

- XML schema illustrating Party Info extensibility
- Sample XML instance using the extension element

Appendix B Acknowledgments

This specification was created in the OASIS ebCore Technical Committee, whose voting members at the time of writing included the following individuals:

Participants:

Albert Kappe, Ministerie van BZK/UBR/I-Interim Rijk
Dale Moberg, Sonnenglanz Consulting
Erlend Klakegg Bergheim, Difi-Agency for Public Management and eGovernment
Ernst Jan van Nigtevecht, Sonnenglanz Consulting
Pim van der Eijk, Sonnenglanz Consulting
Sander Fieten, Individual
Theo Kramer, Flame Computing Enterprises
Torsten Robert Kirschner, NAV

The TC also acknowledges the input of former ebCore TC members:

Cristiano Novelli
David Webber
Farrukh Najmi
Monica Martin
Pete Wenzel
Sacha Schlegel

Appendix C Revision History

Revision	Date	Editor	Changes Made
WD01	2016-10-02	PvdE	First Draft
WD02	2016-11-01	PvdE	Editorial fixes. XKMS resolution in CPA formation. Attribute unification. AU is not a normative reference. Start with support for the SAML conformance clause. New href attribute on ProfileIdentifier for use in CPAs. Clarified that some introductory sections are not normative. Added WSS-Username-Token profile to references. Described the `ChannelProfile` feature.
WD03	2016-12-17	PvdE	Describe some new features in the schema, and impact on formation: - Payload signing and encryption (schema #7) - Authorization attributes (schema #8) Some editorial fixes.
WD04	2017-01-31	PvdE	Describe updates and new features in schema revisions #9 to #12: - SAML token and `SAMLKeyConfirmed-SubjectToken`. - IDP registrations and set of and references to registrations. Other: - CA defined as Certification Authority - Some missing bibliographic references added. - Editorial.
WD05	2017-03-19	PvdE	- IETF and W3C references taken from OASIS lists at http://docs.oasis-open.org/templates/w3c-recommendations-list/w3c-recommendations-list.html and http://docs.oasis-open.org/templates/ietf-rfc-list/ietf-rfc-list.html - Updated HTTP 1.1 reference from obsolete IETF 2616 to RFC 7230. - Note that activation and expiration of CPPs must be synchronized with validity interval of certificates. - Updates for schema revision #13: - `CompressionType` sub-element instead of attribute. - New optional element `SignatureFormat`, added for EDIINT. - New elements `AS1Channel`, `AS2Channel`, `AS3Channel`. Easier for conformance clauses than `EDIINTChannel`, and functionality is slightly different beyond transport. - New section 2.3.3 on bibliographic references

			in XML Schema. - New section 3.4.25 on WS-A `From`. (Unlikely requirement, but completes coverage of Web Services specifications). - New section 3.4.26 on intervals for transport restart and for joining. Schema #14 - New section 2.2.8 on the new CPPA3 delegation feature. - New section 3.4.12 on unification for delegation. - Removed the `Compressed*` elements, and `Compression` is now just another `ChannelFeature`. - New `CertificateDefaults` element.
WD06	2017-04-02	PvdE	Schema #15 (2017-03-27) and #16 (2017-04-02): - New chapter 2.3.4 on schema extensibility. - Description of IPv4 and IPv6 support features and related constraints in unification in new section 3.4.16. - “Certification Authority”, naming consistency. - “Content Coding” support for HTTP. - “HTTP Version” support for HTTP. - Editorial. - Fixed some missing coverage of elements in 3.4.2. - Added bibliographic entries for FTP and SMTP RFCs and MTOM and XOP W3C Recommendations.
WD07	2017-07-26	PvdE	- Updated S/MIME reference to current 3.2 version, relevant for ebMS2. - When matching service specifications, presence and values of ebBP attributes must match. - `TrustAnchor` renamed to `TrustAnchorSet`. - `*TrustAnchorRef` renamed to `*TrustAnchorSetRef` - `CanonicalizationMethod`added to unification of `CanonicalizationMethod` and `*CertificateRef` is covered only in X.509 section. - New `*CertificateRequired` elements in schema #19. - Definition of validity in section 2.3.1 - New conformance section 6. - Different channel types are listed in 2.2.7.
WD08	2017-08-25	PvdE	New section 4, describing the CPA-CPP “match” function.
WD09	2017-11-16	PvdE	Matching:

			- Describe matching and delegation in extended section 4.2.9. - Describe matching and authorization in new section 4.2.3. Extensibility: - Descriptions of added extensibility elements and types. Payload Profiling: - Update to allow alternative payload profile references - Describing how unification and matching of `PayloadProfileId` elements works. Editorial: - Examples adapted for removal of `DataEncryption` container. New AMQP messaging support: - AMQP messaging, transport and security. - SASL feature for AMQP. Transports: - New WebSocket transport support. - New AMQP transport - New SFTP transport. - Note that a `Transport` is required to have an `Endpoint` after unification. Security tokens: - SSH key support for SFTP TLS improvements: - STARTTLS feature added. - TLS 1.2 Server Name Indication feature. Extensive documentation of sample extensions in new appendices.
WD10	2018-03-29	PvdE	Describe use of the authorization attributes to control visibility in section 2.2.13. Changed the order of two appendices Added Citation Format. New chapter 5, CPPA3 Metadata Service Location and Metadata Service. It describes the use of the BDX Location standard for service location and provides an HTTP-based metadata service binding to retrieve CPPA3 CPPs. The CPPA3 authorization/visibility features allow a party to constrain access to (parts of) a CPP to its authorized counterparties.

WD11	2018-09-28	PvdE	Added conformance clauses for metadata service location and metadata service, for client and server implementations. Added reference for TLS 1.3. Changed the extensibility example from JSON RPC to Avian Carrier.
WD 12	2018-12-27	PvdE	Updated extensibility example for Avian Carrier. Also added a second extensibility that uses OASIS CIQ to specify the party address. Handled some schema changes for compression and split/join. Include AlternateChannelId to list of indirectly used channels. New chapter 5.4 for CPPA3 Agreement Registration Service and corresponding Conformance Targets