Internet Engineering Task Force (IETF)
Request for Comments: 6503
Category: Standards Track
ISSN: 2070-1721

M. Barnes
Polycom
C. Boulton
NS-Technologies
S. Romano
University of Napoli
H. Schulzrinne
Columbia University

March 2012

Centralized Conferencing Manipulation Protocol

Abstract

The Centralized Conferencing Manipulation Protocol (CCMP) allows a Centralized Conferencing (XCON) system client to create, retrieve, change, and delete objects that describe a centralized conference. CCMP is a means to control basic and advanced conference features such as conference state and capabilities, participants, relative roles, and details. CCMP is a stateless, XML-based, client server protocol that carries, in its request and response messages, conference information in the form of XML documents and fragments conforming to the centralized conferencing data model schema.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6503.

Table of Contents

1. Introduction 4
2. Conventions and Terminology 5
3. XCON Conference Control System Architecture 5
   3.1. Conference Objects 7
   3.2. Conference Users 7
4. Protocol Overview 8
   4.1. Protocol Operations 9
   4.2. Data Management 10
   4.3. Data Model Compliance 11
   4.4. Implementation Approach 12
5. CCMP Messages 13
   5.1. CCMP Request Message Type 13
   5.2. CCMP Response Message Type 15
   5.3. Detailed Messages 17
        5.3.1. blueprintsRequest and blueprintsResponse 20
        5.3.2. confsRequest and confsResponse 22
        5.3.3. blueprintRequest and blueprintResponse 24
        5.3.4. confRequest and confResponse 26
        5.3.5. usersRequest and usersResponse 30
        5.3.6. userRequest and userResponse 32
        5.3.7. sidebarsByValRequest and sidebarsByValResponse 37
        5.3.8. sidebarByValRequest and sidebarByValResponse 39
        5.3.9. sidebarsByRefRequest and sidebarsByRefResponse 42
        5.3.10. sidebarByRefRequest and sidebarByRefResponse 44
        5.3.11. extendedRequest and extendedResponse 47
        5.3.12. optionsRequest and optionsResponse 49
   5.4. CCMP Response Codes 53
6. A Complete Example of CCMP in Action 57
   6.1. Alice Retrieves the Available Blueprints 58
   6.2. Alice Gets Detailed Information about a Specific Blueprint 60

6.3. Alice Creates a New Conference through a Cloning Operation 62
6.4. Alice Updates Conference Information 65
6.5. Alice Inserts a List of Users into the Conference Object ..66
6.6. Alice Joins the Conference 68
6.7. Alice Adds a New User to the Conference 70
6.8. Alice Asks for the CCMP Server Capabilities 72
6.9. Alice Makes Use of a CCMP Server Extension 75
7. Locating a Conference Server 78
8. Managing Notifications 79
9. HTTP Transport 80
10. Security Considerations 82
10.1. Assuring That the Proper Conference Server Has Been Contacted 83
10.2. User Authentication and Authorization 84
10.3. Security and Privacy of Identity 85
10.4. Mitigating DoS Attacks 86
11. XML Schema 87
12. IANA Considerations 105
12.1. URN Sub-Namespace Registration 105
12.2. XML Schema Registration 106
12.3. MIME Media Type Registration for 'application/ccmp+xml' 106
12.4. DNS Registrations 107
12.4.1. Registration of a Conference Server Application Service Tag 108
12.4.2. Registration of a Conference Server Application Protocol Tag for CCMP 108
12.5. CCMP Protocol Registry 108
12.5.1. CCMP Message Types 109
12.5.2. CCMP Response Codes 111
13. Acknowledgments 113
14. References 113
14.1. Normative References 113
14.2. Informative References 114
Appendix A. Evaluation of Other Protocol Models and Transports Considered for CCMP 116
A.1. Using SOAP for CCMP 117
A.2. A RESTful Approach for CCMP 117

1. Introduction

"A Framework for Centralized Conferencing" [RFC5239] (XCON framework) defines a signaling-agnostic framework, naming conventions, and logical entities required for building advanced conferencing systems. The XCON framework introduces the conference object as a logical representation of a conference instance, representing the current state and capabilities of a conference.

The Centralized Conferencing Manipulation Protocol (CCMP) defined in this document allows authenticated and authorized users to create, manipulate, and delete conference objects. Operations on conferences include adding and removing participants, changing their roles, as well as adding and removing media streams and associated endpoints.

CCMP implements the client-server model within the XCON framework, with the conferencing client and conference server acting as client and server, respectively. CCMP uses HTTP [RFC2616] as the protocol to transfer requests and responses, which contain the domain-specific XML-encoded data objects defined in [RFC6501] "Conference Information Data Model for Centralized Conferencing (XCON)".

Section 2 clarifies the conventions and terminology used in the document. Section 3 provides an overview of the conference control functionality of the XCON framework, together with a description of the main targets CCMP deals with, namely conference objects and conference users. A general description of the operations associated with protocol messages is given in Section 4 together with implementation details. Section 5 delves into the details of specific CCMP messages. A complete, non-normative, example of the operation of CCMP, describing a typical call flow associated with conference creation and manipulation, is provided in Section 6. A survey of the methods that can be used to locate a conference server is provided in Section 7, and Section 8 discusses potential approaches to notifications management. CCMP transport over HTTP is highlighted in Section 9. Security considerations are presented in Section 10. Finally, Section 11 provides the XML schema.

2. Conventions and Terminology

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

In addition to the terms defined in "A Framework for Centralized Conferencing" [RFC5239], this document uses the following terms and acronyms:

XCON-aware client: An XCON conferencing system client that is able to issue CCMP requests.

First-Party Request: A request issued by the client to manipulate its own conferencing data.

Third-Party Request: A request issued by a client to manipulate the conference data of another client.

3. XCON Conference Control System Architecture

CCMP supports the XCON framework. Figure 1 depicts a subset of the "Conferencing System Logical Decomposition" architecture from the XCON framework document. It illustrates the role that CCMP assumes within the overall centralized architecture.

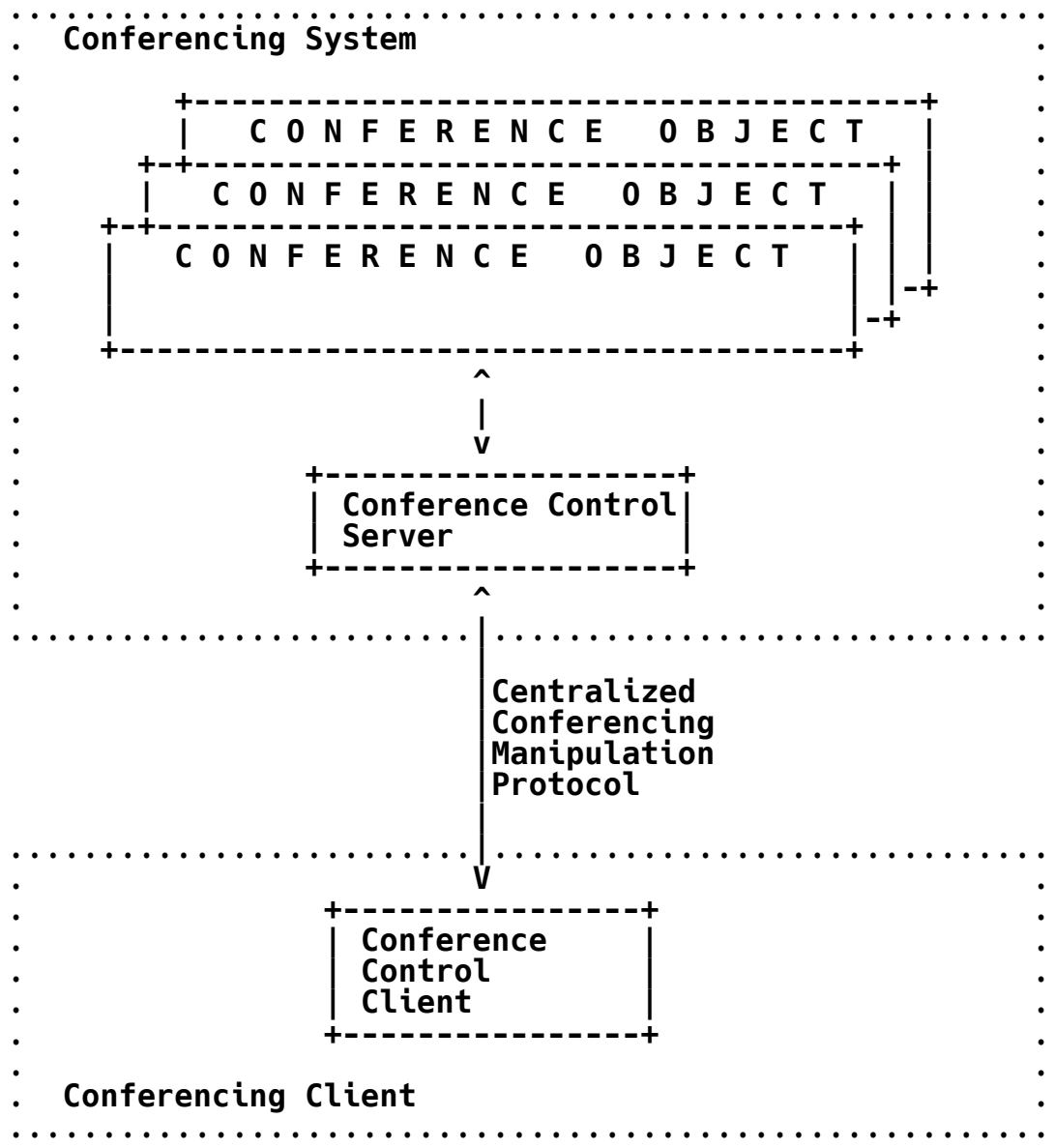


Figure 1: Conferencing Client Interaction

The Centralized Conferencing Manipulation Protocol (CCMP) allows the conference control client (conferencing client) to interface with the conference object maintained by the conferencing system, as depicted in Figure 1. Note that additional functionality of the conferencing client and conferencing system is discussed in the XCON framework and related documents.

This section provides details of the identifiers REQUIRED to address and manage the clients associated with a conferencing system using CCMP.

3.1. Conference Objects

Conference objects feature a simple dynamic inheritance-and-override mechanism. Conference objects are linked into a tree known as a "cloning tree" (see Section 7.1 of [RFC5239]). Each cloning tree node inherits attributes from its parent node. The roots of these inheritance trees are conference templates also known as "blueprints". Nodes in the inheritance tree can be active conferences or simply descriptions that do not currently have any resources associated with them (i.e., conference reservations). An object can mark certain of its properties as unalterable, so that they cannot be overridden. Per the framework, a client may specify a parent object (a conference or blueprint) from which to inherit values when a conference is created using the conference control protocol.

Conference objects are uniquely identified by the XCON-URI within the scope of the conferencing system. The XCON-URI is introduced in the XCON framework and defined in the XCON common data model.

Conference objects are comprehensively represented through XML documents compliant with the XML schema defined in the XCON data model [RFC6501]. The root element of such documents, called <conference-info>, is of type "conference-type". It encompasses other XML elements describing different conference features and users as well. Using CCMP, conferencing clients can use these XML structures to express their preferences in creating or updating a conference. A conference server can convey conference information back to the clients using the XML elements.

3.2. Conference Users

Each conference can have zero or more users. All conference participants are users, but some users may have only administrative functions and do not contribute or receive media. Users are added one user at a time to simplify error reporting. When a conference is cloned from a parent object, users are inherited as well, so that it is easy to set up a conference that has the same set of participants or a common administrator. The conference server creates individual users, assigning them a unique conference user identifier (XCON-USERID). The XCON-USERID as identifier of each conferencing system client is introduced in the XCON framework and defined in the XCON

common data model. Each CCMP request, with an exception pointed out in Section 5.3.6 representing the case of a user at his first entrance in the system as a conference participant, must carry the XCON-USERID of the requestor in the proper <confUserID> parameter.

The XCON-USERID acts as a pointer to the user's profile as a conference actor, e.g., her signaling URI and other XCON protocol URIs in general, her role (moderator, participant, observer, etc.), her display text, her joining information, and so on. A variety of elements defined in the common <conference-info> element as specified in the XCON data model are used to describe the users related to a conference including the <users> element, as well as each <user> element included within it. For example, it is possible to determine how a specific user expects and is allowed to join a conference by looking at the <allowed-users-list> in <users>: each <target> element involved in such a list represents a user and shows a 'method' attribute defining how the user is expected to join the conference, i.e., "dial-in" for users that are allowed to dial, "dial-out" for users that the conference focus will be trying to reach (with "dial-in" being the default mode). If the conference is currently active, dial-out users are contacted immediately; otherwise, they are contacted at the start of the conference. CCMP, acting as the conference control protocol, provides a means to manipulate these and other kinds of user-related features.

As a consequence of an explicit user registration to a specific XCON conferencing system, conferencing clients are usually provided (besides the XCON-USERID) with log-in credentials (i.e., username and password). Such credentials can be used to authenticate the XCON-aware client issuing CCMP requests. Thus, both username and password should be carried in a CCMP request as part of the "subject" parameter whenever a registered conferencing client wishes to contact a CCMP server. CCMP does not maintain a user's subscriptions at the conference server; hence, it does not provide any specific mechanism allowing clients to register their conferencing accounts. The "subject" parameter is just used for carrying authentication data associated with pre-registered clients, with the specific registration modality outside the scope of this document.

4. Protocol Overview

CCMP is a client-server, XML-based protocol for user creation, retrieval, modification, and deletion of conference objects. CCMP is a stateless protocol, such that implementations can safely handle transactions independently from each other. CCMP messages are XML documents or XML document fragments compliant with the XCON data model representation [RFC6501].

Section 4.1 specifies the basic operations that can create, retrieve, modify, and delete conference-related information in a centralized conference. The core set of objects manipulated by CCMP includes conference blueprints, the conference object, users, and sidebars.

Each operation in the protocol model, as summarized in Section 4.1, is atomic and either succeeds or fails as a whole. The conference server MUST ensure that the operations are atomic in that the operation invoked by a specific conferencing client completes prior to another client's operation on the same conference object. While the details for this data locking functionality are out of scope for the CCMP specification and are implementation specific for a conference server, some core functionality for ensuring the integrity of the data is provided by CCMP as described in Section 4.2.

While the XML documents that are carried in CCMP need to comply with the XCON data model, there are situations in which the values for mandatory elements are unknown by the client. The mechanism for ensuring compliance with the data model in these cases is described in Section 4.3.

CCMP is completely independent from underlying protocols, which means that there can be different ways to carry CCMP messages from a conferencing client to a conference server. The specification describes the use of HTTP as a transport solution, including CCMP requests in HTTP POST messages and CCMP responses in HTTP 200 OK replies. This implementation approach is further described in Section 4.4.

4.1. Protocol Operations

The main operations provided by CCMP belong in four general categories:

create: for the creation of a conference object, a conference user, a sidebar, or a blueprint.

retrieve: to get information about the current state of either a conference object (be it an actual conference, a blueprint, or a sidebar) or a conference user. A retrieve operation can also be used to obtain the XCON-URIs of the current conferences (active or registered) handled by the conferencing server and/or the available blueprints.

update: to modify the current features of a specified conference or conference user.

delete: to remove from the system a conference object or a conference user.

Thus, the main targets of CCMP operations are as follows:

- conference objects associated with either active or registered conferences,
- conference objects associated with blueprints,
- conference objects associated with sidebars, both embedded in the main conference (i.e., <entry> elements in <sidebars-by-value>) and external to it (i.e., whose XCON-URIs are included in the <entry> elements of <sidebars-by-ref>),
- <user> elements associated with conference users, and
- the list of XCON-URIs related to conferences and blueprints available at the server, for which only retrieval operations are allowed.

4.2. Data Management

The XCON framework defines a model whereby the conference server centralizes and maintains the conference information. Since multiple clients can modify the same conference objects, a conferencing client might not have the latest version of a specific conference object when it initiates operations. To determine whether the client has the most up-to-date conference information, CCMP defines a versioning approach. Each conference object is associated with a version number. All CCMP response messages containing a conference document (or a fragment thereof) MUST contain a <version> parameter. When a client sends an update message to the server, which includes modifications to a conference object, if the modifications are all successfully applied, the server MUST return a response, with a <response-code> of "200", containing the version number of the modified object. With this approach, a client working on version "X" of a conference object that receives a response, with a <response-code> of "200", with a version number that is "X+1" can be certain that the version it manipulated was the most up to date. However, if the response contains a version that is at least "X+2", the client knows that the object modified by the server was more up to date than the object the client was manipulating. In order to ensure that the client always has the latest version of the modified object, the client can send a request to the conference server to retrieve the conference object. The client can then update the relevant data elements in the conference object prior to invoking a specific operation. Note that a client subscribed to the XCON event package

[RFC6502] notifications about conference object modifications, will receive the most up-to-date version of that object upon receipt of a notification.

The "version" parameter is OPTIONAL for requests, since it is not needed by the server: as long as the required modifications can be applied to the target conference object without conflicts, the server does not care whether the client has stored an up-to-date view of the information. In addition, to ensure the integrity of the data, the conference server first checks all the parameters, before making any changes to the internal representation of the conference object. For example, it would be undesirable to change the <subject> of the conference, but then detect an invalid URI in one of the <service-uris> and abort the remaining updates.

4.3. Data Model Compliance

The XCON data model [RFC6501] identifies some elements and attributes as mandatory. Since the XML documents carried in the body of the CCMP requests and responses need to be compliant with the XCON data model, there can be a problem in cases of client-initiated operations, such as the initial creation of conference objects and cases whereby a client updates a conference object adding new elements, such as a new user. In such cases, not all of the mandatory data can be known in advance by the client issuing a CCMP request. As an example, a client cannot know, at the time it issues a conference creation request, the XCON-URI that the server will assign to the yet-to-be-created conference; hence, it is not able to populate the mandatory 'entity' attribute of the conference document contained in the request with the correct value. To solve this issue, the CCMP client fills all mandatory data model fields, for which no value is available at the time the request is constructed, with placeholder values in the form of a wildcard string, AUTO_GENERATE_X (all uppercase), with X being a unique numeric index for each data model field for which the value is unknown. This form of wildcard string is chosen, rather than the use of random unique strings (e.g., FOO_BAR_LA) or non-numeric values for X, to simplify processing at the server. The values of AUTO_GENERATE_X are only unique within the context of the specific request. The placeholder AUTO_GENERATE_X values MUST be within the value part of an attribute or element (e.g., <userinfo entity="xcon-userid:AUTO_GENERATE_1@example.com">).

When the server receives requests containing values in the form of AUTO_GENERATE_X, the server does the following:

(a) Generates the proper identifier for each instance of AUTO_GENERATE_X in the document. If an instance of AUTO_GENERATE_X is not within the value part of the attribute/element, the server MUST send a <response-code> of "400 Bad Request". In cases where AUTO_GENERATE_X appears only in the user part of a URI (i.e., in the case of XCON-USERIDs or XCON-URIs), the server needs to ensure that the domain name is one that is within the server's domain of responsibility. If the domain name is not within the server's domain of responsibility, then the server MUST send a <response-code> of "427 Invalid Domain Name". The server MUST replace each instance of a specific wildcard field (e.g., AUTO_GENERATE_1) with the same identifier. The identifiers MUST be unique for each instance of AUTO_GENERATE_X within the same XML document received in the request; for example, the value that replaces AUTO_GENERATE_1 MUST NOT be the same as the value that replaces AUTO_GENERATE_2. Note that the values that replace the instances of AUTO_GENERATE_X are not the same across all conference objects; for example, different values can be used to replace AUTO_GENERATE_1 in two different documents.

(b) Sends a response in which all values of AUTO_GENERATE_X received in the request have been replaced by the newly created one(s).

With this approach, compatibility with the data model requirements is maintained, while allowing for client-initiated manipulation of conference objects at the server's side. Note that the use of this mechanism could be avoided in come cases by using multiple operations, such as creating a new user and then adding the new user to an existing conference. However, the AUTO_GENERATE_X mechanism allows a single operation to be used to effect the same change on the conference object.

4.4. Implementation Approach

CCMP is implemented using HTTP, placing the CCMP request messages into the body of an HTTP POST operation and placing the CCMP responses into the body of the HTTP response messages. A non-exhaustive summary of the other approaches that were considered and the perceived advantages of the HTTP solution described in this document are provided in Appendix A.

Most CCMP commands can pend indefinitely, thus increasing the potential that pending requests can continue to increase when a server is receiving more requests than it can process within a

specific time period. In this case, a server SHOULD return a <response-code> of "510" to the pending requests. In addition, to mitigate the situation, clients MUST NOT wait indefinitely for a response and MUST implement a timer such that when it expires, the client MUST close the connection. Thirty seconds is RECOMMENDED as the default value for this timer. Sixty seconds is considered a reasonable upper range. Note that there may be cases where a response message is lost and a request has been successful (e.g., user added to a conference); yet, the client will be unaware and close the connection. However, as described in Section 4.2, there is a versioning mechanism for the conference objects; thus, there is a mechanism for the conference object stored by the client to be brought up to date.

CCMP messages have a MIME-type of "application/ccmp+xml", which appears inside the Content-Type and Accept header fields of HTTP requests and responses. The XML documents in the CCMP messages MUST be encoded in UTF-8. This specification follows the recommendations and conventions described in [RFC3023], including the naming convention of the type ('+xml' suffix) and the usage of the 'charset' parameter. The 'charset' parameter MUST be included with the XML document. Section 9 provides the complete requirements for an HTTP implementation to support CCMP.

5. CCMP Messages

CCMP messages are either requests or responses. The general CCMP request message is defined in Section 5.1. The general CCMP response message is defined in Section 5.2. The details of the specific message type that is carried in the CCMP request and response messages are described in Section 5.3. CCMP response codes are listed in Section 5.4.

5.1. CCMP Request Message Type

A CCMP request message is comprised of the following parameters:

subject: An OPTIONAL parameter containing the username and password of the client registered at the conferencing system. Each user who subscribes to the conferencing system is assumed to be equipped with those credentials and SHOULD enclose them in each CCMP request she issues. These fields can be used to control that the user sending the CCMP request has the authority to perform the requested operation. The same fields can also be used for other authorization and authentication procedures.

confUserID: An OPTIONAL parameter containing the XCON-USERID of the client. The XCON-USERID is used to identify any conferencing client within the context of the conferencing system and it is assigned by the conference server for each conferencing client who interacts with it. The <confUserID> parameter is REQUIRED in the CCMP request and response messages with the exception of the case of a user who has no XCON-USERID and who wants to enter, via CCMP, a conference whose identifier is known. In such case, a side effect of the request is that the user is provided with an appropriate XCON-USERID. An example of the aforementioned case will be provided in Section 5.3.6.

confObjID: An OPTIONAL parameter containing the XCON-URI of the target conference object.

operation: An OPTIONAL parameter refining the type of specialized request message. The <operation> parameter is REQUIRED in all requests except for the blueprintsRequest and confsRequest specialized messages.

conference-password: The parameter is OPTIONAL except that it MUST be inserted in all requests whose target conference object is password-protected i.e., contains the <conference-password> element in [RFC6501]). A CCMP <response-code> of "423" MUST be returned if a conference-password is not included in the request when required.

specialized request message: This is a specialization of the generic request message (e.g., blueprintsRequest), containing parameters that are dependent on the specific request sent to the server. A specialized request message MUST be included in the CCMP request message. The details for the specialized messages and associated parameters are provided in Section 5.3.

```
   <!-- Definition of CCMP Request -->

   <xs:element name="ccmpRequest" type="ccmp-request-type" />

   <!-- Definition of ccmp-request-type-->

   <xs:complexType name="ccmp-request-type">
       <xs:sequence>
           <xs:element name="ccmpRequest"
                       type="ccmp-request-message-type" />
       </xs:sequence>
   </xs:complexType>

    <!--  Definition of ccmp-request-message-type -->

    <xs:complexType abstract="true"
                    name="ccmp-request-message-type">
        <xs:sequence>
            <xs:element name="subject" type="subject-type"
                        minOccurs="0" maxOccurs="1" />
            <xs:element name="confUserID" type="xs:string"
                        minOccurs="0" maxOccurs="1" />
            <xs:element name="confObjID" type="xs:string"
                        minOccurs="0" maxOccurs="1" />
            <xs:element name="operation" type="operationType"
                        minOccurs="0" maxOccurs="1" />
            <xs:element name="conference-password" type="xs:string"
                        minOccurs="0" maxOccurs="1" />
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>
```

Figure 2: Structure of CCMP Request Messages

5.2. CCMP Response Message Type

A CCMP response message is comprised of the following parameters:

confUserID: A REQUIRED parameter in CCMP response messages containing the XCON-USERID of the conferencing client that issued the CCMP request message.

confObjID: An OPTIONAL parameter containing the XCON-URI of the target conference object.

operation: An OPTIONAL parameter for CCMP response messages. This parameter is REQUIRED in all responses except for the "blueprintsResponse" and "confsResponse" specialized messages.

response-code: A REQUIRED parameter containing the response code associated with the request. The response code MUST be chosen from the codes listed in Section 5.4.

response-string: An OPTIONAL reason string associated with the response. In case of an error, in particular, this string can be used to provide the client with detailed information about the error itself.

version: An OPTIONAL parameter reflecting the current version number of the conference object referred by the confObjID. This number is contained in the 'version' attribute of the <conference-info> element related to that conference. This parameter is REQUIRED in CCMP response messages and SHOULD NOT be included in CCMP request messages.

specialized response message: This is specialization of the generic response message, containing parameters that are dependent on the specific request sent to the server (e.g., "blueprintsResponse"). A specialized response message SHOULD be included in the CCMP response message, except in an error situation where the CCMP request message did not contain a valid specialized message. In this case, the conference server MUST return a <response-code> of "400". The details for the specialized messages and associated parameters are provided in Section 5.3.

```
<!-- Definition of CCMP Response -->

<xs:element name="ccmpResponse" type="ccmp-response-type" />

<!-- Definition of ccmp-response-type -->

<xs:complexType name="ccmp-response-type">
    <xs:sequence>
        <xs:element name="ccmpResponse"
                    type="ccmp-response-message-type" />
    </xs:sequence>
</xs:complexType>

<!--  Definition of ccmp-response-message-type -->

<xs:complexType abstract="true"
                name="ccmp-response-message-type">
    <xs:sequence>
        <xs:element name="confUserID" type="xs:string"
                    minOccurs="1" maxOccurs="1" />
        <xs:element name="confObjID" type="xs:string"
                    minOccurs="0" maxOccurs="1" />
        <xs:element name="operation" minOccurs="0"
                    maxOccurs="1" />
        <xs:element name="response-code"
                    type="response-codeType"
                    minOccurs="1" maxOccurs="1" />
        <xs:element name="response-string" type="xs:string"
                    minOccurs="0" maxOccurs="1" />
        <xs:element name="version" type="xs:positiveInteger"
                    minOccurs="0" maxOccurs="1" />
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

Figure 3: Structure of CCMP Response Message

5.3. Detailed Messages

Based on the request and response message structures described in Sections 5.1 and 5.2, the following summarizes the specialized CCMP request and response types described in this document:

1. blueprintsRequest/blueprintsResponse

2. confsRequest/confsResponse

3. blueprintRequest/blueprintResponse

4. confRequest/confResponse

5. usersRequest/usersResponse

6. userRequest/userResponse

7. sidebarsByValRequest/sidebarsByValResponse

8. sidebarsByRefRequest/sidebarsByRefResponse

9. sidebarByValRequest/sidebarByValResponse

10. sidebarByRefRequest/sidebarByRefResponse

11. extendedRequest/extendedResponse

12. optionsRequest/optionsResponse

These CCMP request/response pairs use the fundamental CCMP operations as defined in Section 4.1 to manipulate the conference data. These request/response pairs are included in an IANA registry as defined in Section 12.5. Table 1 summarizes the remaining CCMP operations and corresponding actions that are valid for a specific CCMP request type, noting that neither the blueprintsRequest/blueprintsResponse nor confsRequest/confsResponse require an <operation> parameter. An entity MUST support the response message for each of the request messages that is supported. The corresponding response message MUST contain the same <operation> parameter. Note that some entries are labeled "N/A", indicating that the operation is invalid for that request type. In the case of an "N/A*" label, the operation MAY be allowed for specific privileged users or system administrators but is not part of the functionality included in this document.

Operation ------------ Request Type	Retrieve	Create	Update	Delete
blueprints Request	Get list of blueprints	N/A	N/A	N/A
blueprint Request	Get blueprint	N/A*	N/A*	N/A*
confsRequest	Get list of confs	N/A	N/A	N/A
confRequest	Get conference object	Create conference object	Change conference object	Delete conference object
usersRequest	Get <users>	N/A(**)	Change <users>	N/A(**)
userRequest	Get specified <user>	Add a <user> to a conf (***)	Change specified <user>	Delete specified <user>
sidebarsByVal Request	Get <sidebars-by-val>	N/A	N/A	N/A
sidebarsByRef Request	Get <sidebars-by-ref>	N/A	N/A	N/A
sidebarByValRequest	Get sidebar-by-val	Create sidebar-by-val	Change sidebar-by-val	Delete sidebar-by-val
sidebarByRefRequest	Get sidebar-by-ref	Create sidebar-by-ref	Change sidebar-by-ref	Delete sidebar-by-ref

Table 1: Request Type Operation-Specific Processing

(**): These operations are not allowed for a usersRequest message, since the <users> section, which is the target element of such a request, is created and removed in conjunction with the creation and deletion, respectively, of the associated conference document. Thus, "update" and "retrieve" are the only semantically correct operations for such message.

(***): This operation can involve the creation of an XCON-USERID, if the sender does not add it in the <confUserID> parameter and/or if the entity field of the <userInfo> parameter is void.

Additional parameters included in the specialized CCMP request and response messages are detailed in the subsequent sections. If a required parameter is not included in a request, the conference server MUST return a <response-code> of "400" per Section 5.4.

5.3.1. blueprintsRequest and blueprintsResponse

A blueprintsRequest (Figure 4) message is sent to request the list of XCON-URIs associated with the available blueprints from the conference server. These XCON-URIs can be subsequently used by the client to access detailed information about a specified blueprint with a specific blueprintRequest message per Section 5.3.3.

The <confUserID> parameter MUST be included in every blueprintsRequest/Response message and reflect the XCON-USERID of the conferencing client issuing the request. Since a blueprintsRequest message is not targeted to a specific conference instance and is a "retrieve-only" request, the <confObjID> and <operation> parameters MUST NOT be included in the blueprintsRequest/Response messages.

In order to obtain a specific subset of the available blueprints, a client may specify a selection filter providing an appropriate xpath query in the OPTIONAL "xpathFilter" parameter of the request. The information in the blueprints typically represents general capabilities and characteristics. For example, to select blueprints having both audio and video stream support, a possible xpathFilter value could be: "/conference-info[conference-description/ available-media/entry/type='audio' and conference-description/ available-media/entry/type='video']". A conference server SHOULD NOT provide any sensitive information (e.g., passwords) in the blueprints.

The associated blueprintsResponse message SHOULD contain, as shown in Figure 4, a "blueprintsInfo" parameter containing the above mentioned XCON-URI list.

```
<!-- blueprintsRequest -->
<xs:complexType name="ccmp-blueprints-request-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type">
            <xs:sequence>
                <xs:element ref="blueprintsRequest" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- blueprintsRequestType -->

<xs:element name="blueprintsRequest" type="blueprintsRequestType"/>

<xs:complexType name="blueprintsRequestType">
    <xs:sequence>
        <xs:element name="xpathFilter" type="xs:string" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<!-- blueprintsResponse -->

<xs:complexType name="ccmp-blueprints-response-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-response-message-type">
            <xs:sequence>
                <xs:element ref="blueprintsResponse" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

```
 <!-- blueprintsResponseType -->

 <xs:element name="blueprintsResponse" type="blueprintsResponseType"/>

 <xs:complexType name="blueprintsResponseType">
     <xs:sequence>
         <xs:element name="blueprintsInfo"
                     type="info:uris-type" minOccurs="0"/>
         <xs:any namespace="##other" processContents="lax"
                 minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:anyAttribute namespace="##any" processContents="lax"/>
 </xs:complexType>
```

Figure 4: Structure of the blueprintsRequest and blueprintsResponse Messages

5.3.2. confsRequest and confsResponse

A confsRequest message is used to retrieve, from the server, the list of XCON-URIs associated with active and registered conferences currently handled by the conferencing system. The <confUserID> parameter MUST be included in every confsRequest/Response message and reflect the XCON-USERID of the conferencing client issuing the request. The <confObjID> parameter MUST NOT be included in the confsRequest message. The confsRequest message is of a retrieve-only type, since the sole purpose is to collect information available at the conference server. Thus, an <operation> parameter MUST NOT be included in a confsRequest message. In order to retrieve a specific subset of the available conferences, a client may specify a selection filter providing an appropriate xpath query in the OPTIONAL "xpathFilter" parameter of the request. For example, to select only the registered conferences, a possible xpathFilter value could be "/conference-info[conference-description/conference-state/active='false']". The associated confsResponse message SHOULD contain the list of XCON-URIs in the "confsInfo" parameter. A user, upon receipt of the response message, can interact with the available conference objects through further CCMP messages.

```
<!-- confsRequest -->

<xs:complexType name="ccmp-confs-request-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type">
            <xs:sequence>
                <xs:element ref="confsRequest" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- confsRequestType -->

<xs:element name="confsRequest" type="confsRequestType" />

<xs:complexType name="confsRequestType">
    <xs:sequence>
        <xs:element name="xpathFilter" type="xs:string" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<!-- confsResponse -->

<xs:complexType name="ccmp-confs-response-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-response-message-type">
            <xs:sequence>
                <xs:element ref="confsResponse" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

```
 <!-- confsResponseType -->

 <xs:element name="confsResponse" type="confsResponseType"/>

 <xs:complexType name="confsResponseType">
     <xs:sequence>
         <xs:element name="confsInfo" type="info:uris-type"
                     minOccurs="0"/>
         <xs:any namespace="##other" processContents="lax"
                 minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:anyAttribute namespace="##any" processContents="lax"/>
 </xs:complexType>
```

Figure 5: Structure of the confsRequest and confsResponse Messages

5.3.3. blueprintRequest and blueprintResponse

Through a blueprintRequest, a client can manipulate the conference object associated with a specified blueprint. Along with the <confUserID> parameter, the request MUST include the <confObjID> and the <operation> parameters. Again, the <confUserID> parameter MUST be included in every blueprintRequest/Response message and reflect the XCON-USERID of the conferencing client issuing the request. The <confObjID> parameter MUST contain the XCON-URI of the blueprint, which might have been previously retrieved through a blueprintsRequest message.

The blueprintRequest message SHOULD NOT contain an <operation> parameter with a value other than "retrieve". An <operation> parameter with a value of "create", "update", or "delete" SHOULD NOT be included in a blueprintRequest message except in the case of privileged users (e.g., the conference server administration staff), who might authenticate themselves by the mean of the "subject" request parameter.

A blueprintRequest/retrieve carrying a <confObjID> parameter whose value is not associated with one of the available system's blueprints, will generate, on the server's side, a blueprintResponse message containing a <response-code> of "404". This also holds for the case in which the mentioned <confObjID> parameter value is related to an existing conference document stored at the server, but associated with an actual conference (be it active or registered) or with a sidebar rather than a blueprint.

For a <response-code> of "200" in a "retrieve" operation, the <blueprintInfo> parameter MUST be included in the blueprintResponse message. The <blueprintInfo> parameter contains the conference document associated with the blueprint as identified by the <confObjID> parameter specified in the blueprintRequest.

```
<!--  blueprintRequest -->

<xs:complexType name="ccmp-blueprint-request-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type">
            <xs:sequence>
                <xs:element ref="blueprintRequest" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- blueprintRequestType -->

<xs:element name="blueprintRequest" type="blueprintRequestType" />

<xs:complexType name="blueprintRequestType">
    <xs:sequence>
        <xs:element name="blueprintInfo"
                    type="info:conference-type" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<!-- blueprintResponse -->

<xs:complexType name="ccmp-blueprint-response-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-response-message-type">
            <xs:sequence>
                <xs:element ref="blueprintResponse" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

```
<!-- blueprintResponseType -->

<xs:element name="blueprintResponse" type="blueprintResponseType"/>

<xs:complexType name="blueprintResponseType">
    <xs:sequence>
        <xs:element name="blueprintInfo" type="info:conference-type"
                    minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded">
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

Figure 6: Structure of the blueprintRequest and blueprintResponse Messages

5.3.4. confRequest and confResponse

With a confRequest message, CCMP clients can manipulate conference objects associated with either active or registered conferences. The <confUserID> parameter MUST be included in every confRequest/Response message and reflect the XCON-USERID of the conferencing client issuing the request. confRequest and confResponse messages MUST also include an <operation> parameter. ConfResponse messages MUST return to the requestor a <response-code> and MAY contain a <response-string> explaining it. Depending upon the type of operation, a <confObjID> and <confInfo> parameter MAY be included in the confRequest and response. For each type of operation, the text below describes whether the <confObjID> and <confInfo> parameters need to be included in the confRequest and confResponse messages.

The creation case deserves care. To create a new conference through a confRequest message, two approaches can be considered:

1. Creation through explicit cloning: the <confObjID> parameter MUST contain the XCON-URI of the blueprint or of the conference to be cloned, while the <confInfo> parameter MUST NOT be included in the confRequest. Note that cloning of an active conference is only done in the case of a sidebar operation per the XCON framework and as described in Section 5.3.8.

2. Creation through implicit cloning (also known as "direct creation"): the <confObjID> parameter MUST NOT be included in the request and the CCMP client can describe the desired conference to be created using the <confInfo> parameter. If no <confInfo> parameter is provided in the request, the new conference will be created as a clone of the system default blueprint.

In both creation cases, the confResponse, for a successful completion of a "create" operation, contains a <response-code> of "200" and MUST contain the XCON-URI of the newly created conference in the <confObjID> parameter, in order to allow the conferencing client to manipulate that conference through following CCMP requests. In addition, the <confInfo> parameter containing the conference document created MAY be included, at the discretion of the conferencing system implementation, along with the REQUIRED <version> parameter initialized at "1", since, at creation time, the conference object is at its first version.

In the case of a confRequest with an <operation> parameter of "retrieve", the <confObjID> parameter representing the XCON-URI of the target conference MUST be included and the <confInfo> parameter MUST NOT be included in the request. The conference server MUST ignore any <confInfo> parameter that is received in a confRequest "retrieve" operation. If the confResponse for the retrieve operation contains a <response-code> of "200", the <confInfo> parameter MUST be included in the response. The <confInfo> parameter MUST contain the entire conference document describing the target conference object in its current state. The current state of the retrieved conference object MUST also be reported in the proper "version" response parameter.

In case of a confRequest with an <operation> parameter of "update", the <confInfo> and <confObjID> parameters MUST be included in the request. The <confInfo> represents an object of type "conference-type" containing all the changes to be applied to the conference whose identifier has the same value as the <confObjID> parameter. Note that, in such a case, though the <confInfo> parameter indeed has to follow the rules indicated in the XCON data model, it does not represent the entire updated version of the target conference, since it conveys just the modifications to apply to that conference. For example, in order to change the conference title, the <confInfo> parameter will be of the form:

```
<confInfo entity="xcon:8977777@example.com">
  <conference-description>
    <display-text> *** NEW CONFERENCE TITLE *** </display-text>
  </conference-description>
</confInfo>
```

Figure 7: Updating a Conference Object: Modifying the Title of a Conference

Similarly, to remove the title of an existing conference, a confRequest/update carrying the following <confInfo> parameter would do the job.

```
<confInfo entity="xcon:8977777@example.com">
  <conference-description>
    <display-text/>
  </conference-description>
</confInfo>
```

Figure 8: Updating a Conference Object:
Removing the Title of a Conference

In the case of a confResponse/update with a <response-code> of "200", no additional information is REQUIRED in the response message, which means the return of a <confInfo> parameter is OPTIONAL. A subsequent confRequest/retrieve transaction might provide the CCMP client with the current status of the conference after the modification, or the notification protocol might address that task as well. A <response-code> of "200" indicates that the conference object has been changed according to the request by the conference server. The <version> parameter MUST be enclosed in the confResponse/update message, in order to let the client understand what is the current conference-object version, upon the applied modifications. A <response-code> of "409" indicates that the changes reflected in the <confInfo> parameter of the request are not feasible. This could be due to policies, requestor roles, specific privileges, unacceptable values, etc., with the reason specific to a conferencing system and its configuration. Together with a <response-code> of "409", the <version> parameter MUST be attached in the confResponse/update, allowing the client to later retrieve the current version of the target conference if the one she attempted to modify was not the most up to date.

In the case of a confRequest with an <operation> parameter of "delete", the <confObjID> parameter representing the XCON-URI of the target conference MUST be included while the <confInfo> parameter MUST NOT be included in the request. The conference server MUST ignore any <confInfo> parameter that is received within such a request. The confResponse MUST contain the same value for the <confObjID> parameter that was included in the confRequest. If the confResponse/delete operation contains a <response-code> of "200", the conference indicated in the <confObjID> parameter has been successfully deleted. A confResponse/delete with a <response-code> of "200" MUST NOT contain the <confInfo> parameter. The <version> parameter SHOULD NOT be returned in any confResponse/delete. If the conference server cannot delete the conference referenced by the <confObjID> parameter received in the confRequest because it is the parent of another conference object that is in use, the conference server MUST return a <response-code> of "425".

A confRequest with an <operation> parameter of "retrieve", "update", or "delete" carrying a <confObjID> parameter which is not associated with one of the conferences (active or registered) that the system is holding will generate, on the server's side, a confResponse message containing a <response-code> of "404". This also holds for the case in which the mentioned <confObjID> parameter is related to an existing conference object stored at the server, but associated with a blueprint or with a sidebar rather than an actual conference.

The schema for the confRequest/confResponse pair is shown in Figure 9.

```
<!-- confRequest -->

<xs:complexType name="ccmp-conf-request-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type">
            <xs:sequence>
                <xs:element ref="confRequest" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- confRequestType -->

<xs:element name="confRequest" type="confRequestType" />

<xs:complexType name="confRequestType">
    <xs:sequence>
        <xs:element name="confInfo" type="info:conference-type"
                    minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

```
<!-- confResponse -->

<xs:complexType name="ccmp-conf-response-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-response-message-type">
            <xs:sequence>
                <xs:element ref="confResponse" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- confResponseType -->

<xs:element name="confResponse" type="confResponseType" />

<xs:complexType name="confResponseType">
    <xs:sequence>
        <xs:element name="confInfo" type="info:conference-type"
                    minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

Figure 9: Structure of the confRequest and confResponse Messages

5.3.5. usersRequest and usersResponse

The usersRequest message allows a client to manipulate the <users> element of the conference object represented by the <confObjID> parameter. The <users> element contains the list of <user> elements associated with conference participants, the list of the users to which access to the conference is allowed/denied, conference participation policies, etc. The <confObjID> parameter MUST be included in a usersRequest message.

A <usersInfo> parameter MAY be included in a usersRequest message depending upon the operation. If the <usersInfo> parameter is included in the usersRequest message, the parameter MUST be compliant with the <users> field of the XCON data model.

Two operations are allowed for a usersRequest message:

1. "retrieve": In this case the request MUST NOT include a <usersInfo> parameter, while the successful response MUST contain the desired <users> element in the <usersInfo> parameter. The

conference server MUST ignore a <usersInfo> parameter if it is received in a request with an <operation> parameter of "retrieve".

2. "update": In this case, the <usersInfo> parameter MUST contain the modifications to be applied to the <users> element indicated. If the <response-code> is "200", then the <usersInfo> parameter SHOULD NOT be returned. Any <usersInfo> parameter that is returned SHOULD be ignored. A <response-code> of "426" indicates that the conferencing client is not allowed to make the changes reflected in the <usersInfo> contained in the usersRequest message. This could be due to policies, roles, specific privileges, etc., with the reason being specific to a conferencing system and its configuration.

Operations of "create" and "delete" are not applicable to a usersRequest message and MUST NOT be considered by the server, which means that a <response-code> of "403" MUST be included in the usersResponse message.

```
<!-- usersRequest -->

<xs:complexType name="ccmp-users-request-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type">
            <xs:sequence>
                <xs:element ref="usersRequest" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- usersRequestType -->

<xs:element name="usersRequest" type="usersRequestType" />

<xs:complexType name="usersRequestType">
    <xs:sequence>
        <xs:element name="usersInfo"
                    type="info:users-type" minOccurs="0" />
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

```
<!-- usersResponse -->

<xs:complexType name="ccmp-users-response-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-response-message-type">
            <xs:sequence>
                <xs:element ref="usersResponse" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- usersResponseType -->

<xs:element name="usersResponse" type="usersResponseType" />

<xs:complexType name="usersResponseType">
    <xs:sequence>
        <xs:element name="usersInfo" type="info:users-type"
                    minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

Figure 10: Structure of the usersRequest and usersResponse Messages

5.3.6. userRequest and userResponse

A userRequest message is used to manipulate <user> elements inside a conference document associated with a conference identified by the <confObjID> parameter. Besides retrieving information about a specific conference user, the message is used to request that the conference server either create, modify, or delete information about a user. A userRequest message MUST include the <confObjID> and the <operation> parameters, and it MAY include a <userInfo> parameter containing the detailed user's information depending upon the operation and whether the <userInfo> has already been populated for a specific user. Note that a user may not necessarily be a conferencing control client (i.e., some participants in a conference are not "XCON aware").

An XCON-USERID SHOULD be assigned to each and every user subscribed to the system. In such a way, a user who is not a conference participant can make requests (provided she has successfully passed authorization and authentication checks), like creating a conference or retrieving conference information.

Conference users can be created in a number of different ways. In each of these cases, the <operation> parameter MUST be set to "create" in the userRequest message. Each of the userResponse messages for these cases MUST include the <confObjID>, <confUserID>, <operation>, and <response-code> parameters. In the case of a <response-code> of "200", the userResponse message MAY include the <userInfo> parameter depending upon the manner in which the user was created:

- A conferencing client with an XCON-USERID adds itself to the conference: In this case, the <userInfo> parameter MAY be included in the userRequest. The <userInfo> parameter MUST contain a <user> element (compliant with the XCON data model) and the 'entity' attribute MUST be set to a value that represents the XCON-USERID of the user initiating the request. No additional parameters beyond those previously described are required in the userResponse message, in the case of a <response-code> of "200".

- A conferencing client acts on behalf of another user whose XCON-USERID is known: In this case, the <userInfo> parameter MUST be included in the userRequest. The <userInfo> parameter MUST contain a <user> element and the 'entity' attribute value MUST be set to the XCON-USERID of the other user in question. No additional parameters beyond those previously described are required in the userResponse message, in the case of a <response-code> of "200".

- A conferencing client who has no XCON-USERID and who wants to enter, via CCMP, a conference whose identifier is known: In this case, a side effect of the request is that the user is provided with a new XCON-USERID (created by the server) carried inside the <confUserID> parameter of the response. This is the only case in which a CCMP request can be valid though carrying a void <confUserID> parameter. A <userInfo> parameter MUST be enclosed in the request, providing at least a contact URI of the joining client, in order to let the focus initiate the signaling phase needed to add her to the conference. The mandatory 'entity' attribute of the <userInfo> parameter in the request MUST be filled with a placeholder value with the user part of the XCON-USERID containing a value of AUTO_GENERATE_X as described in Section 4.3, to conform to the rules contained in the XCON data model XML schema. The messages (userRequest and userResponse) in this case should look like the following:

```
Request fields:

confUserID=null;
confObjID=confXYZ;
operation=create;
userInfo=

<userInfo entity="xcon-userid:AUTO_GENERATE_1@example.com">
        <endpoint entity="sip:GHIL345@example.com">
        ...


Response fields (in case of success):

confUserID=user345;
confObjID=confXYZ;
operation=create;
response-code=200;
userInfo=null; //or the entire userInfo object
```

Figure 11: userRequest and userResponse in the Absence of an xcon-userid

- A conferencing client is unaware of the XCON-USERID of a third user: In this case, the XCON-USERID in the request, <confUserID>, is the sender's and the 'entity' attribute of the attached <userInfo> parameter is filled with the placeholder value "xcon-userid:AUTO_GENERATE_1@example.com". The XCON-USERID for the third user MUST be returned to the client issuing the request in the 'entity' attribute of the response <userInfo> parameter, if the <response-code> is "200". This scenario is intended to support both the case where a brand new conferencing system user is added to a conference by a third party (i.e., a user who has not yet been provided with an XCON-USERID) and the case where the CCMP client issuing the request does not know the to-be-added user's XCON-USERID (which means such an identifier could already exist on the server's side for that user). In this last case, the conference server is in charge of avoiding XCON-URI duplicates for the same conferencing client, looking at key fields in the request-provided <userInfo> parameter, such as the signaling URI. If the joining user is brand new, then the generation of a new XCON-USERID is needed; otherwise, if that user exists already, the server must recover the corresponding XCON-USERID.

In the case of a userRequest with an <operation> parameter of "retrieve", the <confObjID> parameter representing the XCON-URI of the target conference MUST be included. The <confUserID>, containing the CCMP client's XCON-USERID, MUST also be included in the

userRequest message. If the client wants to retrieve information about her profile in the specified conference, no <userInfo> parameter is needed in the retrieve request. On the other hand, if the client wants to obtain someone else's info within the given conference, she MUST include in the userRequest/retrieve a <userInfo> parameter whose 'entity' attribute conveys the desired user's XCON-USERID. If the userResponse for the retrieve operation contains a <response-code> of "200", the <userInfo> parameter MUST be included in the response.

In case of a userRequest with an <operation> parameter of "update", the <confObjID>, <confUserID>, and <userInfo> parameters MUST be included in the request. The <userInfo> parameter is of type "user-type" and contains all the changes to be applied to a specific <user> element in the conference object identified by the <confObjID> parameter in the userRequest message. The user to be modified is identified through the 'entity' attribute of the <userInfo> parameter included in the request. In the case of a userResponse with a <response-code> of "200", no additional information is required in the userResponse message. A <response-code> of "200" indicates that the referenced <user> element has been updated by the conference server. A <response-code> of "426" indicates that the conferencing client is not allowed to make the changes reflected in the <userInfo> in the initial request. This could be due to policies, roles, specific privileges, etc., with the reason specific to a conferencing system and its configuration.

In the case of a userRequest with an <operation> parameter of "delete", the <confObjID> representing the XCON-URI of the target conference MUST be included. The <confUserID> parameter, containing the CCMP client's XCON-USERID, MUST be included in the userRequest message. If the client wants to exit the specified conference, no <userInfo> parameter is needed in the delete request. On the other hand, if the client wants to remove another participant from the given conference, she MUST include in the userRequest/delete a <userInfo> parameter whose 'entity' attribute conveys the XCON-USERID of that participant. The userResponse MUST contain the same value for the <confObjID> parameter that was included in the <confObjID> parameter in the userRequest. The userResponse MUST contain a <response-code> of "200" if the target <user> element has been successfully deleted. If the userResponse for the delete operation contains a <response-code> of "200", the userResponse MUST NOT contain the <userInfo> parameter.

```
   <!-- userRequest -->

   <xs:complexType name="ccmp-user-request-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-request-message-type">
               <xs:sequence>
                   <xs:element ref="userRequest" />
               </xs:sequence>
           </xs:extension>
       </xs:complexContent>
   </xs:complexType>

   <!-- userRequestType -->

   <xs:element name="userRequest" type="userRequestType" />

   <xs:complexType name="userRequestType">
       <xs:sequence>
           <xs:element name="userInfo"
                       type="info:user-type" minOccurs="0" />
           <xs:any namespace="##other" processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- userResponse -->

   <xs:complexType name="ccmp-user-response-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-response-message-type">
               <xs:sequence>
                   <xs:element ref="userResponse" />
               </xs:sequence>
           </xs:extension>
       </xs:complexContent>
   </xs:complexType>
```

```
<!-- userResponseType -->

<xs:element name="userResponse" type="userResponseType" />

<xs:complexType name="userResponseType">
    <xs:sequence>
        <xs:element name="userInfo" type="info:user-type"
                    minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

Figure 12: Structure of the userRequest and userResponse Messages

5.3.7. sidebarsByValRequest and sidebarsByValResponse

A sidebarsByValRequest message is used to execute a retrieve-only operation on the <sidebars-by-val> field of the conference object represented by the <confObjID>. The sidebarsByValRequest message is of a retrieve-only type, so an <operation> parameter MUST NOT be included in a sidebarsByValRequest message. As with blueprints and conferences, CCMP allows for the use of xpath filters whenever a selected subset of the sidebars available at the server's side has to be retrieved by the client. This applies both to sidebars by reference and sidebars by value. A sidebarsByValResponse message with a <response-code> of "200" MUST contain a <sidebarsByValInfo> parameter containing the desired <sidebars-by-val> element. A sidebarsByValResponse message MUST return to the client a <version> element related to the current version of the main conference object (i.e., the one whose identifier is contained in the <confObjID> field of the request) with which the sidebars in question are associated. The <sidebarsByValInfo> parameter contains the list of the conference objects associated with the sidebars by value derived from the main conference. The retrieved sidebars can then be updated or deleted using the sidebarByValRequest message, which is described in Section 5.3.8.

```
<!-- sidebarsByValRequest -->

<xs:complexType name="ccmp-sidebarsByVal-request-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type">
            <xs:sequence>
                <xs:element ref="sidebarsByValRequest"/>
            </xs:sequence>
        </xs:extension>
   </xs:complexContent>
</xs:complexType>

<!-- sidebarsByValRequestType -->

<xs:element name="sidebarsByValRequest"
            type="sidebarsByValRequestType" />

<xs:complexType name="sidebarsByValRequestType">
    <xs:sequence>
        <xs:element name="xpathFilter" type="xs:string" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>


<!-- sidebarsByValResponse -->

<xs:complexType name="ccmp-sidebarsByVal-response-message-type">
        <xs:complexContent>
         <xs:extension base="tns:ccmp-response-message-type">
                <xs:sequence>
           <xs:element ref="sidebarsByValResponse"/>
          </xs:sequence>
         </xs:extension>
        </xs:complexContent>
</xs:complexType>
```

```
 <!-- sidebarsByValResponseType -->

 <xs:element name="sidebarsByValResponse"
             type="sidebarsByValResponseType" />

 <xs:complexType name="sidebarsByValResponseType">
     <xs:sequence>
         <xs:element name="sidebarsByValInfo"
                     type="info:sidebars-by-val-type" minOccurs="0"/>
         <xs:any namespace="##other" processContents="lax"
                 minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
 </xs:complexType>
```

Figure 13: Structure of the sidebarsByValRequest and sidebarsByValResponse Messages

5.3.8. sidebarByValRequest and sidebarByValResponse

A sidebarByValRequest message MUST contain the <operation> parameter, which distinguishes among retrieval, creation, modification, and deletion of a specific sidebar. The other required parameters depend upon the type of operation.

In the case of a "create" operation, the <confObjID> parameter MUST be included in the sidebyValRequest message. In this case, the <confObjID> parameter contains the XCON-URI of the main conference in which the sidebar has to be created. If no "sidebarByValInfo" parameter is included, the sidebar is created by cloning the main conference, as envisioned in the XCON framework [RFC5239] following the implementation specific cloning rules. Otherwise, similar to the case of direct creation, the sidebar conference object is built on the basis of the "sidebarByValInfo" parameter provided by the requestor. As a consequence of a sidebar-by-val creation, the conference server MUST update the main conference object reflected by the <confObjID> parameter in the sidebarbyValRequest/create message introducing the new sidebar object as a new <entry> in the proper section <sidebars-by-val>. The newly created sidebar conference object MAY be included in the sidebarByValResponse in the <sidebarByValInfo> parameter, if the <response-code> is "200". The XCON-URI of the newly created sidebar MUST appear in the <confObjID> parameter of the response. The conference server can notify any conferencing clients that have subscribed to the conference event package and that are authorized to receive the notification of the addition of the sidebar to the conference.

In the case of a sidebarByValRequest message with an <operation> parameter of "retrieve", the URI for the conference object created for the sidebar (received in response to a create operation or in a sidebarsByValResponse message) MUST be included in the <confObjID> parameter in the request. This retrieve operation is handled by the conference server in the same manner as in the case of an <operation> parameter of "retrieve" included in a confRequest message, as described in Section 5.3.4.

In the case of a sidebarByValRequest message with an <operation> parameter of "update", the <sidebarByValInfo> MUST also be included in the request. The <confObjID> parameter contained in the request message identifies the specific sidebar instance to be updated. An update operation on the specific sidebar instance contained in the <sidebarByValInfo> parameter is handled by the conference server in the same manner as an update operation on the conference instance as reflected by the <confInfo> parameter included in a confRequest message as detailed in Section 5.3.4. A sidebarByValResponse message MUST return to the client a <version> element related to the current version of the sidebar whose identifier is contained in the <confObjID> field of the request.

If an <operation> parameter of "delete" is included in the sidebarByVal request, the <sidebarByValInfo> parameter MUST NOT be included in the request. Any <sidebarByValInfo> included in the request MUST be ignored by the conference server. The URI for the conference object associated with the sidebar MUST be included in the <confObjID> parameter in the request. If the specific conferencing user, as reflected by the <confUserID> parameter, in the request is authorized to delete the conference, the conference server deletes the conference object reflected by the <confObjID> parameter and updates the data in the conference object from which the sidebar was cloned. The conference server can notify any conferencing clients that have subscribed to the conference event package and that are authorized to receive the notification of the deletion of the sidebar from the conference.

If a sidebarByValRequest with an <operation> parameter of "retrieve", "update", or "delete" carries a <confObjID> parameter which is not associated with any existing sidebar-by-val, a confResponse message containing a <response-code> of "404" will be generated on the server's side. This also holds for the case in which the mentioned <confObjID> parameter is related to an existing conference object stored at the server, but associated with a blueprint or with an actual conference or with a sidebar-by-ref rather than a sidebar-by-val.

```
<!-- sidebarByValRequest -->

<xs:complexType name="ccmp-sidebarByVal-request-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type">
            <xs:sequence>
                <xs:element ref="sidebarByValRequest"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- sidebarByValRequestType -->

<xs:element name="sidebarByValRequest"
            type="sidebarByValRequestType" />

<xs:complexType name="sidebarByValRequestType">
    <xs:sequence>
        <xs:element name="sidebarByValInfo"
                    type="info:conference-type" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>


<!-- sidebarByValResponse -->

<xs:complexType name="ccmp-sidebarByVal-response-message-type">
 <xs:complexContent>
        <xs:extension base="tns:ccmp-response-message-type">
         <xs:sequence>
                <xs:element ref="sidebarByValResponse"/>
         </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

```
<!-- sidebarByValResponseType -->

<xs:element name="sidebarByValResponse"
            type="sidebarByValResponseType" />

<xs:complexType name="sidebarByValResponseType">
  <xs:sequence>
     <xs:element name="sidebarByValInfo"
                 type="info:conference-type" minOccurs="0"/>
     <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

Figure 14: Structure of the sidebarByValRequest and sidebarByValResponse Messages

5.3.9. sidebarsByRefRequest and sidebarsByRefResponse

Similar to the sidebarsByValRequest, a sidebarsByRefRequest can be invoked to retrieve the <sidebars-by-ref> element of the conference object identified by the <confObjID> parameter. The sidebarsByRefRequest message is of a retrieve-only type, so an <operation> parameter MUST NOT be included in a sidebarsByRefRequest message. In the case of a <response-code> of "200", the <sidebarsByRefInfo> parameter, containing the <sidebars-by-ref> element of the conference object, MUST be included in the response. The <sidebars-by-ref> element represents the set of URIs of the sidebars associated with the main conference, whose description (in the form of a standard XCON conference document) is external to the main conference itself. Through the retrieved URIs, it is then possible to access single sidebars using the sidebarByRefRequest message, described in Section 5.3.10. A sidebarsByRefResponse message MUST carry back to the client a <version> element related to the current version of the main conference object (i.e., the one whose identifier is contained in the <confObjId> field of the request) with which the sidebars in question are associated.

```
   <!-- sidebarsByRefRequest -->

   <xs:complexType name="ccmp-sidebarsByRef-request-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-request-message-type">
               <xs:sequence>
                   <xs:element ref="sidebarsByRefRequest"/>
               </xs:sequence>
           </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- sidebarsByRefRequestType -->

   <xs:element name="sidebarsByRefRequest"
               type="sidebarsByRefRequestType" />

   <xs:complexType name="sidebarsByRefRequestType">
       <xs:sequence>
           <xs:element name="xpathFilter"
                       type="xs:string" minOccurs="0"/>
           <xs:any namespace="##other" processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- sidebarsByRefResponse -->

   <xs:complexType name="ccmp-sidebarsByref-response-message-type">
           <xs:complexContent>
            <xs:extension base="tns:ccmp-response-message-type">
                    <xs:sequence>
                           <xs:element ref="sidebarsByRefResponse"/>
                    </xs:sequence>
            </xs:extension>
     </xs:complexContent>
   </xs:complexType>
```

```
<!-- sidebarsByRefResponseType -->

<xs:element name="sidebarsByRefResponse"
            type="sidebarsByRefResponseType" />

<xs:complexType name="sidebarsByRefResponseType">
    <xs:sequence>
       <xs:element name="sidebarsByRefInfo"
                   type="info:uris-type" minOccurs="0"/>
       <xs:any namespace="##other" processContents="lax"
               minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

Figure 15: Structure of the sidebarsByRefRequest and sidebarsByRefResponse Messages

5.3.10. sidebarByRefRequest and sidebarByRefResponse

A sidebarByValResponse message MUST return to the client a <version> element related to the current version of the sidebar whose identifier is contained in the <confObjID> field of the request.

In the case of a create operation, the <confObjID> parameter MUST be included in the sidebyRefRequest message. In this case, the <confObjID> parameter contains the XCON-URI of the main conference in which the sidebar has to be created. If no <sidebarByRefInfo> parameter is included, following the XCON framework [RFC5239], the sidebar is created by cloning the main conference, observing the implementation-specific cloning rules. Otherwise, similar to the case of direct creation, the sidebar conference object is built on the basis of the <sidebarByRefInfo> parameter provided by the requestor. If the creation of the sidebar is successful, the conference server MUST update the <sidebars-by-ref> element in the conference object from which the sidebar was created (i.e., as identified by the <confObjID> in the original sidebarByRefRequest), with the URI of the newly created sidebar. The newly created conference object MAY be included in the response in the <sidebarByRefInfo> parameter with a <response-code> of "200". The URI for the conference object associated with the newly created sidebar object MUST appear in the <confObjID> parameter of the response. The conference server can notify any conferencing clients that have subscribed to the conference event package and that are authorized to receive the notification of the addition of the sidebar to the conference.

In the case of a sidebarByRefRequest message with an <operation> parameter of "retrieve", the URI for the conference object created for the sidebar MUST be included in the <confObjID> parameter in the request. A retrieve operation on the <sidebarByRefInfo> is handled by the conference server in the same manner as a retrieve operation on the confInfo included in a confRequest message as detailed in Section 5.3.4.

In the case of a sidebarByRefRequest message with an <operation> parameter of "update", the URI for the conference object created for the sidebar MUST be included in the <confObjID> parameter in the request. The <sidebarByRefInfo> MUST also be included in the request in the case of an "update" operation. An update operation on the <sidebarByRefInfo> is handled by the conference server in the same manner as an update operation on the <confInfo> included in a confRequest message as detailed in Section 5.3.4. A sidebarByRefResponse message MUST carry back to the client a <version> element related to the current version of the sidebar whose identifier is contained in the <confObjID> field of the request.

If an <operation> parameter of "delete" is included in the sidebarByRefRequest, the <sidebarByRefInfo> parameter MUST NOT be included in the request. Any <sidebarByRefInfo> included in the request MUST be ignored by the conference server. The URI for the conference object for the sidebar MUST be included in the <confObjID> parameter in the request. If the specific conferencing user as reflected by the <confUserID> parameter in the request is authorized to delete the conference, the conference server SHOULD delete the conference object reflected by the <confObjID> parameter and SHOULD update the <sidebars-by-ref> element in the conference object from which the sidebar was originally cloned. The conference server can notify any conferencing clients that have subscribed to the conference event package and that are authorized to receive the notification of the deletion of the sidebar.

If a sidebarByRefRequest with an <operation> parameter of "retrieve", "update", or "delete" carries a <confObjID> parameter that is not associated with any existing sidebar-by-ref, a confResponse message containing a <response-code> of "404" will be generated on the server's side. This also holds for the case in which the value of the mentioned <confObjID> parameter is related to an existing conference object stored at the server, but associated with a blueprint or with an actual conference or with a sidebar-by-val rather than a sidebar-by-ref.

```
   <!-- sidebarByRefRequest -->

   <xs:complexType name="ccmp-sidebarByRef-request-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-request-message-type">
               <xs:sequence>
                   <xs:element ref="sidebarByRefRequest"/>
                </xs:sequence>
           </xs:extension>
       </xs:complexContent>
   </xs:complexType>

   <!-- sidebarByRefRequestType -->

   <xs:element name="sidebarByRefRequest"
               type="sidebarByRefRequestType" />

   <xs:complexType name="sidebarByRefRequestType">
       <xs:sequence>
           <xs:element name="sidebarByRefInfo"
                       type="info:conference-type" minOccurs="0"/>
           <xs:any namespace="##other" processContents="lax"
                  minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- sidebarByRefResponse -->

   <xs:complexType name="ccmp-sidebarByRef-response-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-response-message-type">
               <xs:sequence>
                   <xs:element ref="sidebarByRefResponse"/>
               </xs:sequence>
           </xs:extension>
       </xs:complexContent>
   </xs:complexType>
```

```
<!-- sidebarByRefResponseType -->

<xs:element name="sidebarByRefResponse"
            type="sidebarByRefResponseType" />

<xs:complexType name="sidebarByRefResponseType">
    <xs:sequence>
        <xs:element name="sidebarByRefInfo"
                    type="info:conference-type" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
               minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

Figure 16: Structure of the sidebarByRefRequest and sidebarByRefResponse Messages

5.3.11. extendedRequest and extendedResponse

In order to allow specifying new request and response pairs for conference control, CCMP defines the extendedRequest and extendedResponse messages. Such messages constitute a CCMP skeleton in which implementers can transport the information needed to realize conference control mechanisms not explicitly envisioned in the CCMP specification; these mechanisms are called, in this context, "extensions". Each extension is assumed to be characterized by an appropriate name that MUST be carried in the extendedRequest/extendedResponse pair in the provided <extensionName> field. Extension-specific information can be transported in the form of schema-defined XML elements inside the <any> element present in both extendedRequest and extendedResponse.

The conferencing client SHOULD be able to determine the extensions supported by a CCMP server and to recover the XML schema defining the related specific elements by means of an optionsRequest/optionsResponse CCMP transaction (see Section 5.3.12).

The meaning of the common CCMP parameters inherited by the extendedRequest and extendedResponse from the basic CCMP request and response messages SHOULD be preserved and exploited appropriately while defining an extension.

```
   <!-- extendedRequest -->

   <xs:complexType name="ccmp-extended-request-message-type">
      <xs:complexContent>
          <xs:extension base="tns:ccmp-request-message-type">
             <xs:sequence>
                          <xs:element ref="extendedRequest"/>
             </xs:sequence>
          </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- extendedRequestType -->

   <xs:element name="extendedRequest" type="extendedRequestType"/>

   <xs:complexType name="extendedRequestType">
     <xs:sequence>
           <xs:element name="extensionName"
                       type="xs:string" minOccurs="1"/>
           <xs:any namespace="##other" processContents="lax"
                   minOccurs="0"
               maxOccurs="unbounded" />
    </xs:sequence>
   </xs:complexType>

   <!-- extendedResponse -->

   <xs:complexType name="ccmp-extended-response-message-type">
      <xs:complexContent>
          <xs:extension base="tns:ccmp-response-message-type">
              <xs:sequence>
                          <xs:element ref="extendedResponse"/>
              </xs:sequence>
          </xs:extension>
       </xs:complexContent>
   </xs:complexType>
```

```
<!-- extendedResponseType -->

<xs:element name="extendedResponse" type="extendedResponseType"/>

<xs:complexType name="extendedResponseType">
        <xs:sequence>
                <xs:element name="extensionName"
                            type="xs:string"
                            minOccurs="1"/>
                <xs:any namespace="##other"
                        processContents="lax"
                        minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
</xs:complexType>
```

Figure 17: Structure of the extendedRequest and extendedResponse Messages

5.3.12. optionsRequest and optionsResponse

The optionsRequest message (Figure 18) retrieves general information about conference server capabilities. These capabilities include the standard CCMP messages (request/response pairs) and potential extension messages supported by the conference server. As such, it is a basic CCMP message, rather than a specialization of the general CCMP request.

The optionsResponse returns, in the appropriate <options> field, a list of the supported CCMP message pairs as defined in this specification. These messages are in the form of a list: <standard-message-list> including each of the supported messages as reflected by <standard-message> elements. The optionsResponse message also allows for an <extended-message-list>, which is a list of additional message types in the form of <extended-message-list> elements that are currently undefined, to allow for future extensibility. The following information is provided for both types of messages:

- <name> (REQUIRED): in case of standard messages, it can be one of the 10 standard message names defined in this document (i.e., "blueprintsRequest", "confsRequest", etc.). In case of extensions, this element MUST carry the same value of the <extension-name> inserted in the corresponding extendedRequest/ extendedResponse message pair.

- <operations> (OPTIONAL): this field is a list of <operation> entries, each representing the Create, Read, Update, Delete (CRUD) operation supported by the server for the message. If this element is absent, the client SHOULD assume the server is able to

handle the entire set of CRUD operations or, in case of standard messages, all the operations envisioned for that message in this document.

- <schema-ref> (OPTIONAL): since all CCMP messages can potentially contain XML elements not envisioned in the CCMP schema (due to the presence of <any> elements and attributes), a reference to a proper schema definition specifying such new elements/attributes can also be sent back to the clients by means of such field. If this element is absent, no new elements are introduced in the messages other than those explicitly defined in the CCMP specification.

- <description> (OPTIONAL): human-readable information about the related message.

The only parameter needed in the optionsRequest is the sender confUserID, which is mirrored in the same parameter of the corresponding optionsResponse.

The CCMP server MUST include the <standard-message-list> containing at least one <operation> element in the optionsResponse, since a CCMP server is REQUIRED to be able to handle both the request and response messages for at least one of the operations.

```
<!-- optionsRequest -->

<xs:complexType name="ccmp-options-request-message-type">
        <xs:complexContent>
                <xs:extension base="tns:ccmp-request-message-type"/>
        </xs:complexContent>
</xs:complexType>

<!-- optionsResponse -->

<xs:complexType name="ccmp-options-response-message-type">
   <xs:complexContent>
      <xs:extension base="tns:ccmp-response-message-type">
        <xs:sequence>
          <xs:element ref="optionsResponse"/>
        </xs:sequence>
      </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
   <!-- optionsResponseType -->

   <xs:element name="optionsResponse"
                  type="optionsResponseType" />

   <xs:complexType name="optionsResponseType">
     <xs:sequence>
      <xs:element name="options"
                type="options-type" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- options-type -->

   <xs:complexType name="options-type">
      <xs:sequence>
           <xs:element name="standard-message-list"
                    type="standard-message-list-type"
                    minOccurs="1"/>
       <xs:element name="extended-message-list"
                    type="extended-message-list-type"
                    minOccurs="0"/>
       <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- standard-message-list-type -->

   <xs:complexType name="standard-message-list-type">
     <xs:sequence>
           <xs:element name="standard-message"
                       type="standard-message-type"
                       minOccurs="1" maxOccurs="10"/>
       <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>
```

```
   <!-- standard-message-type -->

   <xs:complexType name="standard-message-type">
      <xs:sequence>
           <xs:element name="name"
                       type="standard-message-name-type"
                       minOccurs="1"/>
           <xs:element name="operations"
                       type="operations-type"
                       minOccurs="0"/>
           <xs:element name="schema-def"
                       type="xs:string" minOccurs="0"/>
           <xs:element name="description"
                       type="xs:string" minOccurs="0"/>
       <xs:any namespace="##other" processContents="lax"
               minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- standard-message-name-type -->

   <xs:simpleType name="standard-message-name-type">
      <xs:restriction base="xs:token">
       <xs:enumeration value="confsRequest"/>
       <xs:enumeration value="confRequest"/>
       <xs:enumeration value="blueprintsRequest"/>
       <xs:enumeration value="blueprintRequest"/>
       <xs:enumeration value="usersRequest"/>
       <xs:enumeration value="userRequest"/>
       <xs:enumeration value="sidebarsByValRequest"/>
       <xs:enumeration value="sidebarByValRequest"/>
       <xs:enumeration value="sidebarsByRefRequest"/>
       <xs:enumeration value="sidebarByRefRequest"/>
      </xs:restriction>
   </xs:simpleType>
```

```
<!-- operations-type -->

<xs:complexType name="operations-type">
  <xs:sequence>
     <xs:element name="operation" type="operationType"
                 minOccurs="1" maxOccurs="4"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

Figure 18: Structure of the optionsRequest and optionsResponse Messages

5.4. CCMP Response Codes

All CCMP response messages MUST include a <response-code>. This document defines an IANA registry for the CCMP response codes, as described in Section 12.5.2. The following summarizes the CCMP response codes:

200 Success:

Successful completion of the requested operation.

400 Bad Request:

Syntactically malformed request.

401 Unauthorized:

User not allowed to perform the required operation.

403 Forbidden:

Operation not allowed (e.g., cancellation of a blueprint).

404 Object Not Found:

The target conference object does not exist at the server (The object in the error message refers to the <confObjID> parameter in the generic request message).

409 Conflict:

A generic error associated with all those situations in which a requested client operation cannot be successfully completed by the server. An example of such a situation is when the modification of an object cannot be applied due to conflicts arising at the

server's side, e.g., because the client version of the object is an obsolete one and the requested modifications collide with the up-to-date state of the object stored at the server. Such code would also be used if a client attempts to create an object (conference or user) with an entity that already exists.

420 User Not Found:

Target user missing at the server (it is related to the XCON-USERID in the 'entity' attribute of the <userInfo> parameter when it is included in userRequests).

421 Invalid confUserID:

User does not exist at the server (This code is returned for requests where the <confUserID> parameter is invalid).

422 Invalid Conference Password:

The password for the target conference object contained in the request is wrong.

423 Conference Password Required:

"conference-password" missing in a request to access a password-protected conference object.

424 Authentication Required:

User's authentication information is missing or invalid.

425 Forbidden Delete Parent:

Cancel operation failed since the target object is a parent of child objects that depend on it, or because it affects, based on the "parent-enforceable" mechanism, the corresponding element in a child object.

426 Forbidden Change Protected:

Update refused by the server because the target element cannot be modified due to its implicit dependence on the value of a parent object ("parent-enforceable" mechanism).

427 Invalid Domain Name:

The domain name in an AUTO_GENERATE_X instance in the conference object is not within the CCMP server's domain of responsibility.

500 Server Internal Error:

The server cannot complete the required service due to a system internal error.

501 Not Implemented:

Operation defined by the protocol, but not implemented by this server.

510 Request Timeout:

The time required to serve the request has exceeded the configured service threshold.

511 Resources Not Available:

This code is used when the CCMP server cannot execute a command because of resource issues, e.g., it cannot create a sub-conference because the system has reached its limits on the number of sub-conferences, or if a request for adding a new user fails because the max number of users has been reached for the conference or the max number of users has been reached for the conferencing system.

The handling of a <response-code> of "404", "409", "420", "421", "425", "426", or "427" is only applicable to specific operations for specialized message responses and the details are provided in Section 5.3. The following table summarizes these response codes and the specialized message and operation to which they are applicable:

Response code	Create	Retrieve	Update	Delete
404	userRequest sidebarByValRequest, sidebarsByRefRequest	All retrieve requests EXCEPT: blueprintsRequest, confsRequest	All update requests	All delete requests
409	N/A	N/A	All update requests	N/A
420	userRequest (third-party invite with third-user entity) (*)	userRequest	userRequest	userRequest
421	All create requests EXCEPT: userRequest with no confUserID (**)	All retrieve requests	All update requests	All delete requests
425	N/A	N/A	N/A	All delete request
426	N/A	N/A	All update requests	N/A
427	ConfRequest UserRequest	N/A	All update requests	N/A

Table 2: Response Codes and Associated Operations

(*) "420" in answer to a "userRequest/create" operation: In the case of a third-party invite, this code can be returned if the <confUserID> (contained in the 'entity' attribute of the <userInfo> parameter) of the user to be added is unknown. In the case above, if instead it is the <confUserID> parameter of the sender of the request that is invalid, a <response-code> of "421" is returned to the client.

(**) "421" is not sent in answer to userRequest/create messages having a "null" confUserID, since this case is associated with a user who is unaware of his own XCON-USERID, but wants to enter a known conference.

In the case of a <response-code> of "510", a conferencing client MAY re-attempt the request within a period of time that would be specific to a conferencing client or conference server.

A <response-code> of "400" indicates that the conferencing client sent a malformed request, which is indicative of an error in the conferencing client or in the conference server. The handling is specific to the conferencing client implementation (e.g., generate a log, display an error message, etc.). It is NOT RECOMMENDED that the client re-attempt the request in this case.

A <response-code> of "401" or "403" indicates the client does not have the appropriate permissions, or there is an error in the permissions: re-attempting the request would likely not succeed and thus it is NOT RECOMMENDED.

Any unexpected or unknown <response-code> SHOULD be treated by the client in the same manner as a <response-code> of "500", the handling of which is specific to the conferencing client implementation.

6. A Complete Example of CCMP in Action

In this section a typical, non-normative, scenario in which CCMP comes into play is described, by showing the actual composition of the various CCMP messages. In the call flows of the example, the conferencing client is a CCMP-enabled client, and the conference server is a CCMP-enabled server. The XCON-USERID of the client, Alice, is "xcon-userid:alice@example.com" and it appears in the <confUserID> parameter in all requests. The sequence of operations is as follows:

1. Alice retrieves the list of available blueprints from the server (Section 6.1);

2. Alice asks for detailed information about a specific blueprint (Section 6.2);

3. Alice decides to create a new conference by cloning the retrieved blueprint (Section 6.3);

4. Alice modifies information (e.g., XCON-URI, name, and description) associated with the newly created blueprint (Section 6.4);

5. Alice specifies a list of users to be contacted when the conference is activated (Section 6.5);

6. Alice joins the conference (Section 6.6);

7. Alice lets a new user, Ciccio, (whose XCON-USERID is "xcon-userid:Ciccio@example.com") join the conference (Section 6.7).

8. Alice asks for the CCMP server capabilities (Section 6.8);

9. Alice exploits an extension of the CCMP server (Section 6.9).

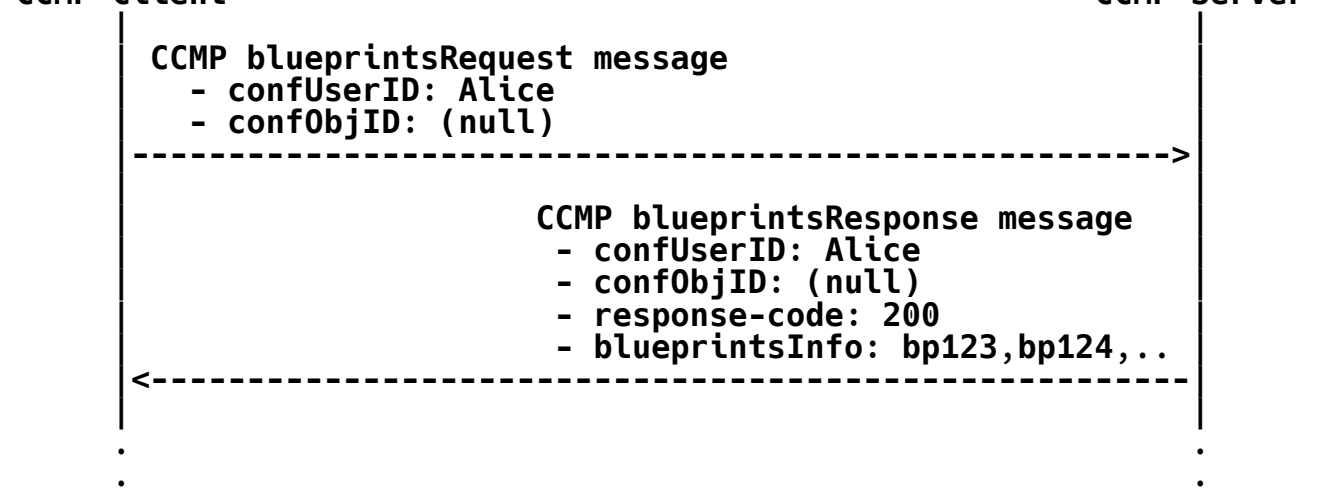
Note that the examples do not include any details beyond the basic operation.

In the following sections, we deal with each of the aforementioned actions separately.

6.1. Alice Retrieves the Available Blueprints

This section illustrates the transaction associated with retrieval of the blueprints, together with a dump of the two messages exchanged (blueprintsRequest and blueprintsResponse). As shown in the figure, the blueprintsResponse message contains, in the <blueprintsInfo> parameter, information about the available blueprints, in the form of the standard XCON-URI of the blueprint, plus additional (and optional) information, like its display-text and purpose.

Alice retrieves from the server the list of available blueprints:



1. blueprintsRequest message:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpRequest
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
    <ccmpRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="ccmp:ccmp-blueprints-request-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <ccmp:blueprintsRequest/>
    </ccmpRequest>
</ccmp:ccmpRequest>
```

2. blueprintsResponse message from the server:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
 xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
 xmlns:info="urn:ietf:params:xml:ns:conference-info"
 xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp">
<ccmpResponse
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="ccmp:ccmp-blueprints-response-message-type">
   <confUserID>xcon-userid:alice@example.com</confUserID>
   <response-code>200</response-code>
     <ccmp:blueprintsResponse>
      <blueprintsInfo>
       <info:entry>
        <info:uri>xcon:AudioRoom@example.com</info:uri>
        <info:display-text>AudioRoom</info:display-text>
        <info:purpose>Simple Room:
           conference room with public access,
           where only audio is available, more users
           can talk at the same time
           and the requests for the AudioFloor
           are automatically accepted.
        </info:purpose>
       </info:entry>
       <info:entry>
        <info:uri>xcon:VideoRoom@example.com</info:uri>
        <info:display-text>VideoRoom</info:display-text>
        <info:purpose>Video Room:
            conference room with public access,
            where both audio and video are available,
            8 users can talk and be seen at the same time,
            and the floor requests are automatically accepted.
        </info:purpose>
```

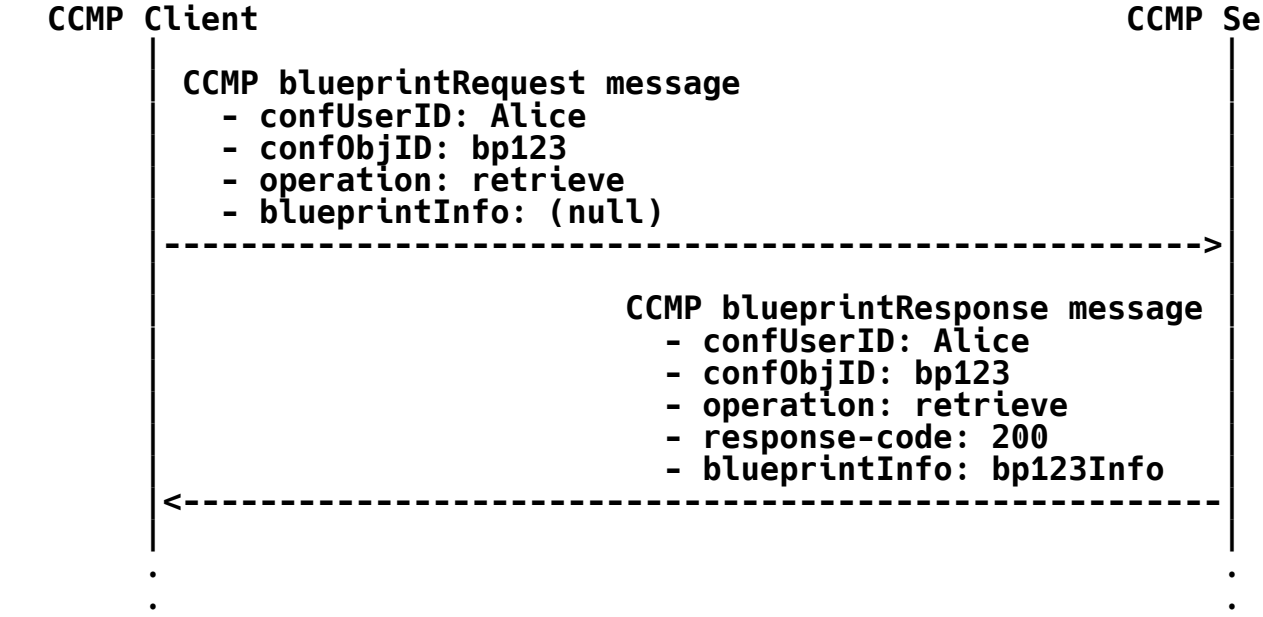
```
          </info:entry>
          <info:entry>
           <info:uri>xcon:AudioConference1@example.com</info:uri>
           <info:display-text>AudioConference1</info:display-text>
           <info:purpose>Public Audio Conference:
                conference with public access,
                where only audio is available,
                only one user can talk at the same time,
                and the requests for the AudioFloor MUST
                be accepted by a Chair.
           </info:purpose>
          </info:entry>
          <info:entry>
           <info:uri>xcon:VideoConference1@example.com</info:uri>
           <info:display-text>VideoConference1</info:display-text>
             <info:purpose>Public Video Conference: conference
                 where both audio and video are available,
                 only one user can talk.
             </info:purpose>
           </info:entry>
           <info:entry>
            <info:uri>xcon:AudioConference2@example.com</info:uri>
            <info:display-text>AudioConference2</info:display-text>
            <info:purpose>Basic Audio Conference:
                 conference with private access,
                 where only audio is available,
                 only one user can talk at the same time,
                 and the requests for the AudioFloor MUST
                 be accepted by a Chair.
            </info:purpose>
           </info:entry>
        </blueprintsInfo>
      </ccmp:blueprintsResponse>
    </ccmpResponse>
  </ccmp:ccmpResponse>
```

Figure 19: Getting Blueprints from the Server

6.2. Alice Gets Detailed Information about a Specific Blueprint

This section illustrates the second transaction in the overall flow. In this case, Alice, who now knows the XCON-URIs of the blueprints available at the server, makes a drill-down query, in the form of a CCMP blueprintRequest message, to get detailed information about one of them (the one called with XCON-URI "xcon:AudioRoom@example.com"). The picture shows such a transaction. Notice that the response contains, in the <blueprintInfo> parameter, a document compliant with the standard XCON data model.

Alice retrieves detailed information about a specified blueprint:



1. blueprintRequest message:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpRequest
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
  <ccmpRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:type="ccmp:ccmp-blueprint-request-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <confObjID>xcon:AudioRoom@example.com</confObjID>
        <operation>retrieve</operation>
        <ccmp:blueprintRequest/>
  </ccmpRequest>
</ccmp:ccmpRequest>
```

2. blueprintResponse message from the server:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp">
 <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="ccmp:ccmp-blueprint-response-message-type">
 <confUserID>xcon-userid:alice@example.com</confUserID>
```

```
  <confObjID>xcon:AudioRoom@example.com</confObjID>
  <operation>retrieve</operation>
  <response-code>200</response-code>
  <response-string>Success</response-string>
  <ccmp:blueprintResponse>
    <blueprintInfo entity="xcon:AudioRoom@example.com">
     <info:conference-description>
        <info:display-text>AudioRoom</info:display-text>
        <info:available-media>
           <info:entry label="audioLabel">
              <info:display-text>audio stream</info:display-text>
              <info:type>audio</info:type>
           </info:entry>
        </info:available-media>
     </info:conference-description>
     <info:users>
       <xcon:join-handling>allow</xcon:join-handling>
      </info:users>
     <xcon:floor-information>
      <xcon:floor-request-handling>confirm</xcon:floor-request-handling>
       <xcon:conference-floor-policy>
           <xcon:floor id="audioFloor">
               <xcon:media-label>audioLabel</xcon:media-label>
           </xcon:floor>
       </xcon:conference-floor-policy>
     </xcon:floor-information>
    </blueprintInfo>
  </ccmp:blueprintResponse>
 </ccmpResponse>
</ccmp:ccmpResponse>
```

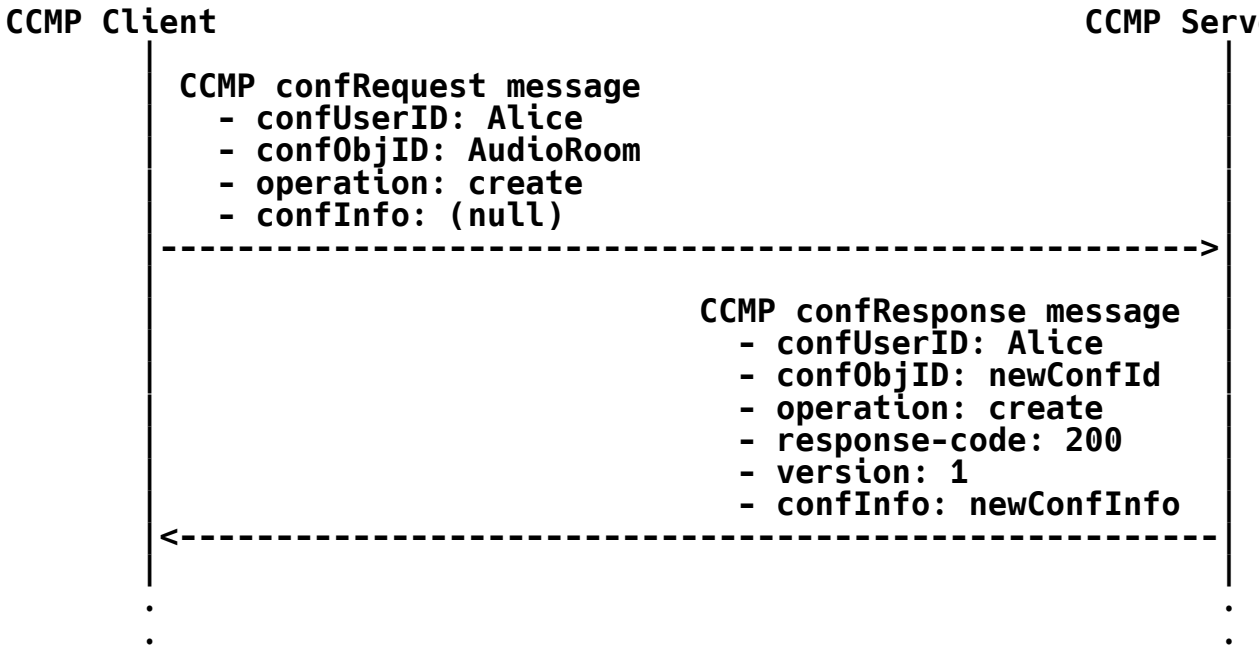
Figure 20: Getting Information about a Specific Blueprint

6.3. Alice Creates a New Conference through a Cloning Operation

This section illustrates the third transaction in the overall flow. Alice decides to create a new conference by cloning the blueprint having XCON-URI "xcon:AudioRoom@example.com", for which she just retrieved detailed information through the blueprintRequest message. This is achieved by sending a confRequest/create message having the blueprint's URI in the <confObjID> parameter. The picture shows such a transaction. Notice that the response contains, in the <confInfo> parameter, the document associated with the newly created conference, which is compliant with the standard XCON data model. The <confObjID> parameter in the response is set to the XCON-URI of the new conference (in this case, "xcon:8977794@example.com"). We also

notice that this value is equal to the value of the 'entity' attribute of the <conference-info> element of the document representing the newly created conference object.

Alice creates a new conference by cloning the "xcon:AudioRoom@example.com" blueprint:



1. confRequest message:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpRequest
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
   <ccmpRequest
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:type="ccmp:ccmp-conf-request-message-type">
      <confUserID>xcon-userid:alice@example.com</confUserID>
      <confObjID>xcon:AudioRoom@example.com</confObjID>
      <operation>create</operation>
      <ccmp:confRequest/>
   </ccmpRequest>
</ccmp:ccmpRequest>
```

2. confResponse message from the server:

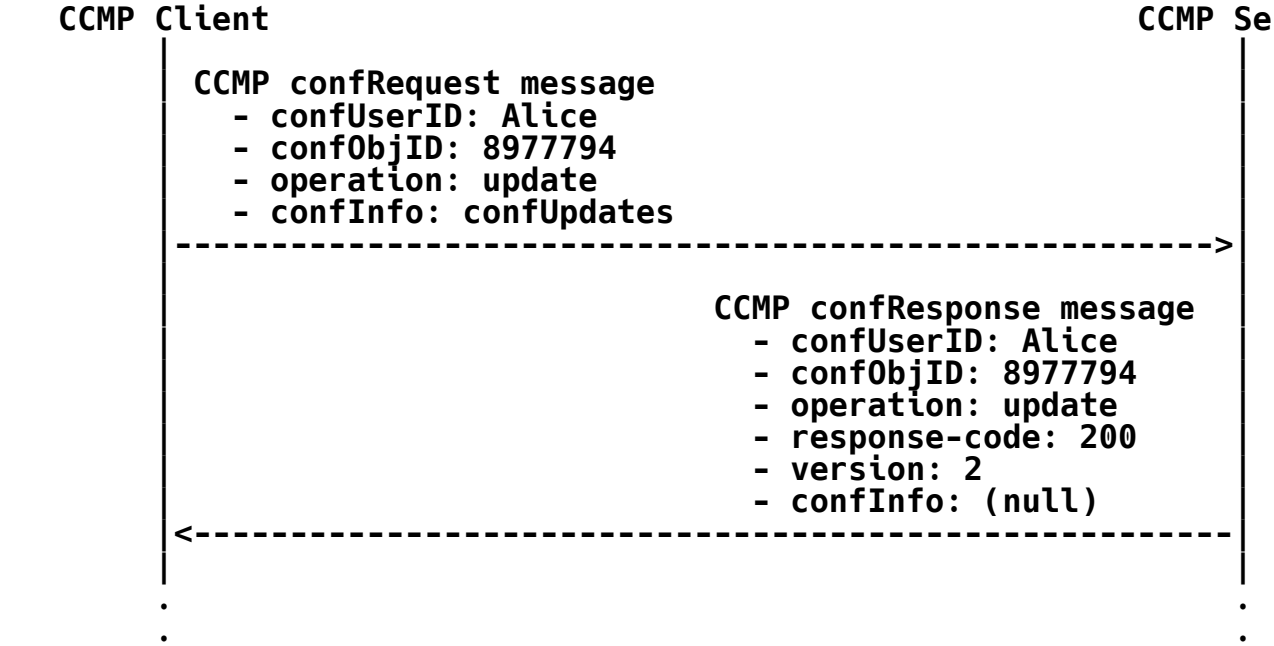
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp">
<ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="ccmp:ccmp-conf-response-message-type">
 <confUserID>xcon-userid:alice@example.com</confUserID>
 <confObjID>xcon:8977794@example.com</confObjID>
 <operation>create</operation>
 <response-code>200</response-code>
 <response-string>Success</response-string>
 <version>1</version>
 <ccmp:confResponse>
   <confInfo entity="xcon:8977794@example.com">
     <info:conference-description>
       <info:display-text>
              New conference by Alice cloned from AudioRoom
       </info:display-text>
       <info:available-media>
              <info:entry label="333">
                <info:display-text>audio stream</info:display-text>
                <info:type>audio</info:type>
              </info:entry>
        </info:available-media>
      </info:conference-description>
      <info:users>
          <xcon:join-handling>allow</xcon:join-handling>
      </info:users>
      <xcon:floor-information>
      <xcon:floor-request-handling>confirm</xcon:floor-request-handling>
       <xcon:conference-floor-policy>
          <xcon:floor id="11">
             <xcon:media-label>333</xcon:media-label>
          </xcon:floor>
       </xcon:conference-floor-policy>
      </xcon:floor-information>
     </confInfo>
    </ccmp:confResponse>
  </ccmpResponse>
</ccmp:ccmpResponse>
```

Figure 21: Creating a New Conference by Cloning a Blueprint

6.4. Alice Updates Conference Information

This section illustrates the fourth transaction in the overall flow. Alice decides to modify some of the details associated with the conference she just created. More precisely, she changes the <display-text> element under the <conference-description> element of the document representing the conference. This is achieved through a confRequest/update message carrying the fragment of the conference document to which the required changes have to be applied. As shown in the picture, the response contains a code of "200", which acknowledges the modifications requested by the client, while also updating the conference version number from 1 to 2, as reflected in the "version" parameter.

Alice updates information about the conference she just created:



1. confRequest message:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpRequest
     xmlns:info="urn:ietf:params:xml:ns:conference-info"
           xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
     xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
  <ccmpRequest
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="ccmp:ccmp-conf-request-message-type">
    <confUserID>xcon-userid:alice@example.com</confUserID>
```

```
     <confObjID>xcon:8977794@example.com</confObjID>
     <operation>update</operation>
     <ccmp:confRequest>
          <confInfo entity="xcon:8977794@example.com">
             <info:conference-description>
               <info:display-text>
                  Alice's conference
               </info:display-text>
             </info:conference-description>
          </confInfo>
       </ccmp:confRequest>
   </ccmpRequest>
 </ccmp:ccmpRequest>
```

2. confResponse message from the server:

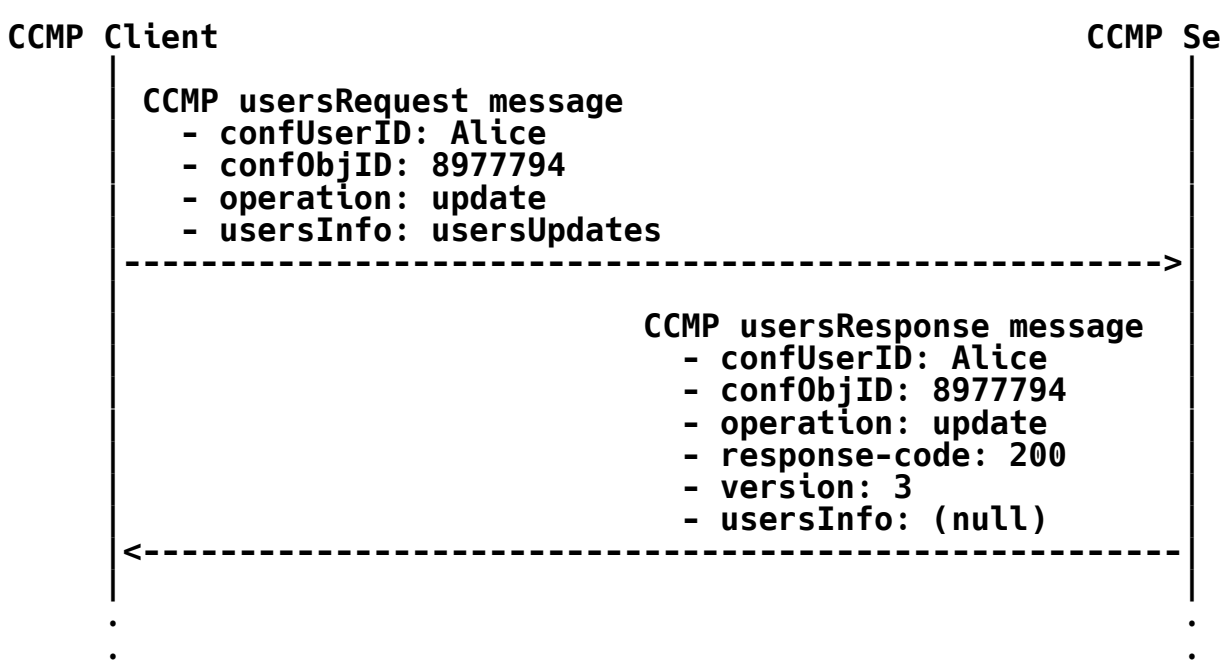
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
     xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
     xmlns:info="urn:ietf:params:xml:ns:conference-info"
     xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp">
    <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="ccmp:ccmp-conf-response-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <confObjID>xcon:8977794@example.com</confObjID>
        <operation>update</operation>
        <response-code>200</response-code>
        <response-string>Success</response-string>
        <version>2</version>
        <ccmp:confResponse/>
    </ccmpResponse>
</ccmp:ccmpResponse>
```

Figure 22: Updating Conference Information

6.5. Alice Inserts a List of Users into the Conference Object

This section illustrates the fifth transaction in the overall flow. Alice modifies the <allowed-users-list> under the <users> element in the document associated with the conference she created. To achieve this, she makes use of the usersRequest message provided by CCMP.

Alice updates information about the list of users to whom access to the conference is permitted:



1. usersRequest message:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpRequest
     xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
         xmlns:info="urn:ietf:params:xml:ns:conference-info"
         xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp">
    <ccmpRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:type="ccmp:ccmp-users-request-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <confObjID>xcon:8977794@example.com</confObjID>
        <operation>update</operation>
        <ccmp:usersRequest>
            <usersInfo>
                <xcon:allowed-users-list>
                    <xcon:target method="dial out"
                                 uri="xmpp:cicciolo@pippozzo.com"/>
                    <xcon:target method="refer"
                                 uri="tel:+1-972-555-1234"/>
                    <xcon:target method="refer"
                                 uri="sip:Carol@example.com"/>
                </xcon:allowed-users-list>
            </usersInfo>
        </ccmp:usersRequest>
    </ccmpRequest>
</ccmp:ccmpRequest>
```

2. usersResponse message from the server:

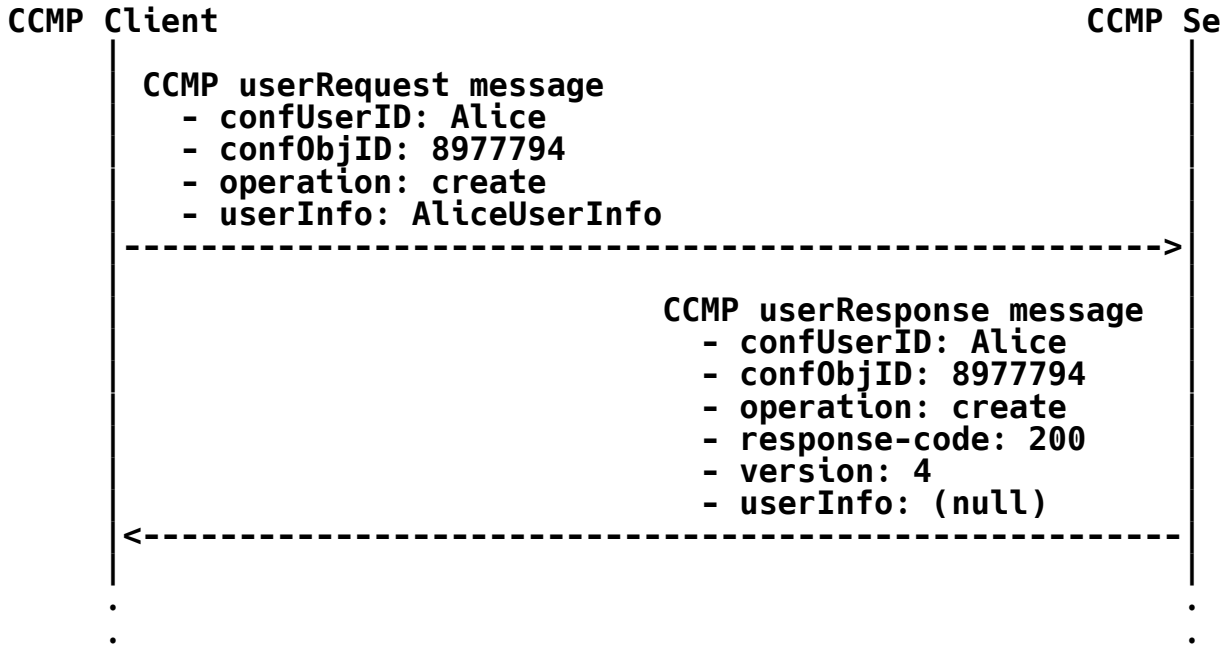
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp">
    <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="ccmp:ccmp-users-response-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <confObjID>xcon:8977794@example.com</confObjID>
        <operation>retrieve</operation>
        <response-code>200</response-code>
        <response-string>Success</response-string>
        <version>3</version>
        <ccmp:usersResponse/>
    </ccmpResponse>
</ccmp:ccmpResponse>
```

Figure 23: Updating the List of Allowed Users for the Conference 'xcon:8977794@example.com'

6.6. Alice Joins the Conference

This section illustrates the sixth transaction in the overall flow. Alice uses CCMP to add herself to the newly created conference. This is achieved through a userRequest/create message containing, in the <userInfo> parameter, a <user> element compliant with the XCON data model representation. Notice that such an element includes information about the user's Addresses of Record, as well as her current endpoint. The picture below shows the transaction. Notice how the <confUserID> parameter is equal to the 'entity' attribute of the <userInfo> element, which indicates that the request issued by the client is a first-party one.

Alice joins the conference by issuing a userRequest/create message with her own ID to the server:



1. userRequest message:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpRequest
       xmlns:info="urn:ietf:params:xml:ns:conference-info"
           xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
           xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
    <ccmpRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="ccmp:ccmp-user-request-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <confObjID>xcon:8977794@example.com</confObjID>
                <operation>create</operation>
        <ccmp:userRequest>
            <userInfo entity="xcon-userid:alice@example.com">
                <info:associated-aors>
                    <info:entry>
                        <info:uri>
                           mailto:Alice83@example.com
                        </info:uri>
                        <info:display-text>email</info:display-text>
                    </info:entry>
                </info:associated-aors>
                <info:endpoint entity="sip:alice_789@example.com"/>
            </userInfo>
        </ccmp:userRequest>
    </ccmpRequest>
</ccmp:ccmpRequest>
```

2. userResponse message from the server:

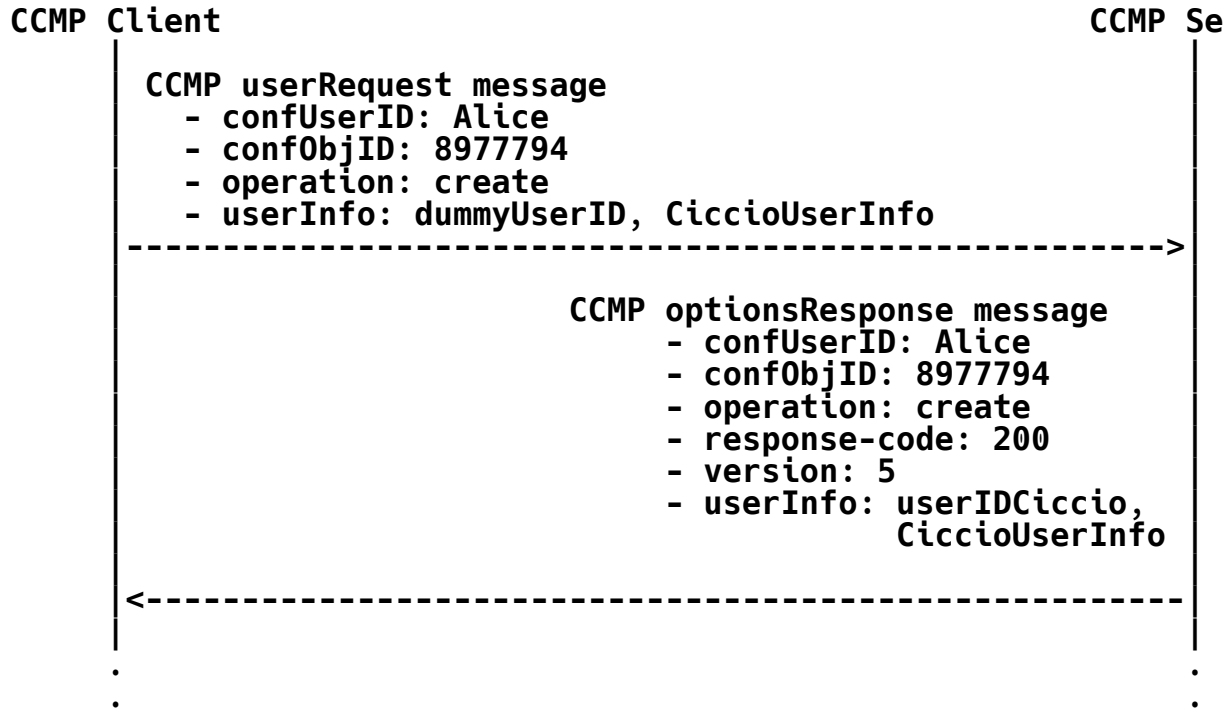
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp">
    <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="ccmp:ccmp-user-response-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <confObjID>xcon:8977794@example.com</confObjID>
        <operation>create</operation>
        <response-code>200</response-code>
        <response-string>Success</response-string>
        <version>4</version>
        <ccmp:userResponse/>
    </ccmpResponse>
</ccmp:ccmpResponse>
```

Figure 24: Alice Joins the Conference through CCMP

6.7. Alice Adds a New User to the Conference

This section illustrates the seventh and last transaction in the overall flow. Alice uses CCMP to add a new conferencing system user, Ciccio, to the conference. This "third-party" request is realized through a userRequest/create message containing, in the <userInfo> parameter, a <user> element compliant with the XCON data model representation. Notice that such an element includes information about Ciccio's Addresses of Record, as well as his current endpoint, but has a placeholder 'entity' attribute, "AUTO_GENERATE_1@example.com" as discussed in Section 4.3, since the XCON-USERID is initially unknown to Alice. Thus, the conference server is in charge of generating a new XCON-USERID for the user Alice indicates (i.e., Ciccio), and returning it in the 'entity' attribute of the <userInfo> parameter carried in the response, as well as adding the user to the conference. The picture below shows the transaction.

Alice adds user "Ciccio" to the conference by issuing a third-party userRequest/create message to the server:



1. "third-party" userRequest message from Alice:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpRequest
       xmlns:info="urn:ietf:params:xml:ns:conference-info"
       xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
       xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
    <ccmpRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:type="ccmp:ccmp-user-request-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <confObjID>xcon:8977794@example.com</confObjID>
        <operation>create</operation>
        <ccmp:userRequest>
            <userInfo entity="xcon-userid:AUTO_GENERATE_1@example.com">
                <info:associated-aors>
                    <info:entry>
                        <info:uri>
                            mailto:Ciccio@example.com
                        </info:uri>
                        <info:display-text>email</info:display-text>
                    </info:entry>
                </info:associated-aors>
                <info:endpoint entity="sip:Ciccio@example.com"/>
            </userInfo>
```

```
        </ccmp:userRequest>
    </ccmpRequest>
</ccmp:ccmpRequest>
```

2. "third-party" userResponse message from the server:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 <ccmp:ccmpResponse
        xmlns:info="urn:ietf:params:xml:ns:conference-info"
        xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
        xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
     <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:type="ccmp:ccmp-user-response-message-type">
         <confUserID>xcon-userid:alice@example.com</confUserID>
         <confObjID>xcon:8977794@example.com</confObjID>
         <operation>create</operation>
         <response-code>200</response-code>
         <version>5</version>
         <ccmp:userResponse>
                 <userInfo entity="xcon-userid:Ciccio@example.com">
                 <info:associated-aors>
                     <info:entry>
                         <info:uri>
                             mailto:Ciccio@example.com
                         </info:uri>
                         <info:display-text>email</info:display-text>
                     </info:entry>
                 </info:associated-aors>
                 <info:endpoint entity="sip:Ciccio@example.com"/>
             </userInfo>
         </ccmp:userResponse>
     </ccmpResponse>
 </ccmp:ccmpResponse>
```

Figure 25: Alice Adds a New User to the Conference through CCMP

6.8. Alice Asks for the CCMP Server Capabilities

This section illustrates how Alice can discover which standard CCMP messages and what extensions are supported by the CCMP server with which she interacts through an optionsRequest/optionsResponse transaction.

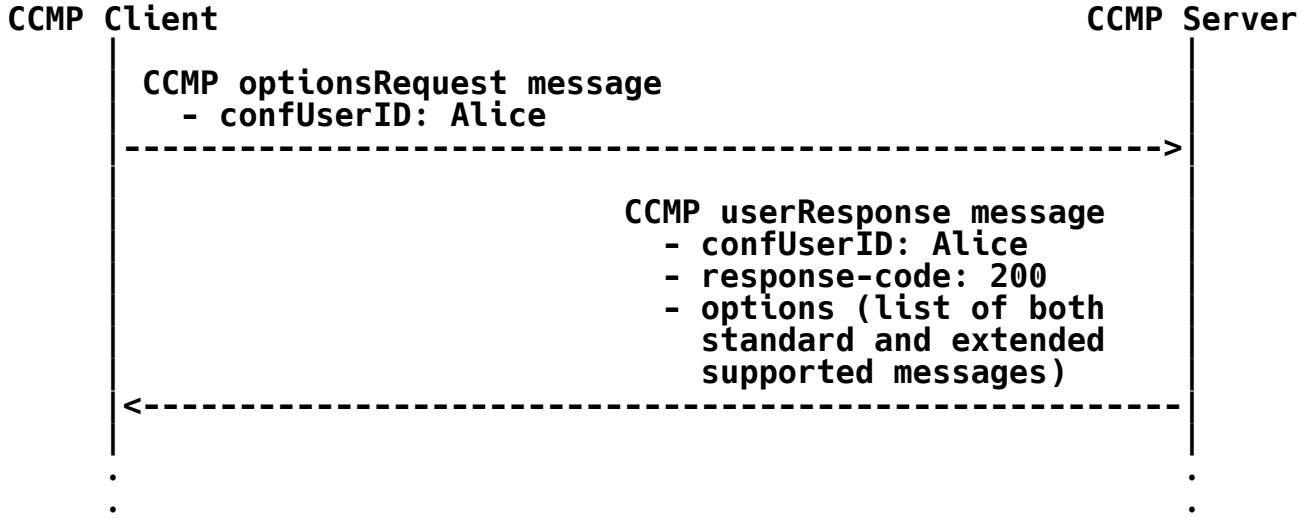
To prepare the optionsRequest, Alice just puts her XCON-USERID in the <confUserID> parameter. Looking at the <options> element in the received optionsResponse, Alice infers the following server capabilities as regards standard CCMP messages:

- the server doesn't support sidebarsByValRequest nor the sidebarByValRequest messages, since they do not appear in the <standard-message-list>;

- the only implemented operation for the blueprintRequest message is "retrieve", since no other <operation> entries are included in the related <operations> field.

By analyzing the <extended-message-list>, Alice discovers the server implements a bluePrint extension, referred to as "confSummaryRequest" in this example. This extension allows Alice to recover via CCMP a brief description of a specific conference; the XML elements involved in this extended conference control transaction are available at the URL indicated in the <schema-ref> element, and the only operation provided by this extension is "retrieve". To better understand how Alice can exploit the "confSummaryRequest" extension via CCMP, see Section 6.9.

The figure below shows the optionsRequest/optionsResponse message exchange between Alice and the CCMP server.



1. optionsRequest (Alice asks for CCMP server capabilities)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ccmp:ccmpRequest
        xmlns:info="urn:ietf:params:xml:ns:conference-info"
        xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
        xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
    <ccmpRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:type="ccmp:ccmp-options-request-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
```

```
    </ccmpRequest>
  </ccmp:ccmpRequest>
```

2. optionsResponse (the server returns the list of its conference control capabilities)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ccmp:ccmpResponse
         xmlns:info="urn:ietf:params:xml:ns:conference-info"
         xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
         xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
    <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="ccmp:ccmp-options-response-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <response-code>200</response-code>
        <response-string>success</response-string>
        <ccmp:optionsResponse>
           <options>
              <standard-message-list>
                 <standard-message>
                    <name>blueprintsRequest</name>
                 </standard-message>
                 <standard-message>
                   <name>blueprintRequest</name>
                   <operations>
                     <operation>retrieve</operation>
                   </operations>
                 </standard-message>
                 <standard-message>
                   <name>confsRequest</name>
                 </standard-message>
                 <standard-message>
                   <name>confRequest</name>
                 </standard-message>
                 <standard-message>
                    <name>usersRequest</name>
                 </standard-message>
                 <standard-message>
                    <name>userRequest</name>
                 </standard-message>
                 <standard-message>
                    <name>sidebarsByRefRequest</name>
                 </standard-message>
                 <standard-message>
                    <name>sidebarByRefRequest</name>
                 </standard-message>
              </standard-message-list>
              <extended-message-list>
```

```
                <extended-message>
                   <name>confSummaryRequest</name>
                   <operations>
                     <operation>retrieve</operation>
                   </operations>
                   <schema-def>
                        http://example.com/ccmp-extension-schema.xsd
                   </schema-def>
                   <description>
                        confSummaryRequest is intended
                        to allow the requestor to retrieve
                        a brief description
                        of the conference indicated in the
                        confObjID request parameter
                   </description>
                </extended-message>
             </extended-message-list>
          </options>
       </ccmp:optionsResponse>
    </ccmpResponse>
  </ccmp:ccmpResponse>
```

Figure 26: Alice Asks for the Server Control Capabilities

6.9. Alice Makes Use of a CCMP Server Extension

In this section, a very simple example of CCMP extension support is provided. Alice can recover information about this and other server-supported extensions by issuing an optionsRequest (see Section 6.8).

The extension in question is named "confSummaryRequest" and allows a CCMP client to obtain from the CCMP server synthetic information about a specific conference. The conference summary is carried in the form of an XML element as follows:

```
  <?xml version="1.0" encoding="UTF-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://example.com/ccmp-extension"
               xmlns="http://example.com/ccmp-extension">

    <xs:element name="confSummary" type="conf-summary-type"/>

    <xs:complexType name="conf-summary-type">
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="status" type="xs:string"/>
        <xs:element name="public" type="xs:boolean"/>
        <xs:element name="media" type="xs:string"/>
```

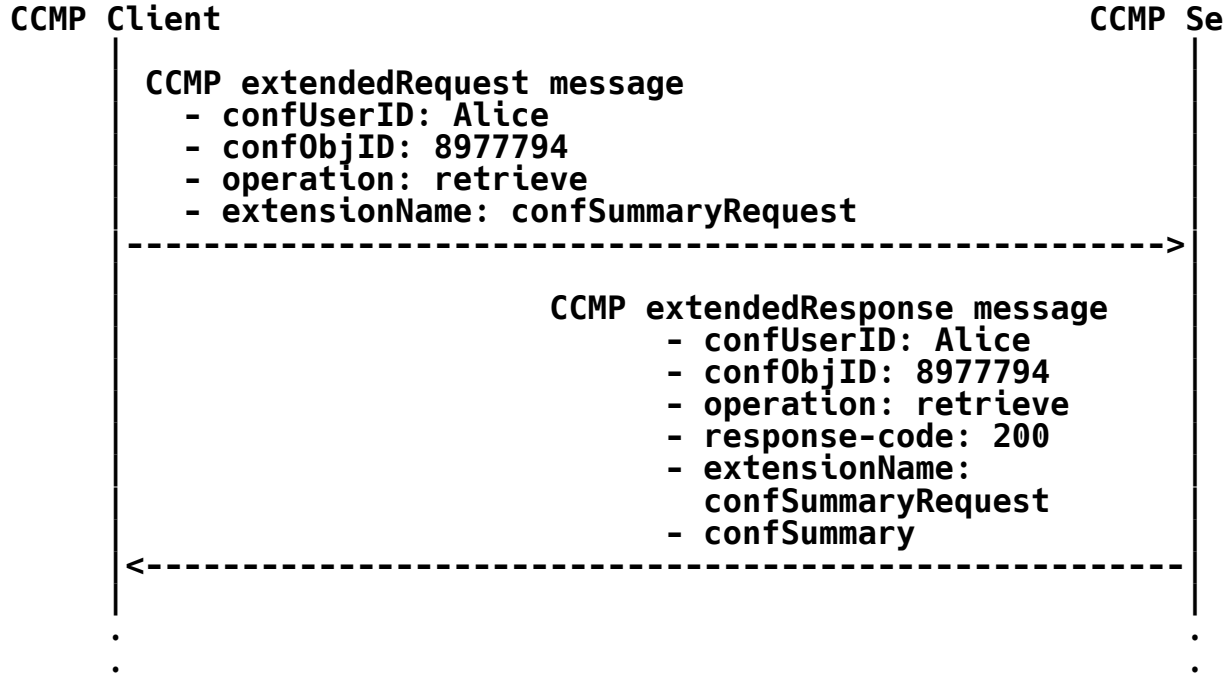
```
        </xs:sequence>
      </xs:complexType>

    </xs:schema>
```

Figure 27: Example of XML Schema defining an extension parameter (ccmp-extension-schema.xsd)

As can be inferred from the schema file, the <confSummary> element contains conference information related to the following:

- title
- status (active or registered)
- participation modality (if everyone is allowed to participate, the boolean <public> element is set to "true")
- involved media

In order to retrieve a conference summary related to the conference she participates in, Alice sends to the CCMP server an extendedRequest with a "confSummaryRequest" <extensionName>, specifying the conference XCON-URI in the confObjID request parameter, as depicted in the figure below.



1. extendedRequest (Alice makes use of the "confSummaryRequest")

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ccmp:ccmpRequest xmlns:info="urn:ietf:params:xml:ns:conference-info"
        xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
        xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
        xmlns:example="http://example.com/ccmp-extension">
    <ccmpRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:type="ccmp:ccmp-extended-request-message-type">
        <confUserID>xcon-userid:alice@example.com</confUserID>
        <confObjID>xcon:8977794@example.com</confObjID>
        <operation>retrieve</operation>
        <ccmp:extendedRequest>
               <extensionName>confRequestSummary</extensionName>
        </ccmp:extendedRequest>
    </ccmpRequest>
  </ccmp:ccmpRequest>
```

2. extendedResponse (the server provides Alice with a brief description of the desired conference)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ccmp:ccmpResponse xmlns:info="urn:ietf:params:xml:ns:conference-info"
          xmlns:ccmp="urn:ietf:params:xml:ns:xcon-ccmp"
          xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
          xmlns:example="http://example.com/ccmp-extension">
    <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:type="ccmp:ccmp-extended-response-message-type">
         <confUserID>xcon-userid:alice@example.com</confUserID>
         <confObjID>xcon:8977794@example.com</confObjID>
         <operation>retrieve</operation>
         <response-code>200</response-code>
         <response-string>success</response-string>
         <ccmp:extendedResponse>
           <extensionName>confSummaryRequest</extensionName>
           <example:confSummary>
               <title> Alice's conference </title>
               <status> active </status>
               <public> true </public>
               <media> audio </media>
           </example:confSummary>
         </ccmp:extendedResponse>
     </ccmpResponse>
  </ccmp:ccmpResponse>
```

Figure 28: Alice Exploits the 'confSummaryRequest' Extension

7. Locating a Conference Server

If a conferencing client is not pre-configured to use a specific conference server for the requests, the client MUST first discover the conference server before it can send any requests. The result of the discovery process, is the address of the server supporting conferencing. In this document, the result is an http: or https: URI, which identifies a conference server.

DNS is RECOMMENDED to be used to locate a conference server in the case that the client is not pre-configured to use a specific conference server. URI-Enabled NAPTR (U-NAPTR) resolution for conferencing takes a domain name as input and produces a URI that identifies the conference server. This process also requires an Application Service tag and an Application Protocol tag, which differentiate conferencing-related NAPTR records from other records for that domain.

Section 12.4.1 defines an Application Service tag of "XCON", which is used to identify the centralized conferencing (XCON) server for a particular domain. The Application Protocol tag "CCMP", defined in Section 12.4.2, is used to identify an XCON server that understands CCMP.

The NAPTR records in the following example (Figure 29) demonstrate the use of the Application Service and Application Protocol tags. Iterative NAPTR resolution is used to delegate responsibility for the conferencing service from "zonea.example.com." and "zoneb.example.com." to "outsource.example.com.".

```
zonea.example.com.
;;       order pref flags
IN NAPTR 100   10   ""  "XCON-CCMP" (     ; service
""                                        ; regex
outsource.example.com.                    ; replacement
)
zoneb.example.com.
;;       order pref flags
IN NAPTR 100   10   ""  "XCON-CCMP" (     ; service
""                                        ; regex
outsource.example.com.                    ; replacement
)
outsource.example.com.
;;       order pref flags
IN NAPTR 100   10   "u"  "XCON-CCMP" (    ; service
"!*.!https://confs.example.com/!"         ; regex
.                                         ; replacement
)
```

Figure 29: Sample XCON-CCMP Service NAPTR Records

Details for the "XCON" Application Service tag and the "CCMP" Application Protocol tag are included in Section 12.4.

8. Managing Notifications

As per [RFC5239], CCMP is one of the following four protocols, which have been formally identified within the XCON framework:

Conference Control Protocol:

> mediates between conference and media control client (conferencing client) and conference server. This document describes such a protocol.

Binary floor Control Protocol:

> operates between the floor control client and the floor control server. An example of such a protocol is the Binary Floor Control Protocol (BFCP), specified in [RFC4582].

Call Signaling Protocol:

> operates between the Call Signaling Client and the focus. Examples of call signaling protocols include SIP, H.323 and IAX. Such protocols are capable of negotiating a conferencing session.

Notification Protocol:

operates between the Notification Client and the XCON Notification Service. This specification does not define a new notification protocol. For clients that use SIP as the call signaling protocol, the XCON event package [RFC6502] MUST be used by the client for notifications of changes in the conference data as described below.

The protocol specified in this document is a proactive one and is used by a conferencing client to send requests to a conference server in order to retrieve information about the conference objects stored by the server and to possibly manipulate them. However, a complete conferencing solution is not prohibited from providing clients with a means for receiving asynchronous updates about the status of the objects available at the server. The notification protocol, while conceptually independent of all the mentioned companion protocols, can nonetheless be chosen in a way that is consistent with the overall protocol architecture characterizing a specific deployment, as discussed in the following.

When the conferencing control client uses SIP [RFC3261] as the signaling protocol to participate in the conference, SIP event notification can be used. In such a case, the conferencing control client MUST implement the conference event package for XCON [RFC6502]. This is the default mechanism for conferencing clients as is SIP for signaling per the XCON framework [RFC5239].

In the case where the interface to the conference server is entirely web based, there is a common mechanism for web-based systems that could be used -- a "call back". With this mechanism, the conferencing client provides the conference server with an HTTP URL that is invoked when a change occurs. This is a common implementation mechanism for e-commerce. This works well in the scenarios whereby the conferencing client is a web server that provides the graphical HTML user interface and uses CCMP as the back-end interface to the conference server. This model can coexist with the SIP event notification model. PC-based clients behind NATs could provide a SIP event URI, whereas web-based clients using CCMP in the back end would probably find the HTTP call back approach much easier. The details of this approach are out of scope for CCMP; thus, we expect a future specification will document this solution.

9. HTTP Transport

This section describes the use of HTTP [RFC2616] and HTTP over TLS [RFC2818] as transport mechanisms for CCMP, which a conforming conference server and conferencing client MUST support.

Although CCMP uses HTTP as a transport, it uses a strict subset of HTTP features, and due to the restrictions of some features, a conferencing server might not be a fully compliant HTTP server. It is intended that a conference server can easily be built using an HTTP server with extensibility mechanisms, and that a conferencing client can trivially use existing HTTP libraries. This subset of requirements helps implementers avoid ambiguity with the many options the full HTTP protocol offers.

Support of HTTP authentication [RFC2617] and cookies [RFC6265] is OPTIONAL for a conferencing client that conforms to this specification. These mechanisms are unnecessary because CCMP requests carry their own authentication information (in the "subject" field; see Section 5.1). A conferencing client SHOULD include support for HTTP proxy authentication.

A CCMP request is carried in the body of an HTTP POST request. The conferencing client MUST include a Host header in the request.

The MIME type of CCMP request and response bodies is "application/ccmp+xml". The conference server and conferencing client MUST provide this value in the HTTP Content-Type and Accept header fields. If the conference server does not receive the appropriate Content-Type and Accept header fields, the conference server SHOULD fail the request, returning a 406 (Not Acceptable) response. CCMP responses SHOULD include a Content-Length header.

Conferencing clients MUST NOT use the Expect header or the Range header in CCMP requests. The conference server MAY return 501 (Not Implemented) errors if either of these HTTP features are used. In the case that the conference server receives a request from the conferencing client containing an If-* (conditional) header, the conference server SHOULD return a 412 (precondition failed) response.

The POST method is the only method REQUIRED for CCMP. If a conference server chooses to support GET or HEAD, it SHOULD consider the kind of application doing the GET. Since a conferencing client only uses a POST method, the GET or HEAD MUST be either a URL that was found outside its normal context (e.g., somebody found a URL in protocol traces or log files and fed it into their browser) or somebody is testing or debugging a system. The conference server could provide information in the CCMP response indicating that the URL corresponds to a conference server and only responds to CCMP POST requests or the conference server could instead try to avoid any leak of information by returning a very generic HTTP error message such as 405 (Method Not Allowed).

The conference server populates the HTTP headers of responses so that they are consistent with the contents of the message. In particular, the CacheControl header SHOULD be set to disable caching of any conference information by HTTP intermediaries. Otherwise, there is the risk of stale information and/or the unauthorized disclosure of the information. The HTTP status code MUST indicate a 2xx series response for all CCMP Response and Error messages.

The conference server MAY redirect a CCMP request. A conference server MUST NOT include CCMP responses in a 3xx response. A conferencing client MUST handle redirects by using the Location header provided by the server in a 3xx response. When redirecting, the conferencing client MUST observe the delay indicated by the Retry-After header. The conferencing client MUST authenticate the server that returns the redirect response before following the redirect. A conferencing client SHOULD authenticate the conference server indicated in a redirect.

The conference server SHOULD support persistent connections and request pipelining. If pipelining is not supported, the conference server MUST NOT allow persistent connections. The conference server MUST support termination of a response by the closing of a connection.

Implementations of CCMP that implement HTTP transport MUST implement transport over TLS [RFC2818]. TLS provides message integrity and confidentiality between the conferencing client and the conference server. The conferencing client MUST implement the server authentication method described in HTTPS [RFC2818]. The device uses the URI obtained during conference server discovery to authenticate the server. The details of this authentication method are provided in Section 3.1 of HTTPS [RFC2818]. When TLS is used, the conferencing client SHOULD fail a request if server authentication fails.

10. Security Considerations

As identified in the XCON framework [RFC5239], there are a wide variety of potential attacks related to conferencing, due to the natural involvement of multiple endpoints and the capability to manipulate the data on the conference server using CCMP. Examples of attacks include the following: an endpoint attempting to listen to conferences in which it is not authorized to participate, an endpoint attempting to disconnect or mute other users, and an endpoint theft of service in attempting to create conferences it is not allowed to create.

The following summarizes the security considerations for CCMP:

1. The client MUST determine the proper conference server. The conference server discovery is described in Section 7.

2. The client MUST connect to the proper conference server. The mechanisms for addressing this security consideration are described in Section 10.1.

3. The protocol MUST support a confidentiality and integrity mechanism. As described in Section 9, implementations of CCMP MUST implement the HTTP transport over TLS [RFC2818].

4. There are security issues associated with the authorization to perform actions on the conferencing system to invoke specific capabilities. A conference server SHOULD ensure that only authorized entities can manipulate the conference data. The mechanisms for addressing this security consideration are described in Section 10.2.

5. The privacy and security of the identity of a user in the conference MUST be assured. The mechanisms to ensure the security and privacy of identity are discussed in Section 10.3.

6. A final issue is related to Denial-of-Service (DoS) attacks on the conference server itself. The recommendations to minimize the potential and impact of DoS attacks are discussed in Section 10.4.

Of the considerations listed above, items 1 and 3 are addressed within the referenced sections earlier in this document. The remaining security considerations are addressed in detail in the following sections.

10.1. Assuring That the Proper Conference Server Has Been Contacted

Section 7 describes a mechanism using DNS by which a conferencing client discovers a conference server. A primary concern is spoofed DNS replies; thus, the use of DNS Security (DNSSEC) is RECOMMENDED to ensure that the client receives a valid response from the DNS server in cases where this is a concern.

When the CCMP transaction is conducted using TLS [RFC5246], the conference server can authenticate its identity, either as a domain name or as an IP address, to the conferencing client by presenting a certificate containing that identifier as a subjectAltName (i.e., as an iPAddress or dNSName, respectively). Any implementation of CCMP MUST be capable of being transacted over TLS so that the client can

request the above authentication. Note that, in order for the presented certificate to be valid at the client, the client MUST be able to validate the certificate following the procedures in [RFC2818] in the case of HTTP as a transport. In particular, the validation path of the certificate must end in one of the client's trust anchors, even if that trust anchor is the conference server certificate itself. If the client has external information as to the expected identity or credentials of the proper conference server, the authentication checks described above MAY be omitted.

10.2. User Authentication and Authorization

Many policy authorization decisions are based on the identity of the user or the role that a user may have. The conference server MUST implement mechanisms for authentication of users to validate their identity. There are several ways that a user might authenticate its identity to the system. For users joining a conference using one of the call signaling protocols, the user authentication mechanisms for the specific protocol can be used. For example, in the case of a user joining the conference using SIP signaling, the user authentication as defined in [RFC3261] MUST be used. For the case of users joining the conference using CCMP, the CCMP Request messages provide a subject field that contains a username and password, which can be used for authentication. Since the CCMP messages are RECOMMENDED to be carried over TLS, this information can be sent securely.

The XCON framework [RFC5239] provides an overview of other authorization mechanisms. In the cases where a user is authorized via multiple mechanisms, it is RECOMMENDED that the conference server associate the authorization of the CCMP interface with other authorization mechanisms; for example, Public Switched Telephone Network (PSTN) users that join with a PIN and control the conference using CCMP. When a conference server presents the identity of authorized users, it MAY provide information about the way the identity was proven or verified by the system. A conference server can also allow a completely unauthenticated user into the system -- this information SHOULD also be communicated to interested parties.

Once a user is authenticated and authorized through the various mechanisms available on the conference server, the conference server MUST allocate a conference user identifier (XCON-USERID) and SHOULD associate the XCON-USERID with any signaling specific user identifiers that were used for authentication and authorization. This XCON-USERID can be provided to a specific user through the conference notification interface and MUST be provided to users that interact with the conferencing system using CCMP (i.e., in the appropriate CCMP response messages). The XCON-USERIDs for each user/

participant in the conference are contained in the 'entity' attribute in the <user> element in the conference object. The XCON-USERID is REQUIRED for any subsequent operations by the user on the conference object and is carried in the confUserID parameter in the CCMP requests and responses.

Note that the policy management of an XCON-compliant conferencing system is out of the scope of this document, as well as of the XCON working group (WG). However, the specification of a policy management framework is realizable with the overall XCON architecture, in particular with regard to a Role-Based Access Control (RBAC) approach. In RBAC, the following elements are identified: (i) Users; (ii) Roles; (iii) Objects; (iv) Operations; (v) Permissions. For all of the above elements, a direct mapping exists onto the main XCON entities. As an example, RBAC objects map onto XCON data model objects and RBAC operations map onto CCMP operations.

Future documents can define an RBAC framework for XCON, by first focusing on the definition of roles and then specifying the needed permission policy sets and role policy sets (used to associate policy permission sets with specific roles). With these policies in place, access to a conference object compliant with the XCON data model can be appropriately controlled. As far as assigning users to roles, the Users in the RBAC model relate directly to the <users> element in the conference object. The <users> element is comprised of <user> elements representing a specific user in the conferencing system.

Each <user> element contains an 'entity' attribute with the XCON-USERID and a <role> element. Thus, each authorized user (as represented by an XCON-USERID) can be associated with a <role> element.

10.3. Security and Privacy of Identity

An overview of the required privacy and anonymity for users of a conferencing system are provided in the XCON framework [RFC5239]. The security of the identity in the form of the XCON-USERID is provided in CCMP through the use of TLS.

The conference server SHOULD support the mechanism to ensure the privacy of the XCON-USERID. The conferencing client indicates the desired level of privacy by manipulation of the <provide-anonymity> element defined in the XCON data model [RFC6501]. The <provide-anonymity> element controls the degree to which a user reveals their identity. The following summarizes the values for the <provide-anonymity> element that the client includes in their requests:

"hidden": Ensures that other participants are not aware that there is an additional participant (i.e., the user issuing the request) in the conference. This could be used in cases of users that are authorized with a special role in a conference (e.g., a supervisor in a call center environment).

"anonymous": Ensures that other participants are aware that there is another participant (i.e., the user issuing the request); however, the other participants are not provided information as to the identity of the user.

"semi-private": Ensures that the user's identity is only to be revealed to other participants or users that have a higher-level authorization (e.g., a conferencing system can be configured such that a human administrator can see all users).

If the client desires privacy, the conferencing client SHOULD include the <provide-anonymity> element in the <confInfo> parameter in a CCMP confRequest message with an <operation> parameter of "update" or "create" or in the <userInfo> parameter in a CCMP userRequest message with an <operation> parameter of "update" or "create". If the <provide-anonymity> element is not included in the conference object, then other users can see the participant's identity. Participants are made aware of other participants that are "anonymous" or "semi-private" when they perform subsequent operations on the conference object or retrieve the conference object or when they receive subsequent notifications.

Note that independent of the level of anonymity requested by the user, the identity of the user is always known by the conferencing system as that is required to perform the necessary authorization as described in Section 10.2. The degree to which human administrators can see the information can be controlled using policies (e.g., some information in the data model can be hidden from human administrators).

10.4. Mitigating DoS Attacks

[RFC4732] provides an overview of possible DoS attacks. In order to minimize the potential for DoS attacks, it is RECOMMENDED that conferencing systems require user authentication and authorization for any client participating in a conference. This can be accomplished through the use of the mechanisms described in Section 10.2, as well as by using the security mechanisms associated with the specific signaling (e.g., Session Initiation Protocol Secure (SIPS)) and media protocols (e.g., Secure Realtime Transport Protocol (SRTP)). In addition, Section 4.4 describes the use of a timer mechanism to alleviate the situation whereby CCMP messages pend

indefinitely, thus increasing the potential that pending requests continue to increase when is a server is receiving more requests than it can process.

11. XML Schema

This section gives the XML schema definition [W3C.REC-xmlschema-1-20041028] [W3C.REC-xmlschema-2-20041028] of the "application/ccmp+xml" format. This is presented as a formal definition of the "application/ccmp+xml" format. A new XML namespace, a new XML schema, and the MIME type for this schema are registered with IANA as described in Section 12. Note that this XML Schema Definition is not intended to be used with on-the-fly validation of the presence XML document. Whitespaces are included in the schema to conform to the line length restrictions of the RFC format without having a negative impact on the readability of the document. Any conforming processor should remove leading and trailing white spaces.

```
<?xml version="1.0" encoding="utf-8"?>

   <xs:schema
      targetNamespace="urn:ietf:params:xml:ns:xcon-ccmp"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="urn:ietf:params:xml:ns:xcon-ccmp"
      xmlns:tns="urn:ietf:params:xml:ns:xcon-ccmp"
      xmlns:dm="urn:ietf:params:xml:ns:xcon-conference-info"
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">


      <xs:import namespace="urn:ietf:params:xml:ns:xcon-conference-info"
                 schemaLocation="DataModel.xsd"/>
      <xs:import namespace="urn:ietf:params:xml:ns:conference-info"
                 schemaLocation="rfc4575.xsd"/>

      <xs:element name="ccmpRequest" type="ccmp-request-type" />
      <xs:element name="ccmpResponse" type="ccmp-response-type" />

<!-- CCMP request definition -->

   <xs:complexType name="ccmp-request-type">
       <xs:sequence>
           <xs:element name="ccmpRequest"
                       type="ccmp-request-message-type" />
       </xs:sequence>
   </xs:complexType>
```

```
   <!-- ccmp-request-message-type -->

   <xs:complexType abstract="true"
                   name="ccmp-request-message-type">
       <xs:sequence>
           <xs:element name="subject" type="subject-type"
                       minOccurs="0" maxOccurs="1" />
           <xs:element name="confUserID" type="xs:string"
                       minOccurs="0" maxOccurs="1" />
           <xs:element name="confObjID" type="xs:string"
                       minOccurs="0" maxOccurs="1" />
           <xs:element name="operation" type="operationType"
                       minOccurs="0" maxOccurs="1" />
           <xs:element name="conference-password" type="xs:string"
                       minOccurs="0" maxOccurs="1" />
          <xs:any namespace="##other" processContents="lax"
                  minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

<!-- CCMP response definition -->

   <xs:complexType name="ccmp-response-type">
       <xs:sequence>
           <xs:element name="ccmpResponse"
                       type="ccmp-response-message-type" />
       </xs:sequence>
   </xs:complexType>

   <!-- ccmp-response-message-type -->

   <xs:complexType abstract="true" name="ccmp-response-message-type">
       <xs:sequence>
           <xs:element name="confUserID" type="xs:string"
                       minOccurs="1" maxOccurs="1" />
           <xs:element name="confObjID" type="xs:string"
                       minOccurs="0" maxOccurs="1" />
           <xs:element name="operation" type="operationType"
                       minOccurs="0"
                       maxOccurs="1" />
           <xs:element name="response-code"
                       type="response-codeType"
                       minOccurs="1" maxOccurs="1" />
           <xs:element name="response-string" type="xs:string"
                       minOccurs="0" maxOccurs="1" />
           <xs:element name="version" type="xs:positiveInteger"
                       minOccurs="0" maxOccurs="1" />
```

```
            <xs:any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

<!-- CCMP REQUESTS -->

    <!-- blueprintsRequest -->

    <xs:complexType name="ccmp-blueprints-request-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-request-message-type">
                <xs:sequence>
                    <xs:element ref="blueprintsRequest" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <!-- blueprintsRequestType -->

    <xs:element name="blueprintsRequest" type="blueprintsRequestType"/>

    <xs:complexType name="blueprintsRequestType">
        <xs:sequence>
            <xs:element name="xpathFilter" type="xs:string"
                        minOccurs="0"/>
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!--  blueprintRequest -->

    <xs:complexType name="ccmp-blueprint-request-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-request-message-type">
                <xs:sequence>
                    <xs:element ref="blueprintRequest" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```

```
   <!-- blueprintRequestType -->

   <xs:element name="blueprintRequest" type="blueprintRequestType" />

   <xs:complexType name="blueprintRequestType">
       <xs:sequence>
           <xs:element name="blueprintInfo"
                       type="info:conference-type" minOccurs="0"/>
           <xs:any namespace="##other" processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- confsRequest -->

   <xs:complexType name="ccmp-confs-request-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-request-message-type">
               <xs:sequence>
                   <xs:element ref="confsRequest" />
               </xs:sequence>
           </xs:extension>
       </xs:complexContent>
   </xs:complexType>

   <!-- confsRequestType -->

   <xs:element name="confsRequest" type="confsRequestType" />
   <xs:complexType name="confsRequestType">
       <xs:sequence>
           <xs:element name="xpathFilter" type="xs:string"
                       minOccurs="0"/>
           <xs:any namespace="##other" processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- confRequest -->

   <xs:complexType name="ccmp-conf-request-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-request-message-type">
               <xs:sequence>
                   <xs:element ref="confRequest" />
               </xs:sequence>
           </xs:extension>
```

```
      </xs:complexContent>
    </xs:complexType>

   <!-- confRequestType -->

   <xs:element name="confRequest" type="confRequestType" />

   <xs:complexType name="confRequestType">
       <xs:sequence>
           <xs:element name="confInfo" type="info:conference-type"
                       minOccurs="0"/>
           <xs:any namespace="##other" processContents="lax"
                  minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- usersRequest -->

   <xs:complexType name="ccmp-users-request-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-request-message-type">
               <xs:sequence>
                   <xs:element ref="usersRequest" />
               </xs:sequence>
           </xs:extension>
       </xs:complexContent>
   </xs:complexType>

   <!-- usersRequestType -->

   <xs:element name="usersRequest" type="usersRequestType" />

   <xs:complexType name="usersRequestType">
       <xs:sequence>
           <xs:element name="usersInfo" type="info:users-type"
                       minOccurs="0" />
           <xs:any namespace="##other" processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

   <!-- userRequest -->

   <xs:complexType name="ccmp-user-request-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-request-message-type">
```

```
            <xs:sequence>
                <xs:element ref="userRequest" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- userRequestType -->

<xs:element name="userRequest" type="userRequestType" />

<xs:complexType name="userRequestType">
    <xs:sequence>
        <xs:element name="userInfo" type="info:user-type"
                    minOccurs="0" />
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<!-- sidebarsByValRequest -->

<xs:complexType name="ccmp-sidebarsByVal-request-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type">
            <xs:sequence>
                <xs:element ref="sidebarsByValRequest" />
             </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

 <!-- sidebarsByValRequestType -->

 <xs:element name="sidebarsByValRequest"
             type="sidebarsByValRequestType" />

 <xs:complexType name="sidebarsByValRequestType">
     <xs:sequence>
         <xs:element name="xpathFilter"
                     type="xs:string" minOccurs="0"/>
         <xs:any namespace="##other" processContents="lax"
                 minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
         <xs:anyAttribute namespace="##any" processContents="lax"/>
 </xs:complexType>
```

```
    <!-- sidebarsByRefRequest -->

    <xs:complexType name="ccmp-sidebarsByRef-request-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-request-message-type">
                <xs:sequence>
                    <xs:element ref="sidebarsByRefRequest" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

   <!-- sidebarsByRefRequestType -->

   <xs:element name="sidebarsByRefRequest"
               type="sidebarsByRefRequestType" />

   <xs:complexType name="sidebarsByRefRequestType">
       <xs:sequence>
           <xs:element name="xpathFilter" type="xs:string"
                       minOccurs="0"/>
           <xs:any namespace="##other" processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- sidebarByValRequest -->

   <xs:complexType name="ccmp-sidebarByVal-request-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-request-message-type">
               <xs:sequence>
                   <xs:element ref="sidebarByValRequest" />
               </xs:sequence>
            </xs:extension>
       </xs:complexContent>
   </xs:complexType>

   <!-- sidebarByValRequestType -->

   <xs:element name="sidebarByValRequest"
               type="sidebarByValRequestType"/>

   <xs:complexType name="sidebarByValRequestType">
       <xs:sequence>
           <xs:element name="sidebarByValInfo"
                       type="info:conference-type" minOccurs="0"/>
```

```
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
<xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<!-- sidebarByRefRequest -->

<xs:complexType name="ccmp-sidebarByRef-request-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type">
            <xs:sequence>
                <xs:element ref="sidebarByRefRequest" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- sidebarByRefRequestType -->

<xs:element name="sidebarByRefRequest"
            type="sidebarByRefRequestType" />

<xs:complexType name="sidebarByRefRequestType">
    <xs:sequence>
        <xs:element name="sidebarByRefInfo"
                    type="info:conference-type" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<!-- extendedRequest -->

<xs:complexType name="ccmp-extended-request-message-type">
   <xs:complexContent>
       <xs:extension base="tns:ccmp-request-message-type">
          <xs:sequence>
                      <xs:element ref="extendedRequest"/>
          </xs:sequence>
       </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

```
   <!-- extendedRequestType -->

   <xs:element name="extendedRequest" type="extendedRequestType"/>

   <xs:complexType name="extendedRequestType">
     <xs:sequence>
        <xs:element name="extensionName"
                    type="xs:string" minOccurs="1"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
                          maxOccurs="unbounded" />
    </xs:sequence>
   </xs:complexType>

   <!-- optionsRequest -->

        <xs:complexType name="ccmp-options-request-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-request-message-type">
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>


<!-- CCMP RESPONSES -->

   <!-- blueprintsResponse -->

   <xs:complexType name="ccmp-blueprints-response-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-response-message-type">
               <xs:sequence>
                   <xs:element ref="blueprintsResponse" />
               </xs:sequence>
           </xs:extension>
       </xs:complexContent>
   </xs:complexType>

   <!-- blueprintsResponseType -->

   <xs:element name="blueprintsResponse" type="blueprintsResponseType"/>

   <xs:complexType name="blueprintsResponseType">
       <xs:sequence>
           <xs:element name="blueprintsInfo" type="info:uris-type"
                       minOccurs="0"/>
           <xs:any namespace="##other" processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
```

```
      <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>

   <!-- blueprintResponse -->

   <xs:complexType name="ccmp-blueprint-response-message-type">
       <xs:complexContent>
           <xs:extension base="tns:ccmp-response-message-type">
               <xs:sequence>
                   <xs:element ref="blueprintResponse" />
               </xs:sequence>
           </xs:extension>
       </xs:complexContent>
    </xs:complexType>

    <!-- blueprintResponseType -->

    <xs:element name="blueprintResponse" type="blueprintResponseType"/>

    <xs:complexType name="blueprintResponseType">
        <xs:sequence>
          <xs:element name="blueprintInfo" type="info:conference-type"
                      minOccurs="0"/>
          <xs:any namespace="##other" processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- confsResponse -->

    <xs:complexType name="ccmp-confs-response-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-response-message-type">
                <xs:sequence>
                    <xs:element ref="confsResponse" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <!-- confsResponseType -->

    <xs:element name="confsResponse" type="confsResponseType" />

    <xs:complexType name="confsResponseType">
        <xs:sequence>
            <xs:element name="confsInfo" type="info:uris-type"
```

```
                    minOccurs="0"/>
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- confResponse -->

    <xs:complexType name="ccmp-conf-response-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-response-message-type">
                <xs:sequence>
                    <xs:element ref="confResponse"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <!-- confResponseType -->

    <xs:element name="confResponse" type="confResponseType" />

    <xs:complexType name="confResponseType">
        <xs:sequence>
            <xs:element name="confInfo" type="info:conference-type"
                    minOccurs="0"/>
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- usersResponse -->

    <xs:complexType name="ccmp-users-response-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-response-message-type">
                <xs:sequence>
                    <xs:element ref="usersResponse" />
                </xs:sequence>
            </xs:extension>
         </xs:complexContent>
    </xs:complexType>
```

```
    <!-- usersResponseType -->

    <xs:element name="usersResponse" type="usersResponseType" />

    <xs:complexType name="usersResponseType">
        <xs:sequence>
            <xs:element name="usersInfo" type="info:users-type"
                        minOccurs="0"/>
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- userResponse -->

    <xs:complexType name="ccmp-user-response-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-response-message-type">
                <xs:sequence>
                    <xs:element ref="userResponse" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <!-- userResponseType -->

    <xs:element name="userResponse" type="userResponseType" />

    <xs:complexType name="userResponseType">
        <xs:sequence>
            <xs:element name="userInfo" type="info:user-type"
                        minOccurs="0"/>
            <xs:any namespace="##other" processContents="lax"
                  minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- sidebarsByValResponse -->

    <xs:complexType name="ccmp-sidebarsByVal-response-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-response-message-type">
                <xs:sequence>
                    <xs:element ref="sidebarsByValResponse" />
                </xs:sequence>
```

```
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- sidebarsByValResponseType -->

<xs:element name="sidebarsByValResponse"
            type="sidebarsByValResponseType" />

<xs:complexType name="sidebarsByValResponseType">
    <xs:sequence>
        <xs:element name="sidebarsByValInfo"
                    type="info:sidebars-by-val-type" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<!-- sidebarsByRefResponse -->

<xs:complexType name="ccmp-sidebarsByRef-response-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-response-message-type">
            <xs:sequence>
                <xs:element ref="sidebarsByRefResponse" />
            </xs:sequence>
        </xs:extension>
   </xs:complexContent>
</xs:complexType>

<!-- sidebarsByRefResponseType -->

<xs:element name="sidebarsByRefResponse"
            type="sidebarsByRefResponseType" />

<xs:complexType name="sidebarsByRefResponseType">
    <xs:sequence>
        <xs:element name="sidebarsByRefInfo" type="info:uris-type"
                    minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

```
    <!-- sidebarByValResponse -->

    <xs:complexType name="ccmp-sidebarByVal-response-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-response-message-type">
                <xs:sequence>
                    <xs:element ref="sidebarByValResponse" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <!-- sidebarByValResponseType -->

    <xs:element name="sidebarByValResponse"
                type="sidebarByValResponseType" />

    <xs:complexType name="sidebarByValResponseType">
        <xs:sequence>
            <xs:element name="sidebarByValInfo"
                        type="info:conference-type" minOccurs="0"/>
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- sidebarByRefResponse -->

    <xs:complexType name="ccmp-sidebarByRef-response-message-type">
        <xs:complexContent>
            <xs:extension base="tns:ccmp-response-message-type">
                <xs:sequence>
                    <xs:element ref="sidebarByRefResponse" />
                 </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <!-- sidebarByRefResponseType -->

    <xs:element name="sidebarByRefResponse"
                type="sidebarByRefResponseType" />

    <xs:complexType name="sidebarByRefResponseType">
        <xs:sequence>
            <xs:element name="sidebarByRefInfo"
                        type="info:conference-type" minOccurs="0"/>
```

```
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<!-- extendedResponse -->

<xs:complexType name="ccmp-extended-response-message-type">
   <xs:complexContent>
       <xs:extension base="tns:ccmp-response-message-type">
           <xs:sequence>
                              <xs:element ref="extendedResponse"/>
           </xs:sequence>
       </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- extendedResponseType -->

<xs:element name="extendedResponse" type="extendedResponseType"/>

<xs:complexType name="extendedResponseType">
    <xs:sequence>
            <xs:element name="extensionName"
                        type="xs:string" minOccurs="1"/>
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0"
                maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>

<!-- optionsResponse -->

    <xs:complexType name="ccmp-options-response-message-type">
    <xs:complexContent>
        <xs:extension base="tns:ccmp-response-message-type">
            <xs:sequence>
                <xs:element ref="optionsResponse"/>
             </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- optionsResponseType -->

<xs:element name="optionsResponse"
            type="optionsResponseType" />
```

```
    <xs:complexType name="optionsResponseType">
        <xs:sequence>
            <xs:element name="options"
                        type="options-type" minOccurs="0"/>
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

<!-- CCMP ELEMENT TYPES -->

    <!-- response-codeType-->

    <xs:simpleType name="response-codeType">
         <xs:restriction base="xs:positiveInteger">
                 <xs:pattern value="[0-9][0-9][0-9]" />
         </xs:restriction>
    </xs:simpleType>

    <!-- operationType -->

    <xs:simpleType name="operationType">
      <xs:restriction base="xs:token">
        <xs:enumeration value="retrieve"/>
        <xs:enumeration value="create"/>
        <xs:enumeration value="update"/>
        <xs:enumeration value="delete"/>
      </xs:restriction>
    </xs:simpleType>

   <!-- subject-type -->

   <xs:complexType name="subject-type">
       <xs:sequence>
           <xs:element name="username" type="xs:string"
                       minOccurs="0" maxOccurs="1" />
           <xs:element name="password" type="xs:string"
                       minOccurs="0" maxOccurs="1" />
           <xs:any namespace="##other" processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>
```

```
   <!-- options-type -->

    <xs:complexType name="options-type">
      <xs:sequence>
        <xs:element name="standard-message-list"
                    type="standard-message-list-type"
                    minOccurs="1"/>
        <xs:element name="extended-message-list"
                    type="extended-message-list-type"
                    minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- standard-message-list-type -->

    <xs:complexType name="standard-message-list-type">
      <xs:sequence>
        <xs:element name="standard-message"
                    type="standard-message-type"
                    minOccurs="1" maxOccurs="10"/>
        <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- standard-message-type -->

    <xs:complexType name="standard-message-type">
      <xs:sequence>
        <xs:element name="name"
                    type="standard-message-name-type"
                    minOccurs="1"/>
        <xs:element name="operations"
                    type="operations-type"
                    minOccurs="0"/>
        <xs:element name="schema-def" type="xs:string" minOccurs="0"/>
        <xs:element name="description" type="xs:string" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>
```

```
    <!-- standard-message-name-type -->

    <xs:simpleType name="standard-message-name-type">
      <xs:restriction base="xs:token">
        <xs:enumeration value="confsRequest"/>
        <xs:enumeration value="confRequest"/>
        <xs:enumeration value="blueprintsRequest"/>
        <xs:enumeration value="blueprintRequest"/>
        <xs:enumeration value="usersRequest"/>
        <xs:enumeration value="userRequest"/>
        <xs:enumeration value="sidebarsByValRequest"/>
        <xs:enumeration value="sidebarByValRequest"/>
        <xs:enumeration value="sidebarsByRefRequest"/>
        <xs:enumeration value="sidebarByRefRequest"/>
      </xs:restriction>
    </xs:simpleType>

    <!-- operations-type -->

    <xs:complexType name="operations-type">
      <xs:sequence>
        <xs:element name="operation" type="operationType"
                    minOccurs="1" maxOccurs="4"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- extended-message-list-type -->

    <xs:complexType name="extended-message-list-type">
      <xs:sequence>
        <xs:element name="extended-message"
                    type="extended-message-type"
                    minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <!-- extended-message-type -->

    <xs:complexType name="extended-message-type">
      <xs:sequence>
        <xs:element name="name" type="xs:string" />
        <xs:element name="operations"
                    type="operations-type"
                    minOccurs="0"/>
```

```
        <xs:element name="schema-def" type="xs:string" />
        <xs:element name="description"
                    type="xs:string"
                    minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

 </xs:schema>
```

Figure 30: CCMP XML Schema

12. IANA Considerations

This document registers a new XML namespace, a new XML schema, and the MIME type for the schema. This document also registers the "XCON" Application Service tag and the "CCMP" Application Protocol tag and defines registries for the CCMP operation types and response codes.

12.1. URN Sub-Namespace Registration

This section registers a new XML namespace, "urn:ietf:params:xml:ns:xcon-ccmp".

URI: urn:ietf:params:xml:ns:xcon-ccmp

Registrant Contact: IETF XCON working group (xcon@ietf.org), Mary Barnes (mary.ietf.barnes@gmail.com).

XML:

```
BEGIN
  <?xml version="1.0"?>
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
    <head>
      <title>CCMP Messages</title>
    </head>
    <body>
      <h1>Namespace for CCMP Messages</h1>
      <h2>urn:ietf:params:xml:ns:xcon-ccmp</h2>
      <p>See <a href="http://www.rfc-editor.org/rfc/rfc6503.txt">
         RFC 6503</a>.</p>
    </body>
  </html>
END
```

12.2. XML Schema Registration

This section registers an XML schema per the guidelines in [RFC3688].

URI: urn:ietf:params:xml:schema:xcon-ccmp

Registrant Contact: IETF XCON working group (xcon@ietf.org), Mary Barnes (mary.ietf.barnes@gmail.com).

Schema: The XML for this schema can be found as the entirety of Section 11 of this document.

12.3. MIME Media Type Registration for 'application/ccmp+xml'

This section registers the "application/ccmp+xml" MIME type.

To: ietf-types@iana.org

Subject: Registration of MIME media type application/ccmp+xml

MIME media type name: application

MIME subtype name: ccmp+xml

Required parameters: (none)

Optional parameters: charset
Same as the charset parameter of "application/xml" as specified in [RFC3023], Section 3.2.

Encoding considerations: Same as the encoding considerations of "application/xml" as specified in [RFC3023], Section 3.2.

Security considerations: This content type is designed to carry protocol data related to conference control. Some of the data could be considered private. This media type does not provide any protection and thus other mechanisms such as those described in Section 10 are required to protect the data. This media type does not contain executable content.

Interoperability considerations: None.

Published specification: RFC 6503.

Applications that use this media type: Centralized Conferencing control clients and servers.

Additional Information: Magic Number(s): (none)
File extension(s): .ccmp
Macintosh File Type Code(s): TEXT

Person & email address to contact for further information: Mary Barnes <mary.ietf.barnes@gmail.com>

Intended usage: LIMITED USE

Author/Change controller: The IETF

Other information: This media type is a specialization of application/xml [RFC3023], and many of the considerations described there also apply to application/ccmp+xml.

12.4. DNS Registrations

Section 12.4.1 defines an Application Service tag of "XCON", which is used to identify the centralized conferencing (XCON) server for a particular domain. The Application Protocol tag "CCMP", defined in Section 12.4.2, is used to identify an XCON server that understands CCMP.

12.4.1. Registration of a Conference Server Application Service Tag

This section registers a new S-NAPTR/U-NAPTR Application Service tag for XCON, as mandated by [RFC3958].

Application Service Tag: XCON

Intended usage: Identifies a server that supports centralized conferencing.

Defining publication: RFC 6503

Contact information: The authors of this document

Author/Change controller: The IESG

12.4.2. Registration of a Conference Server Application Protocol Tag for CCMP

This section registers a new S-NAPTR/U-NAPTR Application Protocol tag for CCMP, as mandated by [RFC3958].

Application Service Tag: CCMP

Intended Usage: Identifies the Centralized Conferencing (XCON) Manipulation Protocol.

Applicable Service Tag(s): XCON

Terminal NAPTR Record Type(s): U

Defining Publication: RFC 6503

Contact Information: The authors of this document

Author/Change Controller: The IESG

12.5. CCMP Protocol Registry

The IANA has created a new registry for CCMP: http://www.iana.org/assignments/ccmp-parameters. The document creates initial sub-registries for CCMP operation types and response codes.

12.5.1. CCMP Message Types

The following summarizes the registry for CCMP messages:

Related Registry: CCMP Message Types Registry

Defining RFC: RFC 6503.

Registration/Assignment Procedures: Following the policies outlined in [RFC5226], the IANA policy for assigning new values for the CCMP message types for CCMP is Specification Required.

Registrant Contact: IETF XCON working group (xcon@ietf.org), Mary Barnes (mary.ietf.barnes@gmail.com).

This specification establishes the Message sub-registry under http://www.iana.org/assignments/ccmp-messages. The initial Message table is populated using the CCMP messages described in Section 4.1 and defined in the XML schema in Section 11.

Message	Description	Reference
optionsRequest	Used by a conferencing client to query a conference server for its capabilities, in terms of supported messages.	[RFC6503]
optionsResponse	Returns a list of CCMP messages supported by the specific conference server.	[RFC6503]
blueprintsRequest	Used by a conferencing client to query a conference server for its capabilities, in terms of available conference blueprints.	[RFC6503]
blueprintsResponse	Returns a list of blueprints supported by the specific conference server.	[RFC6503]
blueprintRequest	Sent to retrieve the conference object associated with a specific blueprint.	[RFC6503]
blueprintResponse	Returns the conference object associated with a specific blueprint.	[RFC6503]
confsRequest	Used by a conferencing client to query a conference server for its scheduled/active conferences.	[RFC6503]

confsResponse	Returns the list of the currently activated/scheduled conferences at the server.	[RFC6503]
confRequest	Used to create a conference object and/or to request an operation on the conference object as a whole.	[RFC6503]
confResponse	Indicates the result of the operation on the conference object as a whole.	[RFC6503]
userRequest	Used to request an operation on the <user> element in the conference object.	[RFC6503]
userResponse	Indicates the result of the requested operation on the <user> element in the conference object.	[RFC6503]
usersRequest	Used to manipulate the <users> element in the conference object, including parameters such as the <allowed-users-list>, <join-handling>, etc.	[RFC6503]
usersResponse	Indicates the result of the request to manipulate the <users> element in the conference object.	[RFC6503]
sidebarsByValRequest	Used to retrieve the <sidebars-by-val> element of the target conference object.	[RFC6503]
sidebarsByValResponse	Returns the list of the sidebar-by-val conferences within the target conference object.	[RFC6503]
sidebarsByRefRequest	Used to retrieve the <sidebars-by-ref> element of the target conference object.	[RFC6503]
sidebarsByRefResponse	Returns the list of the sidebar-by-ref conferences associated with the target conference object.	[RFC6503]
sidebarByValRequest	Used to request an operation on a sidebar-by-val conference.	[RFC6503]
sidebarByValResponse	Indicates the result of the request to manipulate a sidebar-by-val conference.	[RFC6503]

sidebarByRefRequest	Used to request an operation on a sideber-by-ref conference.	[RFC6503]
sidebarByRefResponse	Indicates the result of the request to manipulate a sidebar-by-ref conference.	[RFC6503]

12.5.2. CCMP Response Codes

The following summarizes the requested registry for CCMP response codes:

Related Registry: CCMP Response Code Registry

Defining RFC: RFC 6503.

Registration/Assignment Procedures: Following the policies outlined in [RFC5226], the IANA policy for assigning new values for the Response codes for CCMP shall be Specification Required.

Registrant Contact: IETF XCON working group (xcon@ietf.org), Mary Barnes (mary.ietf.barnes@gmail.com).

This specification establishes the Response-code sub-registry under http://www.iana.org/assignments/ccmp-parameters. The initial Response-code table is populated using the Response codes defined in Section 5.4 as follows:

Number	Default Response String	Description	Reference
200	Success	The request was successfully processed.	[RFC6503]
400	Bad Request	The request was badly formed in some fashion.	[RFC6503]
401	Unauthorized	The user was not authorized for the specific operation on the conference object.	[RFC6503]
403	Forbidden	The specific operation is not valid for the target conference object.	[RFC6503]
404	Object Not Found	The specific conference object was not found.	[RFC6503]

409	Conflict	A requested operation cannot be successfully completed by the server. For example, the modification of an object cannot be applied because the client version of the object is obsolete and the requested modifications collide with the up-to-date state of the object stored at the server.	[RFC6503]
420	User Not Found	The user who is the target of the requested operation is unknown.	[RFC6503]
421	Invalid confUserID	The <confUserID> parameter of the sender in the request is invalid.	[RFC6503]
422	Invalid Conference Password	A request to access/manipulate a password-protected conference object contained an invalid <conference-password> parameter.	[RFC6503]
423	Conference Password Required	A request to access/manipulate a password-protected conference object did not contain a <conference-password> parameter.	[RFC6503]
424	Authentication Required	The server wants to authenticate the request through the <subject> parameter but the parameter is not provided in the request.	[RFC6503]
425	Forbidden Delete Parent	The conferencing system cannot delete the specific conference object because it is a parent for another conference object.	[RFC6503]
426	Forbidden Change Protected	The target conference object cannot be changed (e.g., due to policies, roles or privileges).	[RFC6503]
427	Invalid Domain Name	The domain name in an AUTO_GENERATE_X instance in the conference object is not within the conference server's domain of responsibility.	[RFC6503]

500	Server Internal Error	The conference server experienced some sort of internal error.	[RFC6503]
501	Not Implemented	The specific operation is not implemented on the conferencing system.	[RFC6503]
510	Request Timeout	The request could not be processed within a reasonable time (as specified by the conferencing system).	[RFC6503]
511	Resources Not Available	The conference server cannot execute a command because of resource issues, e.g., it cannot create a conference because the system has reached its limits on the number of conferences.	[RFC6503]

13. Acknowledgments

The authors appreciate the feedback provided by Dave Morgan, Pierre Tane, Lorenzo Miniero, Tobia Castaldi, Theo Zourzouvillys, Sean Duddy, Oscar Novo, Richard Barnes, Simo Veikkolainen, Keith Drage, Peter Reissner, Tony Lindstrom, Stephen Kent (secdir review), Brian Carpenter (genart review), and Mykyta Yevstifeyev (IANA considerations). Special thanks go to Roberta Presta for her invaluable contribution to this document. Roberta has worked on the specification of CCMP at the University of Napoli for the preparation of her Master thesis. She has also implemented the CCMP prototype used for the trials and from which the dumps provided in Section 6 have been extracted.

14. References

14.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC5239] Barnes, M., Boulton, C., and O. Levin, "A Framework for Centralized Conferencing", RFC 5239, June 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[RFC6501] Novo, O., Camarillo, G., Morgan, D., and J. Urpalainen, "Conference Information Data Model for Centralized Conferencing (XCON)", RFC 6501, March 2012.

[W3C.REC-xmlschema-1-20041028]
Beech, D., Thompson, H., Mendelsohn, N., and M. Maloney, "XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[W3C.REC-xmlschema-2-20041028]
Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

14.2. Informative References

[REST] Fielding, "Architectural Styles and the Design of Network-based Software Architectures", 2000.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3958] Daigle, L. and A. Newton, "Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)", RFC 3958, January 2005.

[RFC4582] Camarillo, G., Ott, J., and K. Drage, "The Binary Floor Control Protocol (BFCP)", RFC 4582, November 2006.

[RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC6502] Camarillo, G., Srinivasan, S., Even, R., and J. Urpalainen, "Conference Event Package Data Format Extension for Centralized Conferencing (XCON)", RFC 6502, March 2012.

[W3C.REC-soap12-part1-20070427]
Nielsen, H., Mendelsohn, N., Moreau, J., Gudgin, M., Hadley, M., Lafon, Y., and A. Karmarkar, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)", World Wide Web Consortium Recommendation REC-soap12-part1-20070427, April 2007, <http://www.w3.org/TR/2007/REC-soap12-part1-20070427>.

[W3C.REC-soap12-part2-20070427]
Moreau, J., Gudgin, M., Karmarkar, A., Mendelsohn, N., Hadley, M., Lafon, Y., and H. Nielsen, "SOAP Version 1.2 Part 2: Adjuncts (Second Edition)", World Wide Web Consortium Recommendation REC-soap12-part2-20070427, April 2007, <http://www.w3.org/TR/2007/REC-soap12-part2-20070427>.

Appendix A. Evaluation of Other Protocol Models and Transports Considered for CCMP

This section provides some background as to the selection of HTTP as the transport for the CCMP requests/responses. In addition to HTTP, the operations on the objects can be implemented in at least two different ways, namely as remote procedure calls -- using SOAP as described in Appendix A.1 and by defining resources following a RESTful architecture Appendix A.2.

In both the SOAP and RESTFUL approaches, servers will have to recreate their internal state representation of the object with each update request, checking parameters and triggering function invocations. In the SOAP approach, it would be possible to describe a separate operation for each atomic element, but that would greatly increase the complexity of the protocol. A coarser-grained approach to CCMP does require that the server process XML elements in updates that have not changed and that there can be multiple changes in one update. For CCMP, the resource (REST) model might appear more attractive, since the conference operations fit the CRUD approach.

However, neither of these approaches were considered ideal. SOAP was considered to bring additional overhead. It is quite awkward to apply a RESTful approach since CCMP requires a more complex request/response protocol in order to maintain the data both in the server and at the client. This doesn't map very elegantly to the basic request/response model, whereby a response typically indicates whether the request was successful or not, rather than providing additional data to maintain the synchronization between the client and server data. In addition, the CCMP clients may also receive the data in notifications. While the notification method or protocol used by some conferencing clients can be independent of CCMP, the same data in the server is used for both CCMP and notifications - this requires a server application above the transport layer (e.g., HTTP) for maintaining the data, which in the CCMP model is transparent to the transport protocol.

Thus, the solution for CCMP defined in this document is viewed as a good compromise amongst the most notable past candidates and is referred to as "HTTP single-verb transport plus CCMP body". With this approach, CCMP is able to take advantage of existing HTTP functionality. As with SOAP, CCMP uses a "single HTTP verb" for transport (i.e., a single transaction type for each request/response pair); this allows decoupling CCMP messages from HTTP messages. Similarly, as with any RESTful approach, CCMP messages are inserted directly in the body of HTTP messages, thus avoiding any unnecessary processing and communication burden associated with further intermediaries. With this approach, no modification to the CCMP

messages/operations is required to use a different transport protocol.

A.1. Using SOAP for CCMP

A remote procedure call (RPC) mechanism for CCMP could use SOAP (Simple Object Access Protocol [W3C.REC-soap12-part1-20070427] [W3C.REC-soap12-part2-20070427]), where conferences and the other objects are modeled as services with associated operations. Conferences and other objects are selected by their own local identifiers, such as email-like names for users. This approach has the advantage that it can easily define atomic operations that have well-defined error conditions.

All SOAP operations would use a single HTTP verb. While the RESTful approach requires the use of a URI for each object, SOAP can use any token.

A.2. A RESTful Approach for CCMP

Conference objects can also be modeled as resources identified by URIs, with the basic CRUD operations mapped to the HTTP methods POST/PUT for creating objects, GET for reading objects, PATCH/POST/PUT for changing objects, and DELETE for deleting them. Many of the objects, such as conferences, already have natural URIs.

CCMP can be mapped into the CRUD (Create, Read, Update, Delete) design pattern. The basic CRUD operations are used to manipulate conference objects, which are XML documents containing the information characterizing a specified conference instance, be it an active conference or a conference blueprint used by the conference server to create new conference instances through a simple clone operation.

Following the CRUD approach, CCMP could use a general-purpose protocol such as HTTP [RFC2616] to transfer domain-specific XML-encoded data objects defined in the "Conference Information Data Model for Centralized Conferencing" [RFC6501].

Following on the CRUD approach, CCMP could follow the well-known REST (REpresentational State Transfer) architectural style [REST]. CCMP could map onto the REST philosophy, by specifying resource URIs, resource formats, methods supported at each URI and status codes that have to be returned when a certain method is invoked on a specific URI. A REST-style approach must ensure sure that all operations can be mapped to HTTP operations.

The following summarizes the specific HTTP method that could be used for each of the CCMP Requests:

Retrieve: HTTP GET could be used on XCON-URIs, so that clients can obtain data about conference objects in the form of XML data model documents.

Create: HTTP PUT could be used to create a new object as identified by the XCON-URI or XCON-USERID.

Change: Either HTTP PATCH or HTTP POST could be used to change the conference object identified by the XCON-URI.

Delete: HTTP DELETE could be used to delete conference objects and parameters within conference objects identified by the XCON-URI.

Authors' Addresses

Mary Barnes
Polycom
TX
USA

EMail: mary.ietf.barnes@gmail.com

Chris Boulton
NS-Technologies

EMail: chris@ns-technologies.com

Simon Pietro Romano
University of Napoli
Via Claudio 21
Napoli 80125
Italy

EMail: spromano@unina.it

Henning Schulzrinne
Columbia University
Department of Computer Science
450 Computer Science Building
New York, NY 10027

EMail: hgs+xcon@cs.columbia.edu