XCON Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 20, 2010

M. Barnes
Nortel
C. Boulton
NS-Technologies
S P. Romano
University of Napoli
H. Schulzrinne
Columbia University
June 18, 2010

Centralized Conferencing Manipulation Protocol
draft-ietf-xcon-ccmp-08

Abstract

The Centralized Conferencing Manipulation Protocol (CCMP) allows an XCON conferencing 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 state-less, 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.

Table of Contents

1. Introduction

The 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 end points.

The CCMP implements the client-server model within the XCON Framework, with the Conference Control Client and Conference Control Server acting as client and server, respectively. The CCMP uses HTTP [RFC2616] as the protocol to transfer requests and responses, which contain the domain-specific XML-encoded data objects defined in [I-D.ietf-xcon-common-data-model] Conference Information Data Model for Centralized Conferencing (XCON Data Model).

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 the specific CCMP messages. A complete, not normative, example of the operation of the 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 Control Server is provided in Section 7, whereas 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

In additon to the terms defined in the Framework for Centralized

Conferencing [RFC5239], this document uses the following terms and acronyms:

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

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.

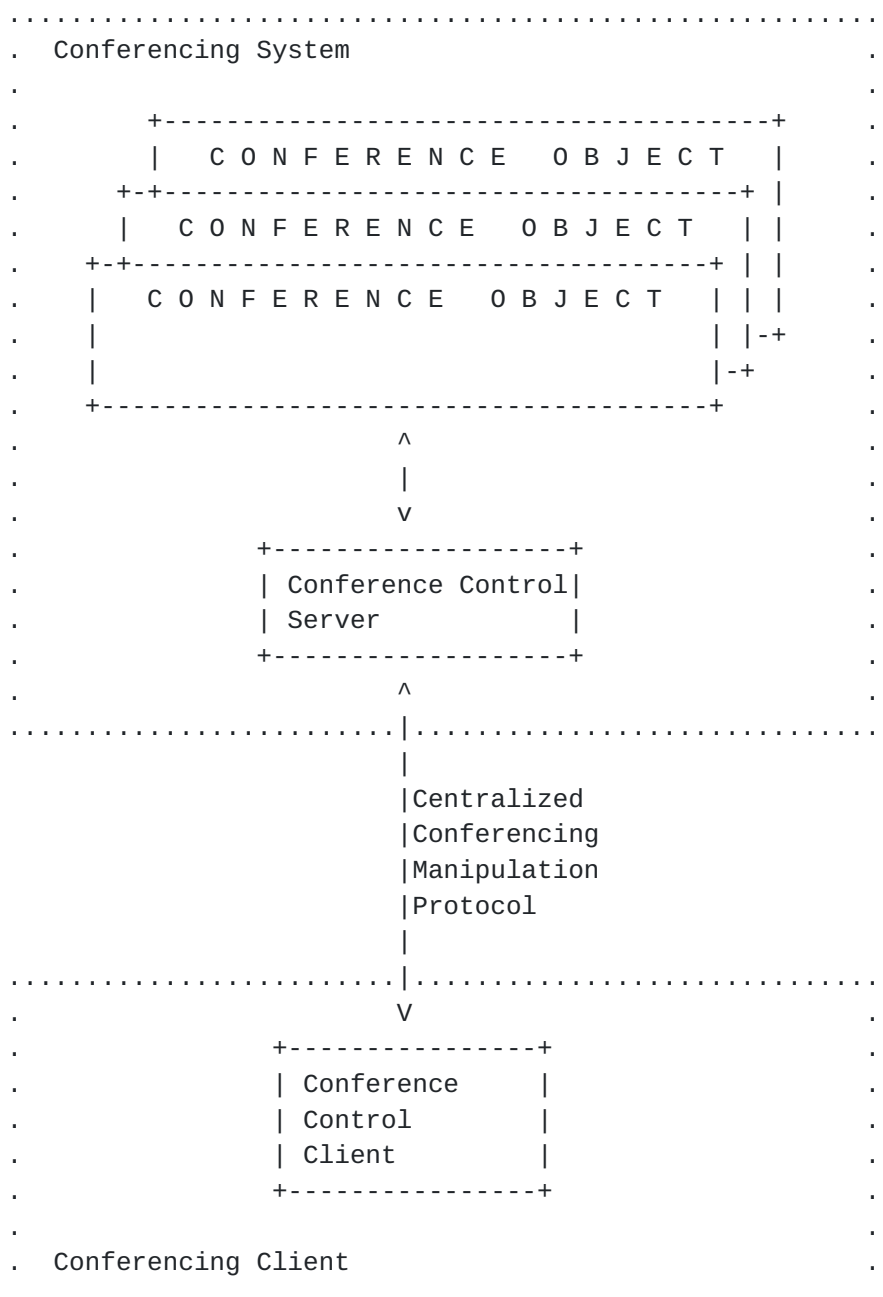


Figure 1: Conference Client Interaction

CCMP serves as the Conference Control Protocol, allowing the conference control client to interface with the conference object maintained by the conferencing system, as represented in Figure 1. Conference Control is one part of functionality for advanced conferencing supported by a conferencing client. Other functions are discussed in the XCON framework and related documents.

Conference object and conference users do represent key elements involved in Conference Control Protocol operations. Their identifiers, respectively the conference XCON-URI and the conferencing client XCON-USERID, and their XML representations compliant with the XML Schema defined in the XCON data model are widely used for creating the CCMP requests and responses. The main conference objects and users features envisioned by the XCON framework are briefly described in the following subsections.

3.1. Conference Objects

Conference objects feature a simple dynamic inheritance-and-override mechanism. Conference objects are linked into a tree known as "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 (conference reservations). An object can mark certain of its properties as unalterable, so that they cannot be overridden. It is envisaged by the framework that a client may specify a parent object (a conference or blueprint) from which the conference to be created has to inherit values by the mean of the Conference Control Protocol.

Conference objects are uniquely identified by the XCON-URI within the scope of the conferencing system. Such identifier 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 [I-D.ietf-xcon-common-data-model]. 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. By the mean of CCMP, conferencing clients can use such XML structures to express their preferences in creation or update. A conferencing server can convey conference information of such a form back to the clients.

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 Control 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 5.3.7 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 signalling 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, in the first place the <users> element, as well as each <user> element included in 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-user-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. The 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. To this purpose, 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. The CCMP does not look after users 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.

4. Protocol Overview

CCMP is a client-server, XML-based protocol, which has been specifically conceived to provide users with the necessary means for the creation, retrieval, modification and deletion of conference objects. CCMP is also state-less, which means implementations can

safely handle transactions independently from each other. Conference-related information is encapsulated into CCMP messages in the form of XML documents or XML document fragments compliant with the XCON data model representation.

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 in the CCMP protocol includes conference blueprints, the conference object, users, and sidebars.

CCMP has been conceived as completely independent from underlying protocols, which means that there can be different ways to carry CCMP messages across the network, from a conferencing client to a conferencing server. Nevertheless, it is recommended to use HTTP as a transport solution, including CCMP requests in HTTP POST messages and CCMP responses in HTTP 200 OK replies. Implementation details are presented in Section 4.2

4.1. Protocol Operations

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

create: for the creation of a conference, 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 or 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:

- 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>),

- o <user> elements associated with conference users,
- o the list of XCON-URIs related to conferences and blueprints available at the server, for which only retrieval operations are allowed.

Each operation in the protocol model 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 conference client completes prior to another client's operation on the same conference object. The details for this data locking functionality are out of scope for the CCMP protocol specification and are implementation specific for a conference server. Thus, 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. Also, since multiple clients can modify the same conference objects, conference clients should first obtain the current object from the conference server and then update the relevant data elements in the conference object prior to invoking a specific operation on the conference server. In order to effectively manage modifications to conference data, a versioning approach is exploited in the CCMP. More precisely, each conference object is associated with a version number indicating the most up to date view of the conference at the server's side. Such version number is reported to the clients when answering their requests. A client willing to make modifications to a conference object has to send an update message to the server. In case the modifications are all successfully applied, the server sends back to the client a "200" response which also carries information about the current server-side version of the modified object. With such approach, a client which is working on version "X" of a conference object and finds inside a "200" response a version number which is "X+1" can be sure that the version it was aware of was the most up to date. On the other hand, if the "200" response carries back a version which is at least "X+2", the client can detect that the object that has been modified at the server's side was more up to date than the one it was working upon. This is clearly due to the effect of concurrent modification requests issued by independent clients. Hence, for the sake of having available the latest version of the modified object, the client can send to the conference server a further "retrieve" request. In no case a copy of the conference object available at the server is returned to the client as part of the update response message. Such a copy can always be obtained through an ad-hoc "retrieve" message.

Based on the above considerations, all CCMP response messages carrying in their body a conference document (or a fragment of it) must contain a "version" parameter. This does not hold for request

messages, for which the "version" parameter is not at all required, since it represents useless information for the server: as long as the required modifications can be applied to the target conference object with no conflicts, the server does not care whether or not the client had an up to date view of the information stored at its side. This said, it stands clear that a client which has subscribed at the server, through the XCON event package [I-D.ietf-xcon-event-package], to notifications about conference object modifications, will always have the most up to date version of that object available at his side.

A final consideration concerns the relation between the CCMP and the main entities it manages, i.e. conference objects. Such objects have to be compliant with the XCON data-model, which identifies some elements/attributes as mandatory. From the CCMP standpoint this can become a problem in cases of client-initiated operations, like the creation/update of conference objects. In such cases, not all of the mandatory data can be known in advance to the client issuing a CCMP request. As an example, a client has no means to 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 and hence it is not able to appropriately fill with that value the mandatory "entity" attribute of the conference document contained in the request. To solve this kind of issues, the CCMP will fill all mandatory data model fields, for which no value is available at the client at the time the request is constructed, with fake values in the form of wildcard strings (e.g. AUTO_GENERATE_X, with X being an incremental index initialized to a value of 1). Upon reception of the mentioned kinds of requests, the server will: (i) generate the proper identifier(s); (ii) produce a response in which the received fake identifier(s) carried in the request has (have) been replaced by the newly created one(s). With this approach we maintain compatibility with the data model requirements, at the same time allowing for client-initiated manipulation of conference objects at the server's side (which is, by the way, one of the main goals for which the CCMP protocol has been conceived at the outset).

4.2. Implementation Approach

There have been a number of different proposals as to the most suitable implementation solution for the CCMP. A non-exhaustive summary of the most interesting ones is provided in Appendix A. The solution for the 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, the 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.

The remainder of this document focuses on the selected approach. The CCMP protocol inserts XML-based CCMP requests into the body of HTTP POST operations and retrieves responses from the body of HTTP "200 OK" messages. CCMP messages have a MIME-type of "application/ccmp+xml", which appears inside the "Content-Type" and "Accept" fields of HTTP requests and responses. Section 9 provides the complete requirements for an HTTP implementation to support the 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 which 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 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 entailed operation. The same fields can also be exploited to carry out other Authorization, Authentication and Accounting (AAA) 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 assinged by the conferencing server at 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 above mentioned case will be provided in Section 5.3.7.

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: An optional parameter that MUST be inserted in all requests whose target conference object is password-protected (as per the <conference-password> element in [I-D.ietf-xcon-common-data-model]).

specialized request message: This is 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 mandatory parameter in CCMP response messages containing the XCON-USERID of the conferencing client who 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 mandatory 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, such 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.

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 ref="response-code" 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 Section 5.1 and Section 5.2, the following summarizes the specialized CCMP request/response types described in this document:

1. optionsRequest/optionsResponse

2. blueprintsRequest/blueprintsResponse
3. confsRequest/confsResponse
4. blueprintRequest/blueprintResponse
5. confRequest/confResponse
6. usersRequest/usersResponse
7. userRequest/userResponse
8. sidebarsByValRequest/sidebarsByValResponse
9. sidebarsByRefRequest/sidebarsByRefResponse
10. sidebarByValRequest/sidebarByValResponse
11. sidebarByRefRequest/sidebarByRefResponse

These CCMP request/response pairs use the fundamental CCMP operations as defined in Section 4.1 to manipulate the conference data. The optionsRequest/optionsResponse message pair deserves a specific discussion, since it is not used for manipulating information about either conferences or conference users, but rather to retrieve general information about conference server capabilities, in terms of standard CCMP messages it supports, plus potential extension messages it understands, as it will be further explained in Section 5.3.1. 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. The corresponding response MUST contain the same operation. Note that some entries are labeled "N/A" indicating the operation is invalid for that request type. In the case of an "N/A*", 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	Gets conference object	Creates conference object	Changes conference object	Deletes conference object
usersRequest	Gets <users>	N/A(**)	Changes <users>	N/A(**)
userRequest	Gets specified <user>	Adds a <user> to a conf (***)	Changes specified <user>	Deletes specified <user>
sidebarsByVal Request	Gets <sidebars-by-val>	N/A	N/A	N/A
sidebarsByRef Request	Gets <sidebars-by-ref>	N/A	N/A	N/A
sidebarByVal Request	Gets sidebar-by-val	Creates sidebar-by-val	Changes sidebar-by-val	Deletes sidebar-by-val
sidebarByRef Request	Gets sidebar-by-ref	Creates sidebar-by-ref	Changes sidebar-by-ref	Deletes 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 conjuntcion with, respectively, the creation and deletion 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/ response messages are detailed in the subsequent sections.

5.3.1. optionsRequest and optionsResponse

An "optionsRequest" (Figure 4) message is basic CCMP message, i.e. it does not provide any specialization of the general CCMP request. Coming to the response, it can contain information about both standard (i.e. IETF-defined) CCMP messages and extension messages supported by the server. In both cases, the response carries back a list of names of the supported (standard and extended) messages. 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 (in the <schema-ref> element).

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

<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>

<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>


<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>

<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>

<xs:complexType name="standard-message-type">
  <xs:sequence>
        <xs:element name="name" type="standard-message-name-type"
minOccurs="1"/>
        <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>

<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>

<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>

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

<!-- options -->
```

Figure 4: Structure of the optionsRequest and optionsResponse messages

5.3.2. blueprintsRequest and blueprintsResponse

A "blueprintsRequest" (Figure 5) message is sent to request the list of XCON-URIs associated with the available blueprints from the conference server. Such URIs can be subsequently used by the client to access detailed information about a specified blueprint with a specific blueprintRequest message per Section 5.3.4. The "confUserID" parameter MUST be included in every blueprintsRequest/Response message and reflect the XCON-USERID of the conferencing client issuing the request. A blueprintsRequest message REQUIRES no "confObjID" and "operation" parameters, since it is not targetted to a specific conference instance and it is conceived as a "retrieve-

only" request. 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. 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']".

The associated "blueprintsResponse" message SHOULD contain, as shown in Figure 5, 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 5: Structure of the blueprintsRequest and blueprintsResponse messages

5.3.3. 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 6: Structure of the confsRequest and confsResponse messages

5.3.4. blueprintRequest and blueprintResponse

Through a "blueprintRequest", a client can manipulate the conference object associated with a specified blueprint. Further than the "confUserID" parameter, the request MUST include the "confObjID" and the "operation" one. 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 other than "retrieve". The "create", "update" and "delete" operations 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" which is not associated with one of the available system's blueprints will generate, on the server's side, a blueprintResponse message containing a "404" error code. This holds also for the case in which the mentioned "confObjID" 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.

In the case of "response-code" of "200" for 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 7: Structure of the blueprintRequest and blueprintResponse messages

5.3.5. 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. The requirements for inclusion of "confObjID" and "confInfo" parameter in the confRequest/confResponse messages and are detailed below for each "operation" case.

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;
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 transporting the created conference document MAY be included, at the discretion of the conferencing system implementation, along with an optional version parameter initialized at "1", since at creation time the conference object is at its first version.

In the case of a confRequest with a "retrieve" operation, the "confObjID" representing the XCON-URI of the target conference MUST be included and the "confInfo" parameter MUST NOT be included in the request. The conferencing server MUST ignore any "confInfo" parameter that is received in a confRequest/retrieve. 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 "update" operation, the "confInfo" and "confObjID" 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 is "confObjID". Note that, in such a case, though the confInfo parameter has indeed to follow the rules indicated in the XCON data model, it does not represent the entire updated version of the target conference, since it rather 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 8: 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 9: 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 confInfo parameter is not mandatory. A following confRequest/retrieve transaction might provide the CCMP client with the current aspect of the conference upon the modification, or the Notification Protocol might address that task as well. A "200"

response-code indicates that the referenced conference document has been changed accordingly to the request by the conferencing server. The "version" parameter MUST be enclosed in the confResponse/update message, in order to let the client understand what is the actual current conference-object version, upon the applied modifications. An "409" response-code indicates that the changes reflected in the request "confInfo" 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 the "409" response-code, the "version" parameter MUST be attached in the confResponse/update, by this way allowing the client to eventually 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 a "delete" operation, the "confObjID" representing the XCON-URI of the target conference MUST be included while the "confInfo" MUST NOT be included in the request. The conferencing server MUST ignore any "confInfo" parameter that is received within such a request. The confResponse MUST contain the same "confObjID" that was included in the confRequest. If the confResponse/delete operation contains a "200" response-code, the conference indicated in the "confObjID" has been successfully deleted. A "200" confResponse/delete MUST NOT contain the "confInfo" parameter. The "version" parameter SHOULD NOT be returned in any confResponse/delete. If the conferencing server cannot delete the conference referenced by the "confObjID" received in the confRequest because it is the parent of another conference object that is in use, the conferencing server MUST return a response-code of "425".

A confRequest with an "operation" of "retrieve", "update" or "delete" carrying a "confObjID" which is not associated with one of the conferences (active or registered) the system is holding will generate, on the server's side, a confResponse message containing a "404" error code. This holds also for the case in which the mentioned "confObjID" 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 10.

```
<!-- 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 10: Structure of the confRequest and confResponse messages

5.3.6. usersRequest and usersResponse

Through a "usersRequest" message the CCMP client manipulates the XML <users> element of the conference document associated with the conference identified by the "confObjID" parameter complusory included in the request. Inside the <users> element, along with the list of <user> elements associated with conference participants, there is information that the client may be interested in controlling, such the list of the users to which access to the conference is allowed/denied, conference participation policies, etc.; for this reason, a customized message has been designed to allow for the manipulation of this specific part of a conference document.

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 a "retrieve" operation.
2. update: In this case, the "usersInfo" parameter MUST contain the modifications to be applied to the referred <users> element. 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 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 11: Structure of the usersRequest and usersResponse messages

5.3.7. 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", the "operation" parameter, and 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 AAA checks), like creating a conference, retrieving conference information, etc..

Conference users can be created in a number of different ways. In each of these cases the operation 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:

- 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 which 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".
- Conferencing client acts on behalf of a third 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 third 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 such 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 instigate the signaling phase needed to add her to the conference. The mandatory "entity" attribute of the "userInfo" parameter in the request is filled with a dummy value recognizable on the server side, so to conform to the rules contained in the XCON data model XML schema. The involved messages (userRequest and userResponse) in such case should look like the following:

```
Request fields:

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

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


Response fields (in case of success):

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

Figure 12: userRequest and userResponse in the absence of an xcon-userid

- 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 one and the "entity" attribute of the attached userInfo is filled with the pre-defined fake value "xcon-userid:AUTO_GENERATE@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". [RP] 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 is not yet 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 conferencing 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 signalling URI: if the joining user is a brand new one, then the generation of a new XCON identifier is needed; otherwise, if that user is an existing one, the server must recover the corresponding XCON identifier.

In the case of a userRequest with a "retrieve" operation, the "confObjID" 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 "update" operation, the "confObjID", "confUserID" and "userInfo" MUST be included in the request. The "userInfo" 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" 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 a "delete" operation, the "confObjID" representing the XCON-URI of the target conference MUST be included. The "confUserID", 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 "confObjID" that was included 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 13: Structure of the userRequest and userResponse messages

5.3.8. sidebarsByValRequest and sidebarsByValResponse

A "sidebarsByValRequest" 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, also with sidebars, 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 to sidebars by value. A "sidebarsByValResponse" with a "response-code" of "200" MUST contain a "sidebarsByValInfo" parameter containing the desired <sidebars-by-val> element. A "sidebarsByValResponse" 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) to 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 5.3.9.

```
<!-- 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 14: Structure of the sidebarsByValRequest and sidebarsByValResponse messages

5.3.9. sidebarByValRequest and sidebarByValResponse

A sidebarByValRequest message MUST contain the "operation" parameter which discriminates 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, as envisaged in the XCON framework ([RFC5239]), the sidebar is created by cloning the main conference, following the implementation specific cloning rules. Otherwise, similarly 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 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 are authorized to receive the notifications, of the addition of the sidebar to the conference.

In the case of a "sidebarByVal" request with an operation of "retrieve", the URI for the conference object created for the sidebar (received in the 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 a "retrieve" operation included in a confRequest message as detailed in Section 5.3.5.

In the case of a "sidebarByVal" request with an operation 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 "sidebarByValInfo" 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 5.3.5. A "sidebarByValResponse" 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" 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" 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 are authorized to receive the notifications, of the deletion of the sidebar to the conference.

If a sidebarByValRequest with an "operation" of "retrieve", "update" or "delete" carries a "confObjID" which is not associated with any existing sidebar-by-val, a confResponse message containing a "404" error code will be generated on the server's side. This holds also for the case in which the mentioned "confObjID" 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 15: Structure of the sidebarByValRequest and sidebarByValResponse messages

5.3.10. 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 "sidebarByRef" request message, described in Section 5.3.11. 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) to 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 16: Structure of the sidebarsByRefRequest and sidebarsByRefResponse messages

5.3.11. sidebarByRefRequest and sidebarByRefResponse

A sidebarByRefRequest message MUST contain the "operation" parameter which discriminates 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 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, as envisaged in the XCON framework ([RFC5239]), the sidebar is created by cloning the main conference, following the implementation specific cloning rules. Otherwise, similarly 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 sidebarByRef request), 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 are authorized to receive the notifications, of the addition of the sidebar to the conference.

In the case of a "sidebarByRef" request with an operation 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 5.3.5.

In the case of a "sidebarByRef" request with an operation 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 "operation" of "update". 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 5.3.5. 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" of "delete" is included in the sidebarByRef request, 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" 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 are authorized to receive the notifications, of the deletion of the sidebar.

If a sidebarByRefRequest with an "operation" of "retrieve", "update" or "delete" carries a "confObjID" which is not associated with any existing sidebar-by-ref, a confResponse message containing a "404" error code will be generated on the server's side. This holds also for the case in which the mentioned "confObjID" 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 17: Structure of the sidebarByRefRequest and sidebarByRefResponse messages

5.4. CCMP Response Codes

All CCMP response messages MUST include a ""response-code"". 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: Target conference object missing at the server (it 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 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 missing at the server (this code is returned in the case of requests in which the "confUserID" of the sender is invalid).

422 Invalid Conference Password: Target conference object's password 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 which depend on it, or because it effects, 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).

500 Server Internal Error: The server cannot complete the required service due to a system internal error.

501 Not Implemented: Operation envisaged in the protocol, but not implemented in the contacted server.

510 Request Timeout: The time required to serve the request has exceeded the envisaged 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" and "426" are 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 sidebars ByRefRequest	All retrieve requests, EXCEPT: blueprints Request, confsRequest	All update requests	All delete requests
409	N/A	N/A	All update requests	N/A
420	userRequest(3rd party invite with thir duser entity) (*)	userRequest	userRequest	userRequest
421	All create requests, EXCEPT: userRequest twith 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

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" of the sender of the request that is invalid, a "421" error code is returned to the client.

(**) "421" is not sent in answers 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 conference control client or conference control server.

A response code of "400" indicates that the conference control client sent a malformed request, which is indicative of an error in the conference control client or in the conference control server. The handling is specific to the conference control 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.

Response codes such as "401" and "403" indicate 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 "500" "response-code", the handling of which is specific to the conference control client implementation.

6. A complete example of the CCMP in action

In this section a typical, not normative, scenario in which the CCMP comes into play is described, by showing the actual composition of the various CCMP messages. In the call flows of the example, the Conference Control Client is a CCMP-enabled client, whereas the Conference Control Server is a CCMP-enabled server. The "confUserID" of the client, Alice, is "xcon-userid:Alice@example.com" and appears in all requests. The sequence of operations is as follows:

1. Alice retrieves from the server the list of available blueprints (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, 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 "confUserID" is "xcon-userid:Ciccio@example.com") join the conference (Section 6.7).

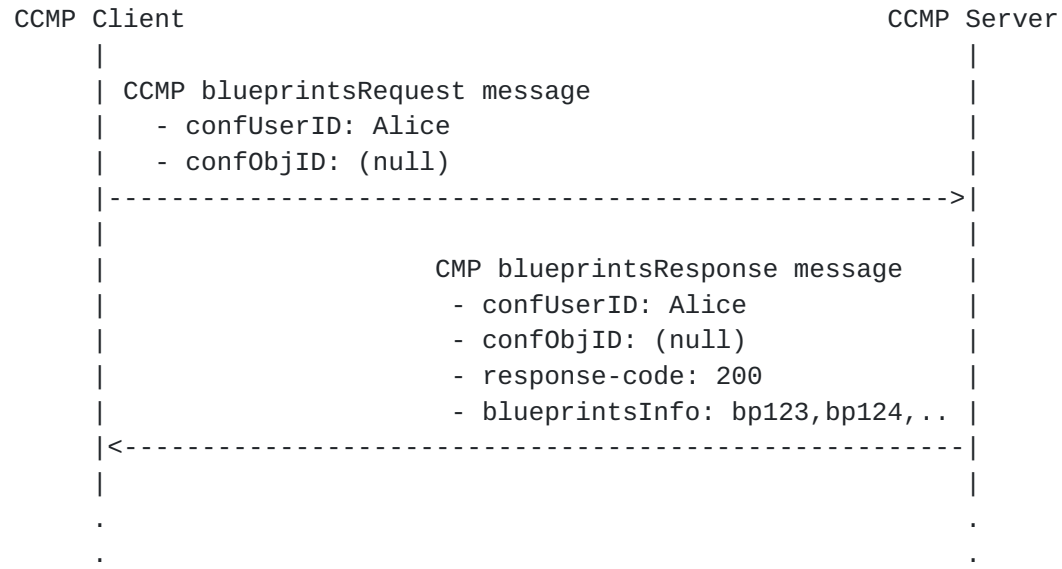
Note, the examples do not include any details beyond the basic operation.

In the following sections we deal with each of the above mentioned 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 it comes out from 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="xcon:ccmp-blueprints-request-message-type">
       <confUserID>xcon-userid:Alice@example.com</confUserID>
    </ccmpRequest>
   </ccmp:ccmpRequest>
```

2. blueprintsResponse message form 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>
    <ccmp:response-code>200</ccmp: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,
```

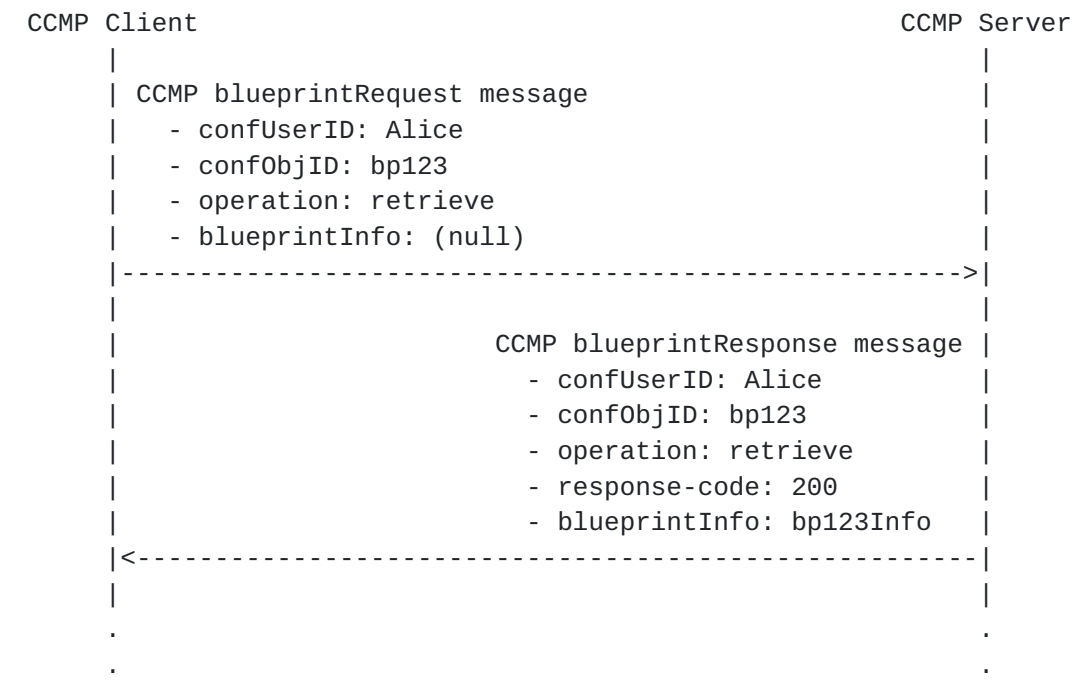
```
                only one user can talk at the same time,
                and the requests for the AudioFloor MUST
                be accepted by a Chair.
             </info:purpose>
            </info:entry>
            <info:entry>
             <info:uri>xcon:VideoConference1@example.com</info:uri>
             <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 18: 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 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 form 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>
```

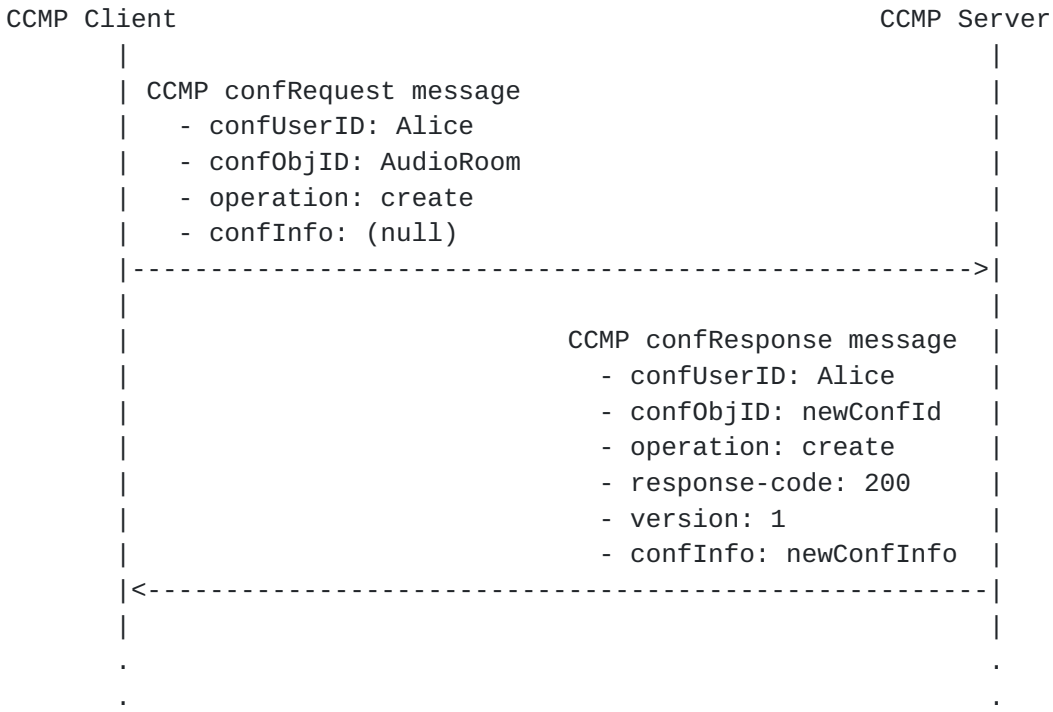
```
      <ccmp:response-code>200</ccmp:response-code>
      <ccmp:blueprintResponse>
        <blueprintInfo entity="xcon:AudioRoom@example.com">
          <info:conference-description>
             <info:display-text>AudioRoom</info:display-text>
             <info:maximum-user-count>2</info:maximum-user-count>
             <info:available-media>
               <info:entry label="audioLabel">
                   <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="audioLabel"></xcon:floor>
            </xcon:conference-floor-policy>
          </xcon:floor-information>
        </blueprintInfo>
      </ccmp:blueprintResponse>
    </ccmpResponse>
  </ccmp:ccmpResponse>
```

Figure 19: Getting info 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 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" 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>
       <ccmp:response-code>200</ccmp:response-code>
       <ccmp:version>1</ccmp: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:conf-uris>
                     <info:entry>
                        <info:uri>
                            xcon:8977794@example.com
                        </info:uri>
                        <info:display-text>
                            conference xcon-uri
                        </info:display-text>
                        <xcon:conference-password>
                            8601
                        </xcon:conference-password>
                     </info:entry>
                  </info:conf-uris>
                  <info:maximum-user-count>10</info:maximum-user-count>
                  <info:available-media>
                         <info:entry label="11">
                             <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:conference-floor-policy>
                 </xcon:floor-information>
```

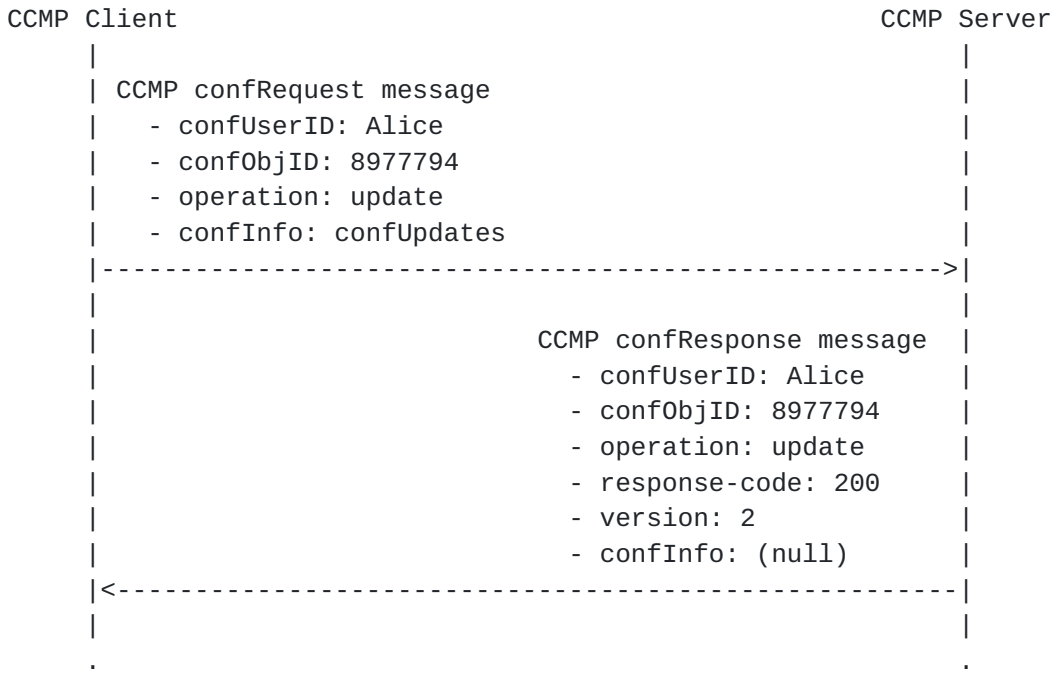
```
            </confInfo>
          </ccmp:confResponse>
      </ccmpResponse>
  </ccmp:ccmpResponse>
```

Figure 20: 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 form the server:

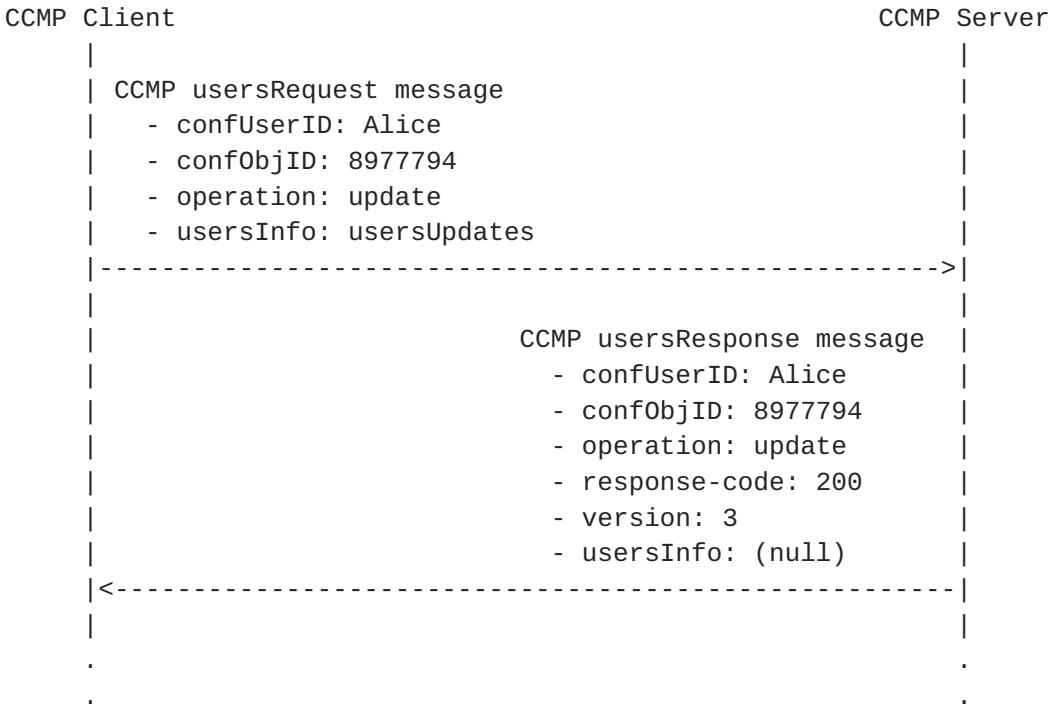
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
    xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
    xmlns:info="urn:ietf:params:xml:ns:conference-info"
          xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp">
  <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"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

Figure 21: Updating conference information

6.5. Alice inserts a list of users in 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 the purpose, she exploits the "usersRequest" message provided by the CCMP. The picture below shows the transaction.

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:+390817683823"/>
                    <xcon:target method="refer"
                                 uri="sip:Carol@example.com"/>
                </xcon:allowed-users-list>
            </usersInfo>
        </ccmp:usersRequest>
    </ccmpRequest>
</ccmp:ccmpRequest>
```

2. usersResponse message form 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>
        <ccmp:response-code>200</ccmp:response-code>
        <ccmp:version>3</ccmp:version>
        <ccmp:confResponse/>
    </ccmpResponse>
</ccmp:ccmpResponse>
```

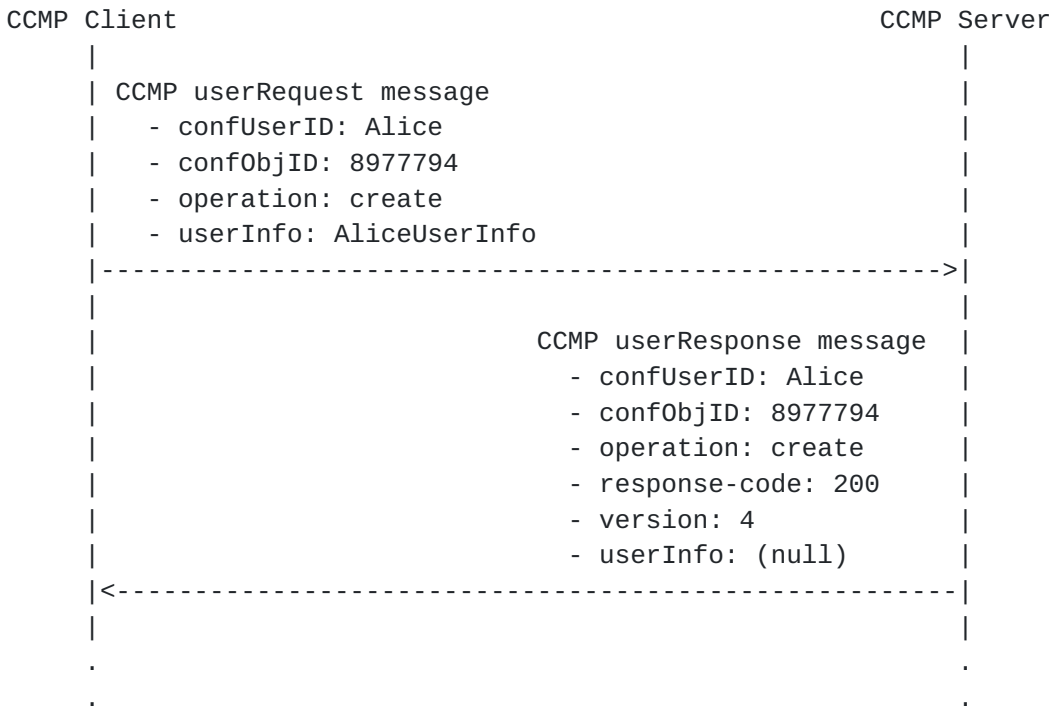
Figure 22: Updating the list of allowed users for the conference 'xcon:8977794@example.com'

6.6. Alice joins the conference

This section illustrates the sixth transaction in the overall flow. Alice uses the 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 element includes information about the user's Address of Records, as well as her current end-point. 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 form the server:

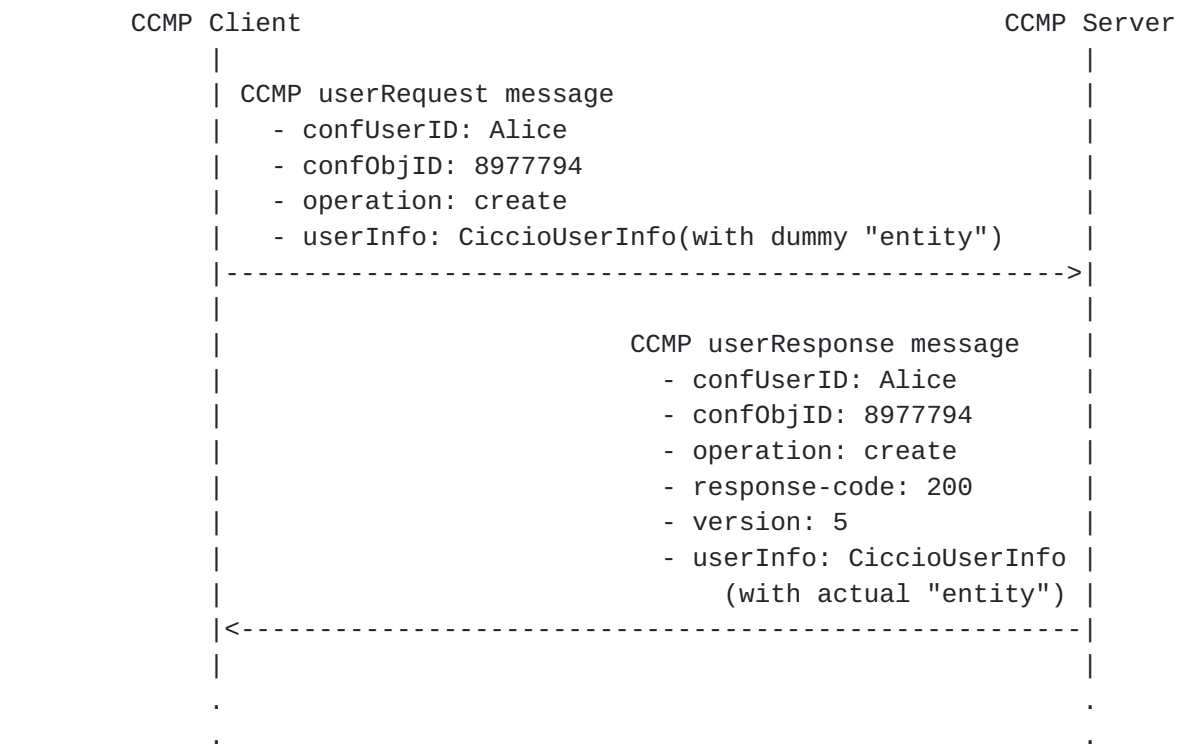
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
    xmlns:info="urn:ietf:params:xml:ns:conference-info"
    xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp"
    xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
   <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"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

Figure 23: Alice joins the conference through the 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 the 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 element includes information about Ciccio's Address of Records, as well as his current end-point, but has a fake "entity" attribute, since it is unknown to Alice, in this example. Thus, the server is in charge of generating a new XCON-USERID for the user Alice indicates, 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@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 form 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>
        <ccmp:response-code>200</ccmp:response-code>
        <ccmp:version>5</ccmp: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 24: Alice adds a new user to the conference through the CCMP

7. Locating a Conference Control Server

If a conference control client is not pre-configured to use a specific conference control server for the requests, the client MUST first discover the conference control 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.

This document proposes the use of DNS to locate the conferencing server. U-NAPTR resolution for conferencing takes a domain name as input and produces a URI that identifies the conferencing 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 the CCMP protocol.

The NAPTR records in the following example Figure 25 demonstrate the use of the Application Service and 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 25: 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], the CCMP is one of the following four protocols which have been formally identified within the XCON framework:

Conference Control Protocol: between Conference and Media Control Client and Conference Server. Such protocol is the subject of the present document.

Binary floor Control Protocol: between the Floor Control Client and the Floor Control Server. Such protocol is the BFCP, specified in [RFC4582].

Call Signaling Protocol: between the Call Signaling Client and the Focus. Such protocol can be either SIP or any other call signaling protocol (e.g. H.323, IAX, etc.) capable of negotiating a conferencing session.

Notification Protocol: between the Notification Client and the XCON Notification Service. Such protocol has not been identified as yet.

The CCMP protocol specified in this document is a pro-active one and is used by a conferencing client to send requests to a conferencing server in order to retrieve information about the conference objects it stores and potentially manipulate them. Though, it stands clear that a complete conferencing solution cannot refrain from providing clients with a means for receiving asynchronous updates about the status of the objects available at the server. The notification protocol to be adopted in XCON, 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 it is discussed in the following.

When the conference 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 conference control client MUST implement the Conference event package for XCON [I-D.ietf-xcon-event-package]. 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 conference client provides the conference server with an HTTP URL which 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 backend interface to the conference server. And, this model can co-exist 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 backend would probably find the HTTP call back approach much easier. The details of this approach are out of scope for the CCMP per se, thus the expectation is that 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 the CCMP protocol, 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 may 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 implementors avoid ambiguity with the many options the full HTTP protocol offers.

A conferencing client that conforms to this specification is not required to support HTTP authentication [RFC2617] or cookies [RFC2965]. These mechanism are unnecessary because CCMP requests carry their own authentication information (in the "subject" field; see Section 5.1).

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 a 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 an escaped URL (e.g., somebody found a URL in protocol traces or log files and fed it into their browser) or somebody doing testing/ debugging. 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 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 conference control client and the conference control 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 theft of service by an endpoint 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 conferencing server itself. 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., SIPS) and media protocols (e.g., SRTP).

10.1. Assuring that the Proper Conferencing Server has been contacted

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 conference client by presenting a certificate containing that identifier as a subjectAltName (i.e., as an iPAddress or dNSName, respectively). With the use of HTTP as a transport for CCMP, this is exactly the authentication described by TLS [RFC2818]. If the client has external information as to the

expected identity or credentials of the proper conference server (e.g., a certificate fingerprint), these checks MAY be omitted. Any implementation of CCMP MUST be capable of being transacted over TLS so that the client can request the above authentication, and a conference server implementation MUST include this feature. Note that in order for the presented certificate to be valid at the client, the client must be able to validate the certificate. 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.

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 conferencing 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 the case of users joining the conference using the CCMP, TLS is RECOMMENDED.

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 correlate the authorization of the CCMP interface with other authorization mechanisms - e.g., 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 the CCMP (i.e., in the appropriate CCMP response messages). This conference user identifier is REQUIRED for any subsequent operations on the conference object.

We herein remark that the definition of a complete solution for policy-based management of an XCON-compliant conference system is out of the scope of this document, as well as of the XCON WG. We believe

that the specification of such a framework is orthogonal to the overall XCON architecture, especially if a Role Based Access Control (RBAC) approach is embraced. 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 might work on the definition of an RBAC framework for XCON, by first focusing on the definition of roles and eventually arriving at the specification of 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. Finally, coming to the issue of assigning users to roles, this can be done through so-called role-assignment policies. Such assignment should be under the responsibility of an ad-hoc dedicated Role Assignment entity.

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 the CCMP protocol through the use of TLS.

The conference server SHOULD provide mechanisms to ensure the privacy of the XCON-USERID. This is accomplished by the conference client manipulation of the "provide-anonymity" element defined in the XCON data model ([I-D.ietf-xcon-common-data-model]. The "provide-anonymity" element controls the degree to which a user reveals their identity. The conference client MUST set the "provide-anonymity" element to "hidden" if the user does not want other participants to even be aware that there is an additional participant in the conference. The conference client MUST set the "provide-anonymity" field to "private" if the user wants to be entirely "anonymous" (i.e., other participants are aware that there is another participant, but have no information as to their identity). The conference client MUST set the "provide-anonymity" field to "semi-private" if their 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 an administrator can see all users). To provide the required privacy, the conference client SHOULD include the "provide-anonymity" element in the "confInfo" parameter in a CCMP confRequest message with an "update" or "create" operation or in the "userInfo" parameter in a CCMP userRequest message with an "update" or "create" operation, to ensure

the user is provided the appropriate level of privacy. If the "provide-anonymity" element is not included in the conference object, then other users can see the participant's identity.

11. XML Schema

This section provides the XML schema definition of the "application/ccmp+xml" format.

```
<?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 response definition -->

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

  <!-- 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>

  <!--  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 ref="response-code" 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>

<!-- 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>

   <!-- response-code -->

   <xs:element name="response-code" type="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>

<!-- CCMP request extensions -->
   <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>

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

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

```
<!-- CCMP response extensions -->
  <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>

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

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

<!-- options -->

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

        <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>

    <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>


    <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>

     <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>

    <xs:complexType name="standard-message-type">
      <xs:sequence>
        <xs:element name="name" type="standard-message-name-type"
minOccurs="1"/>
        <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>

    <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>

 <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>

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

    <!-- options -->

</xs:schema>
```

Figure 26

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. This document also 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.barnes@nortel.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>
[[NOTE TO IANA/RFC-EDITOR: Please update RFC URL and replace XXXX
    with the RFC number for this specification.]]
      <p>See <a href="[[RFC URL]]">RFCXXXX</a>.</p>
    </body>
  </html>
END
```

12.2. XML Schema Registration

This section registers an XML schema as 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.barnes@nortel.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
Indicates the character encoding of enclosed XML for which the default is UTF-8.
Encoding considerations: Uses XML, which can employ 8-bit characters, depending on the character encoding used. See RFC 3023 [RFC3023], section 3.2.
Security considerations: This content type is designed to carry protocol data related conference control. Some of the data could be considered private and thus should be protected.
Interoperability considerations: None.
Published specification: RFC XXXX [[NOTE TO IANA/RFC-EDITOR: Please replace XXXX with the RFC number for this specification.]]
Applications which use this media type: Centralized Conferencing control clients and servers.
Additional Information: Magic Number(s): (none)
File extension(s): .xml
Macintosh File Type Code(s): (none)
Person & email address to contact for further information: Mary Barnes <mary.barnes@nortel.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 the CCMP protocol.

12.4.1. Registration of a Conference Control 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: RFCXXXX

Contact information: The authors of this document

Author/Change controller: The IESG

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

This section registers a new S-NAPTR/U-NAPTR Application Protocol tag for the CCMP protocol, 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: RFCXXXX

Contact Information: The authors of this document

Author/Change Controller: The IESG

12.5. CCMP Protocol Registry

This document requests that the IANA create a new registry for the CCMP protocol including an initial registry for operation types and response codes.

12.5.1. CCMP Message Types

The CCMP messages are described in Section 4.1 and defined in the XML schema in Section 11. The following summarizes the requested registry:

Related Registry: CCMP Message Types Registry
Defining RFC: RFC XXXX [NOTE TO IANA/RFC-EDITOR: Please replace XXXX with the RFC number for this specification.]
Registration/Assignment Procedures: New CCMP message types are allocated on a specification required basis.
Registrant Contact: IETF, XCON working group, (xcon@ietf.org), Mary Barnes (mary.barnes@nortel.com).

This section pre-registers the following initial CCMP message types:

optionsRequest: Used by a conference control client to query a conferencing system for its capabilities, in terms of supported messages (both standard and non-standard).

optionsResponse: The optionsResponse returns a list of CCMP messages (both standard and non-standard) supported by the specific conference server.

blueprintsRequest: Used by a conference control client to query a conferencing system for its capabilities, in terms of available conference blueprints.

blueprintsResponse: The blueprintsResponse returns a list of blueprints supported by the specific conference server.

confsRequest: Used by a conference control client to query a conferencing system for its scheduled/active conferences.

confsResponse: The "confsResponse" returns the list of the currently activated/scheduled conferences at the server.

confRequest: The "confRequest" is used to create a conference object and/or to request an operation on the conference object as a whole.

confResponse: The "confResponse" indicates the result of the operation on the conference object as a whole.

userRequest: The "userRequest" is used to request an operation on the "user" element in the conference object.

userResponse: The "userResponse" indicates the result of the requested operation on the "user" element in the conference object.

usersRequest This "usersRequest" is used to manipulate the "users" element in the conference object, including parameters such as the "allowed-users-list", "join-handling", etc.

usersResponse: This "usersResponse" indicates the result of the request to manipulate the "users" element in the conference object.

sidebarRequest: This "sidebarRequest" is used to retrieve the information related to a sidebar or to create, change or delete a specific sidebar.

sidebarResponse: This "sidebarResponse" indicates the result of the sidebarRequest.

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 XXXX [NOTE TO IANA/RFC-EDITOR: Please replace XXXX with the RFC number for this specification.]

Registration/Assignment Procedures: New response codes are allocated on a first-come/first-serve basis with specification required.
Registrant Contact: IETF, XCON working group, (xcon@ietf.org), Mary Barnes (mary.barnes@nortel.com).

This section pre-registers the following thirteen initial response codes as described above in Section 4.1:

200: This code indicates that the request was successfully processed.
409: This code indicates that a requested "update" cannot be successfully completed by the server. An example of such 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).
400: This code indicates that the request was badly formed in some fashion.
401: This code indicates that the user was not authorized for the specific operation on the conference object.
403: This code indicates that the specific operation is not valid for the target conference object.
404: This code indicates that the specific conference object was not found.
420: This code is returned in answer to a "userRequest/create" operation, in the case of a third-party invite, when the "confUserID" (contained in the "entity" attribute of the "userInfo" parameter) of the user to be added is unknown.
421: This code is returned in the case of requests in which the "confUserID" of the sender is invalid.
422: This code is returned in response to all requests wishing to access/manipulate a password-protected conference object, when the "conference-password" parameter contained in the request is wrong.
423: This code is returned in response to all requests wishing to access/manipulate a password-protected conference object, when the "conference-password" parameter is missing in the request.
424: This code is returned in response whenever the server wants to authenticate the requestor through the "subject" parameter and such a parameter is not provided in the request.
425: This code indicates that the conferencing system cannot delete the specific conference object because it is a parent for another conference object.
426: This code indicates that the target conference object cannot be changed (e.g., due to policies, roles, privileges, etc.).

510: This code indicates that the request could not be processed within a reasonable time, with the time specific to a conferencing system implementation.
500: This code indicates that the conferencing system experienced some sort of internal error.
501: This code indicates that the specific operation is not implemented on that conferencing system.

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 and Simo Veikkolainen. Special thanks go to Roberta Presta for her invaluable contribution to this document. Roberta has worked on the specification of the CCMP protocol 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. Changes since last Version

NOTE TO THE RFC-Editor: Please remove this section prior to publication as an RFC.

The following summarizes the changes between the WG 03 and the 04:

1. Re-organized document based on feedback from Richard Barnes.
2. Editorial clarifications and nits, including those identified by Richard and Simo Veikkolainen.

The following summarizes the changes between the WG 07 and the 08:

1. Added a new "optionsRequest" message (and related "optionsResponse" message) to the list of CCMP messages.
2. Clarified discussion about notifications management, by clarifying they are out of the scope of the present document, but at the same time providing some pointers to potential ways for implementing them.
3. Clarified discussion about policies in XCON, by clarifying they are out of the scope of the present document, but at the same time providing some pointers to potential ways for implementing them.
4. Corrected minor typos in the schema.

5. Corrected schema definitions which brought to Unique Particle Attribution exceptions.
6. Added the newly defined "optionsRequest" and "optionsResponse" messages to the schema.
7. Misc editorial nits...

15. References

15.1. 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.

[RFC2965] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, October 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.

[I-D.ietf-xcon-common-data-model] Novo, O., Camarillo, G., Morgan, D., and J. Urpalainen, "Conference Information Data Model for Centralized Conferencing (XCON)", draft-ietf-xcon-common-data-model-19 (work in progress), May 2010.

15.2. 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.

[RFC3966] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.

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

[I-D.ietf-xcon-event-package] Camarillo, G., Srinivasan, S., Even, R., and J. Urpalainen, "Conference Event Package Data Format Extension for Centralized Conferencing (XCON)", draft-ietf-xcon-event-package-01 (work in progress), September 2008.

[I-D.ietf-xcon-examples] Barnes, M., Miniero, L., Presta, R., and S. Romano, "Centralized Conferencing Manipulation Protocol (CCMP) Call Flow Examples", draft-ietf-xcon-examples-04 (work in progress), April 2010.

[W3C.REC-soap12-part1-20030624] Hadley, M., Gudgin, M., Nielsen, H., Moreau, J., and N. Mendelsohn, "SOAP Version 1.2 Part 1: Messaging Framework", World Wide Web Consortium FirstEdition REC-soap12-part1-20030624, June 2003, <http://www.w3.org/TR/2003/REC-soap12-part1-20030624>.

[W3C.REC-soap12-part2-20030624] Gudgin, M., Mendelsohn, N., Nielsen, H., Moreau, J., and M. Hadley, "SOAP Version 1.2 Part 2: Adjuncts", World Wide Web Consortium FirstEdition REC-soap12-part2-20030624, June 2003, <http://www.w3.org/TR/2003/REC-soap12-part2-20030624>.

Appendix A. Appendix A: Other protocol models and transports considered for CCMP

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 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 the 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.

Neither of these approaches were considered ideal as SOAP was not considered to be general purpose enough for use in a broad range of operational environments. It is quite awkward to apply a RESTful approach since the 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 the CCMP, the same data in the server is used for both the 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.

A.1. Using SOAP for the CCMP

A remote procedure call (RPC) mechanism for the CCMP could use SOAP (Simple Object Access Protocol[W3C.REC-soap12-part1-20030624][W3C.REC-soap12-part2-20030624]), 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 the 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 [I-D.ietf-xcon-common-data-model].

Following on the CRUD approach, CCMP could follow the well-known REST (REpresentational State Transfer) architectural style [REST]. The 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
Nortel

Email: mary.barnes@nortel.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