XCON Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 11, 2009

M. Barnes
Nortel
C. Boulton
NS-Technologies
S P. Romano
University of Napoli
H. Schulzrinne
Columbia University
March 10, 2009

Centralized Conferencing Manipulation Protocol
draft-ietf-xcon-ccmp-02

Status of this Memo

Copyright Notice

Abstract

The Centralized Conferencing Manipulation Protocol (CCMP) can create, retrieve, change and delete objects describing a centralized conference, such as state and capabilities of the conference, participants, and their roles. The conference information is contained in XML documents and fragments conforming to the centralized conferencing data model schema. CCMP is a state-less client-server protocol based on a request/response model. Conferencing clients send requests to conference servers, which respond to the client with the conference information.

This document also discusses options for using existing notification protocols to inform conference client about the changes in the state of a conference during its entire lifetime.

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 conferencing client and conference control server acting as client and server, respectively. The CCMP uses HTTP [RFC2616] as the protocol to transfer the CCMP requests and responses, which contain the domain-specific XML-encoded data objects defined in the Conference Information Data Model for Centralized Conferencing (XCON Data Model) [I-D.ietf-xcon-common-data-model]. Other protocol models such as the use of a REST (REpresentational State Transfer) architectural style [REST] were considered. However, the CCMP is a request/response protocol with new or updated data relevant to the specific conference object returned in the response. Whereas, a REST approach involves singular/monolithic operations on data, with the response typically indicating either success or failure, rather than providing updated data based on a specific operation, thus, it was not considered a good choice. Details of the use of REST for the CCMP, as well as other protocols considered (e.g., SOAP) are provided in Appendix A.

Section 4 provides an overview of the design of the CCMP, followed by the system architecture in Section 5. Section 6 discusses the primary keys in the conference object carried in the protocol. An overview of the operations associated with each protocol request and response is provided in Section 7. A complete example of the operation of the CCMP, describing a typical call flow associated with conference creation and manipulation, is provided in Section 8. Section 10 provides the XML schema.

2. Conventions

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant implementations.

3. Terminology

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

CRUD: CRUD stands for Create/Read/Update/Delete and indicates a design pattern supporting creating, retrieving, updating and destroying objects.

REST: REpresentational State Transfer (REST) is an architectural style, i.e., a coordinated set of architectural constraints. REST is based on the consideration that a software architecture can often be specified as an appropriate configuration of components, data and connectors, all coordinated through constraining their mutual relationships. Coordination and constraints help achieve a desired set of architectural properties. [REST]

SOAP: Simple Object Access Protocol defined in [W3C.REC-soap12-part1-20030624] and [W3C.REC-soap12-part2-20030624].

4. Protocol Overview

This document specifies the basic operations that can create, retrieve, modify and delete conference-related information in a centralized conference. The core set of objects includes conference blueprints, the conference itself, users, and sidebars.

Each update operation in the protocol model is atomic and either succeeds or fails as a whole. Thus, a server has to first check all 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.

Because multiple clients can modify the same conference objects, clients need to obtain the current object and then update the whole object.

Editor's Note: Do we need locking, using WebDAV or floor control? Otherwise, changes made by user A could get lost when user B wants to modify some other parameter. For example, A changes the subject, B adds the a service URI.

It is likely that implementations and future standardization work will add more conference attributes and parameters. There are three types of extensions. The first and simplest type of extension adds elements to the overall conference description, media descriptions or descriptions of users. The XML namespace mechanism makes such extensions relatively easy, although implementations still have to deal with implementations that may not understand the new namespaces. The CCMP "blueprintsRequest" message allows clients to determine the capabilities of a specific server, reflected by the specific blueprints supported by that server.

A second type of extension replaces the conference, user or media objects with completely new schema definitions, i.e., the namespaces for these objects themselves differ from the basic one defined in this document. As long as the OPTIONS request remains available and keeps to a mutually-understood definition, a compatible client and server will be able to bootstrap themselves into using these new objects.

Finally, it is conceivable that new object types are needed beyond the core conference, user and media objects and their children. These would also be introduced by namespaces and new URIs.

5. 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.

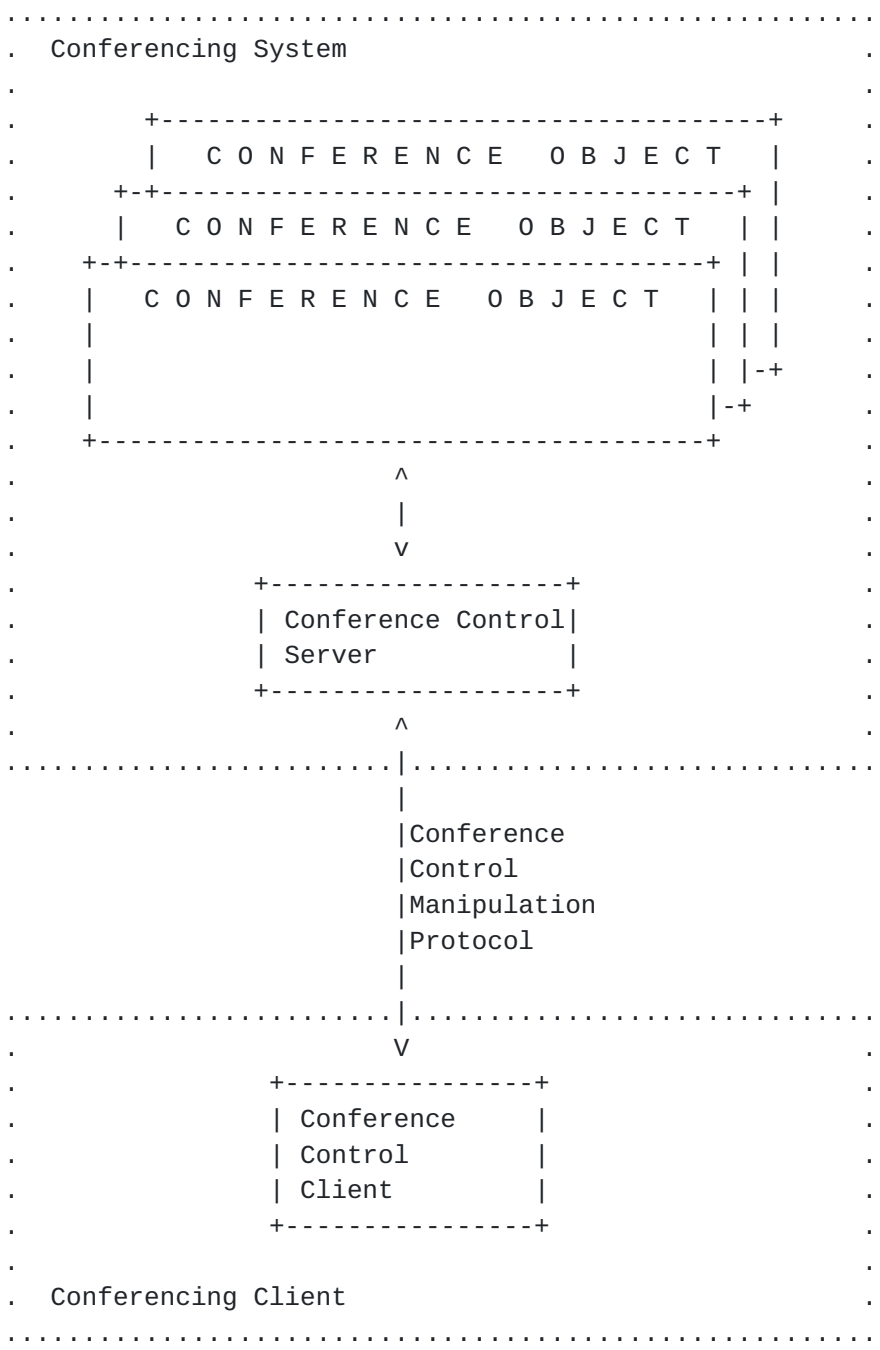


Figure 1: Conference Client Interaction

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.

6. Conference Object and User Identifiers

This section provides an overview of the conference object and conference users which are key protocol elements for creating the CCMP requests and responses. The identifiers used in CCMP for the conference object (XCON-URI) and conference user (XCON-USERID) are introduced in the XCON framework and defined in the XCON data model [I-D.ietf-xcon-common-data-model].

6.1. Conference Object

Conference objects feature a simple dynamic inheritance-and-override mechanism. Conference objects are linked into a tree, where each tree node inherits attributes from its parent node. The roots of these inheritance trees are 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. An object can mark certain of its properties as unalterable, so that they cannot be overridden.

The schema for the conference object is defined in the XCON data model. Conference objects are uniquely identified by the XCON-URI. A client MAY specify a parent element that indicates the parent from which the conference is to inherit values. When creating conferences, the XCON-URI included by the client is only a suggestion. To avoid identifier collisions and to conform to local server policy, the conference control server MAY choose a different identifier.

6.2. Conference Users and Participants

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. 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).

A variety of elements defined in the common <conference-info> element as specified in the XCON data model are used to determine how a specific user expects and is allowed to join a conference as a participant or as a user with specific privileges (e.g., observer). For example, the <method> attribute defines how the caller joins the conference, with a set of defined XML elements, namely <dial-in> for users that are allowed to dial in and <dial-out> for users that the conference focus will be trying to reach. <dial-in> is the default.

If the conference is currently active, dial-out users are contacted immediately; otherwise, they are contacted at the start of the conference. The conference control server assigns a unique Conference User Identifier (XCON-USERID) to each user. The conference control server uses the XCON-USERID to change or delete <user> elements. Depending upon policies and privileges, specific users MAY also manipulate <user> elements.

In many conferences, users can dial in if they know the XCON-URI and an access code shared by all conference participants. In this case, the system is typically not aware of the call signaling URL. Thus, the initial <user> element does not have an entity attribute and the default type of <dial-in> is used to support this type of user. For this case, the server assigns a locally-unique URI, such as a locally-scoped tel URI. The conference control server assigns a unique Conference User Identifier (XCON-USERID) to these users when they dial-in to join the conference. If the user supports the notification event package [I-D.ietf-xcon-event-package], they can receive their XCON-USERID, thus allowing them to also manipulate the <user> attribute, including the entity attribute, in the conference object.

7. Protocol Operations

CCMP is a client-server, XML-based, stateless protocol, which has been specifically conceived to provide users with the necessary means for the creation, retrieval, modification and deletion of conference objects. Conference-related information is encapsulated into CCMP messages in the form of documents or document fragments compliant with the XCON data model representation.

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 active conferences and/or blueprints available at the server.

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. <sidebars-by-value> elements) and external to it (i.e. <sidebars-by-ref> elements),
- <user> elements associated with conference users,
- the list of XCON-URIs related to conferences and blueprints available at the server, for which only retrieval operations are allowed.

7.1. 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 proposed solution for the CCMP 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.

7.2. CCMP protocol messages

CCMP messages are either requests or responses. The general CCMP request message is defined in Section 7.2.1. The general CCMP response message is defined in Section 7.2.2. The details of the specific message type which is carried in the CCMP request and response messages are described in Section 7.2.3.

7.2.1. CCMP Request Message

A CCMP request message is comprised of the following parameters:

confUserId: A mandatory parameter containing the XCON-URI of the client. This parameter is REQUIRED by the conferencing server for system specific Authorization, Authentication and Accounting (AAA) procedures.

confObjId: an optional parameter containing, whenever necessary, the XCON-URI of the target conference object;

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, the details of which are provided in Section 7.2.3

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

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

   <xs:complexType abstract="true"
        name="ccmp-request-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:sequence>
   </xs:complexType>
```

Figure 2: Structure of CCMP Request messages

7.2.2. CCMP Response Message

A CCMP response message is comprised of the following parameters:

confUserId: A mandatory parameter containing the XCON-URI of the client which issued the request

confObjId: An optional parameter containing the XCON-URI of the target conference object

responseCode: A mandatory parameter containing the response code associated with the request, chosen among the codes listed in Section 7.2.2.1

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), the details of which are provided in Section 7.2.3

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

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

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

Figure 3: Structure of CCMP Response message

7.2.2.1. CCMP Response Codes

All CCMP response messages MUST include a "responseCode". The following summarizes the CCMP response codes:

success: Successful completion of the requested operation.

modified: Successful completion of the requested operation, with partial data returned in the confObjID having been modified from the data included in the confObjID included request, either for a "create" or a "change" operation

badRequest: Syntactically malformed request

objectNotFound: Target object missing at the server

unauthorized: User not allowed to perform the required operation

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

forbiddenDeleteParent: 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

forbiddenChangeProtected: 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)

requestTimeout: The time required to serve the request has exceeded the envisaged service threshold

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

notImplemented: Operation envisaged in the protocol, but not implemented in the contacted server.

7.2.3. Detailed CCMP Messages

Based on the request and response message structures described in Section 7.2.1 and Section 7.2.2, the following summarizes the specialized CCMP request/response types described in this document:

1. blueprintsRequest/blueprintsResponse

2. confsRequest/confsResponse

3. blueprintRequest/blueprintResponse

4. confRequest/confResponse

5. usersRequest/usersResponse

6. userRequest/userResponse

7. sidebarsByValRequest/sidebarsByValResponse

8. sidebarsByRefRequest/sidebarsByRefResponse

9. sidebarByValRequest/sidebarByValResponse

10. sidebarByRefRequest/sidebarByRefResponse

These CCMP request/response pairs use the fundamental CCMP operations as defined in Section 7 to manipulate the conference data. Table 1 summarizes the CCMP operations and corresponding actions that are valid for a specific CCMP request type, noting that neither the blueprintsRequest/blueprints/Response or 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
blueprintsRequest	Get list of blueprints.	N/A	N/A	N/A
blueprintRequest	Get blueprint.	N/A*	N/A*	N/A*
confsRequest	Get list of conferences (active, etc.)	N/A	N/A	N/A
confRequest	Gets conference object or blueprint.	Creates conference object	Changes conference object	Deletes conference Object as a whole.
usersRequest	Gets a specific users element.	N/A	Adds or modifies the specified users element.	N/A
userRequest	Gets a specific user element.	Creates XCON-UserID.	Adds or modifies the specified user element.	Deletes a user element as a whole.

sidebarsByVal Request	Gets sidebars-by -val element	N/A	N/A	N/A	
sidebarsByRef Request	Gets sidebars-by -ref element	N/A	N/A	N/A	
sidebarByValRequest	Gets a sidebar element.	N/A	Adds or modifies a sidebar.	Removes/dell etes the entire sidebar.	
sidebarByRefRequest	Gets a sidebar element.	N/A	Adds or modifies a sidebar.	Removes/dell etes the entire sidebar.	

Table 1: Request Type Operation Specific Processing

The following additional parameters are included in the specialized CCMP request/response messages detailed in the subsequent sections:

operation: An optional parameter for each CCMP request/response message. This parameter is REQUIRED in all messages except for the "blueprintRequest", "blueprintResponse", "confsRequest" and "confsResponse" messages.

blueprintInfo: An optional parameter used for the blueprintResponse message. It is of type "conference-type" as defined in the XCON data model and contains the data of the conference object representing the blueprint in the conference server. This parameter SHOULD not be included in any other response type messages and SHOULD only be included in a "create", "update" or "delete" operation for a blueprintRequest message in special cases where a user has special privileges such as an administrator.

blueprintsInfo: An optional parameter used for the blueprintsResponse message. It contains a list of elements of type "blueprintInfo". This parameter SHOULD not be included in any other response type messages and SHOULD only be included in a "create", "update" or "delete" operation for a blueprintRequest message in special cases where a user has special privileges such as an administrator.

confsInfo: An optional parameter used for the confsResponse message. It contains a list of XCON-URIs. This parameter SHOULD not be included in any other response type messages and SHOULD only be included in a "create", "update" or "delete" operation for a blueprintRequest message in special cases where a user has special privileges such as an administrator.

usersInfo: An OPTIONAL parameter that MAY be included in a usersRequest and usersReponse message, depending upon the operation. The 'usersInfo' parameter carries an object compliant with the <users> field of the XCON data model.

7.2.3.1. blueprintsRequest and blueprintsResponse messages

A 'blueprintsRequest' (Figure 4) 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 7.2.3.3. A 'blueprintsRequest' message REQUIRES no additional parameters beyond those specified for the basic CCMP request message. The associated 'blueprintsResponse' message SHOULD contain, as shown in Figure 4, a 'blueprintsInfo' parameter containing the above mentioned XCON-URI list. The 'confObjId' parameter is NOT REQUIRED in the request or response for this transaction.

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

Figure 4: Structure of the blueprintsRequest and blueprintsResponse messages

7.2.3.2. confsRequest and confsResponse messages

A 'confsRequest' message is used to retrieve, from the server, the list of XCON-URIs associated with active and registered conferences A 'confsRequest' message REQUIRES no additional parameters beyond those specified for the basic CCMP request message. The associated 'confsResponse' message SHOULD contain the list of XCON-URIs in the 'confsInfo' parameter. The 'confObjId' parameter is NOT REQUIRED for this transaction. A user, upon receipt of the response message, can interact with the available conference objects through further CCMP messages. 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 SHOULD NOT be

included in a 'confsRequest' message.

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

Figure 5: Structure of the confsRequest and confsResponse messages

7.2.3.3. blueprintRequest and blueprintResponse messages

Through a 'blueprintRequest', a client can manipulate the conference object associated with a specified blueprint. The request MUST include an 'operation' parameter and a 'confObjId' parameter. Only the "retrieve" 'operation' SHOULD should be included in a 'blueprintRequest message. 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). The 'confObjId' parameter contains the XCON-URI of the blueprint, which might have been previously retrieved

through a 'blueprintsRequest' message.

In the case of responseCode of "success" for a 'retrieve' operation, the 'blueprintInfo' parameter SHOULD be included in the 'blueprintResponse' message. Inside responses, the 'blueprintInfo' parameter carries the conference document associated with the blueprint specified in the request.

```
   <!--  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="operation" type="operationType"
                    minOccurs="1" maxOccurs="1" />
        <xs:element name="blueprintInfo"
                    type="info:conference-type" minOccurs="0"/>
     </xs:sequence>
   </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"/>
      </xs:sequence>
    </xs:complexType>
```

Figure 6: Structure of the blueprintRequest and blueprintResponse

messages

7.2.3.4. confRequest and confResponse messages

With a 'confRequest' message, CCMP clients can manipulate conference objects associated with either active or registered conferences (blueprints or reservations). The request MUST include an 'operation' parameter. Depending upon the type of 'operation' a 'confObjId' parameter and/or 'confInfo' parameter MAY be included. The 'confObjId' parameter contains the XCON-URI of the specific active or registered conference. The 'confInfo' parameter contains the conference data that is the target of the 'confRequest' - i.e. the <conference-info> document (compliant with the XCON data model structure).

To create a new conference through a 'confRequest' message, two approaches can be embraced:

1. Creation through explicit cloning: the 'confObjId' parameter MUST contain the XCON-URI of the blueprint to be cloned, while the 'confInfo' parameter SHOULD NOT be included in the request;

2. Creation through implicit cloning (also known as "direct creation"): the 'confObjId' parameter SHOULD NOT be included in the request, whereas the 'confInfo' parameter describing the conference to be created MUST be included in the request.

In both cases, a successful completion of the request carries back a responseCode of 'success' and SHOULD contain, in the 'confObjId' parameter, the XCON-URI of the created conference. In addition, the 'confInfo' parameter transporting the created conference document SHOULD be included. Obviously, the newly created object can be manipulated by the client through subsequent 'update' operations.

In the case of 'retrieve' or 'delete' operations, the 'confObjId' representing the XCON-URI of the target conference MUST be included and the 'confInfo' SHOULD NOT be included in the request. Inside the response to a 'retrieve' request, in case of responseCode of 'success', the 'confInfo' containing a description of the target conference object MUST be included. On the other hand, a response to a 'delete' operation SHOULD NOT include the 'confInfo' parameter.

In case of 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'. In the case of a responseCode of success, no additional information is REQUIRED in the 'confResponse'message. For a successful response, the conference

server should consider as unchanged all parts of the referenced conference document. However, if the target conference object has not been modified exactly as required by the client the responsecode MUST be set to 'modified' and the 'confInfo' parameter MUST contain the entire conference document to which the required changes have been (at least partially) applied.

The schema for the confRequest/confResponse pair is shown in Figure 7.

```
   <!-- 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="operation" type="operationType"
                    minOccurs="1" maxOccurs="1" />
        <xs:element name="confInfo"
                    type="info:conference-type" minOccurs="0"/>
     </xs:sequence>
   </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"/>
     </xs:sequence>
   </xs:complexType>
```

Figure 7: Structure of the confRequest and confResponse messages

The following provides an example of the 'confInfo' parameter required to change the title of a conference:

```
<conf-info entity="123">
        <conference-description>
                <display-text>New conference title</display-text>
        </conference-description>
</conf-info>
```

Figure 8: Updating a conference object: modifying the title of a conference

Similarly, to remove the title of an existing conference, an 'update' operation carrying the following 'confInfo' parameter would do the job.

```
<conf-info entity="123">
        <conference-description>
                <display-text/>
        </conference-description>
</conf-info>
```

Figure 9: Updating a conference object: removing the title of a conference

7.2.3.5. usersRequest and usersResponse messages

Through a usersRequest message the CCMP client manipulates the <users> element of the conference document associated with the conference identified by the 'confObjId' parameter. Inside the <users> element, along with the list of conference users, there is information that the client may be interested in controlling, such as the lists of 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. Besides the usual 'operation' parameter, a 'usersInfo' parameter MAY also be included depending upon the operation. The 'usersInfo' parameter carries an object compliant with the <users> field of the XCON data model.

An 'operation' parameter MUST be included in a "usersRequest"

message. Two operations are allowed in a "usersRequest" message:

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

2. update: In this case, the 'usersInfo' parameter MUST contain the modifications to be applied to the referred <users> element. If the responseCode is 'success', then the 'usersInfo' parameter SHOULD NOT be returned. If the responseCode is 'modified', the 'usersInfo' parameter MUST be included in the response. The 'usersInfo' reflects to the client the (partial) modifications that have been applied.

Operations of 'create' and 'delete' make little sense in the case of a usersRequest and SHOULD NOT be considered by the server, which means that a responseCode of 'forbidden' SHOULD 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="operation" type="operationType"
                           minOccurs="1" maxOccurs="1" />
                   <xs:element name="usersInfo"
                   type="info:users-type" minOccurs="0" />
           </xs:sequence>
   </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"/>
     </xs:sequence>
   </xs:complexType>
```

Figure 10: Structure of the usersRequest and usersResponse messages

7.2.3.6. userRequest and userResponse messages

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 MAY be used to either create, or modify, or delete information about a user. A "userRequest" MUST include the 'operation' parameter, and MAY include a 'userInfo' parameter containing the detailed user's information.

Conference users can be created in a number of different ways. Let us first consider the case of a user who wants to enter a conference (i.e. to add himself to the conference). In such a case, the 'userInfo' parameter in the request (which MUST have an 'operation' value set to "create") SHOULD contain a <user> element (compliant with the XCON data model) having its 'entity' attribute set to a value which represents the XCON-USERID of the user in question.

A different situation is one in which the CCMP client acts on behalf of a third user, whose XCON-USERID is known. In this case, the <user> element SHOULD contain an 'entity' attribute whose value is set to the XCON-USERID of the user in question. As a final case, if the CCMP client is not aware of the XCON-USERID of the user to be inserted, the key attribute (i.e. 'entity') SHOULD NOT be included in the request: the XCON-USERID generated by the conference server for such a user MUST be returned to the client as the value of the 'entity' attribute in the 'userInfo' parameter of the response if the responseCode is "success". The last case also applies to a CCMP client that obtains a new user profile in the context of a conference.

```
   <!-- 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="operation" type="operationType"
                           minOccurs="1" maxOccurs="1" />
                   <xs:element name="userInfo"
                   type="info:user-type" minOccurs="0" />
           </xs:sequence>
   </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"/>
     </xs:sequence>
   </xs:complexType>
```

Figure 11: Structure of the userRequest and userResponse messages

7.2.3.7. sidebarsByValRequest and sidebarsByValResponse messages

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 request MUST include an 'operation' of "retrieve" and a 'confObjId'. A "sidebarsByValResponse" MUST contain a 'sidebarsByValInfo' parameter reporting the desired <sidebars-by-val> element. The 'sidebarsByValInfo' parameter contains the identifiers of the sidebars derived from the main conference. For the creation and manipulation of sidebars, a different message has been envisaged, namely "sidebarByValRequest", which is described in Section 7.2.3.8

```
<!-- sidebarsByValRequest -->

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

<!-- sidebarsByValRequestType -->

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

<xs:complexType name="sidebarsByValRequestType">
  <xs:sequence>
     <xs:element name="operation" type="operationType"
                 minOccurs="1" maxOccurs="1" />
     <xs:element name="sidebarsByValInfo"
                 type="info:sidebars-by-val-type" minOccurs="0"/>
  </xs:sequence>
</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"/>
     </xs:sequence>
   </xs:complexType>
```

Figure 12: Structure of the sidebarsByValRequest and sidebarsByValResponse messages

7.2.3.8. sidebarByValRequest and sidebarByValResponse messages

A "sidebarByValRequest" message MUST contain the 'operation' parameter which discriminates among creation, modification and deletion of a specific sidebar. The 'sidebarByValInfo' parameter, in turn, contains the description (in an XCON data model compliant fashion) of the sidebar itself. The 'confObjId' parameter of such messages MUST contain the XCON-URI of the main conference which the sidebar belongs to. The XCON-URI of the sidebar is contained in the 'entity' attribute of the above mentioned 'sidebarByValInfo' document. In case of creation, the 'sidebarByValInfo' SHOULD NOT be included in the request, since, as envisaged in the XCON framework ([RFC5239]), sidebars are always created by cloning the main conference. The 'sidebarByValInfo' parameter MUST be included in a successful response. The 'sidebarByValInfo' represents the created sidebar, whose URI appears in the 'entity' attribute.

```
   <!-- 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="operation" type="operationType"
                    minOccurs="1" maxOccurs="1" />
       <xs:element name="sidebarByValInfo"
                    type="info:conference-type" minOccurs="0"/>
     </xs:sequence>
    </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"/>
      </xs:sequence>
    </xs:complexType>
```

Figure 13: Structure of the sidebarByValRequest and sidebarByValResponse messages

7.2.3.9. sidebarsByRefRequest and sidebarsByRefResponse messages

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 'confObjID' parameter MUST be included in the request. An operation of 'retrieve' MUST also be included in the request. In the case of a responseCode of success, 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 by exploiting the ad-hoc defined "sidebarByRef" request, described in Section 7.2.3.10.

```
<!-- 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="operation" type="operationType"
                 minOccurs="1" maxOccurs="1" />
     <xs:element name="sidebarsByRefInfo"
                 type="info:uris-type" minOccurs="0"/>
  </xs:sequence>
</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"/>
  </xs:sequence>
</xs:complexType>
```

Figure 14: Structure of the sidebarsByRefRequest and sidebarsByRefResponse messages

7.2.3.10. sidebarByRefRequest and sidebarByRefResponse messages

A "sidebarByRefRequest" message along with the REQUIRED 'operation' parameter, MAY contain a 'sidebarByRefInfo' parameter describing the conference object (compliant with the XCON data model) associated with the sidebar. In case of 'retrieve', 'delete', 'update' and 'create' operations, the 'confObjId' parameter representing the XCON-URI of the target sidebar MUST be included. The 'sidebarByRefInfo' parameter is NOT REQUIRED in the first two cases ('retrieve' and 'delete'), whereas in the case of an 'update' the 'sidebarByRefInfo' parameter MUST contain the changes to be applied to the referenced sidebar. The 'sidebarByRefInfo' MUST NOT be included for a 'create' operation since, as already stated, sidebar creation is by default achieved by cloning the main conference.

```
<!-- 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="operation" type="operationType"
                    minOccurs="1" maxOccurs="1" />
        <xs:element name="sidebarByRefInfo"
                    type="info:conference-type" minOccurs="0"/>
     </xs:sequence>
   </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"/>
     </xs:sequence>
   </xs:complexType>
```

Figure 15: Structure of the sidebarByRefRequest and sidebarByRefResponse messages

8. A complete example of the CCMP in action

In this section a typical 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 is "Alice" and appears in all requests. The sequence of operations is as follows:

1. Alice retrieves from the server the list of available blueprints (Section 8.1);

2. Alice asks for detailed information about a specific blueprint (Section 8.2);

3. Alice decides to create a new conference by cloning the retrieved blueprint (Section 8.3);

4. Alice modifies information (e.g. XCON-URI, name, description) associated with the newly created blueprint (Section 8.4);

5. Alice specifies a list of users to be contacted when the conference is activated (Section 8.5);

6. Alice joins the conference (Section 8.6);

7. Alice lets a new user (whose 'confUserId' is "Ciccio") join the conference (Section 8.7).

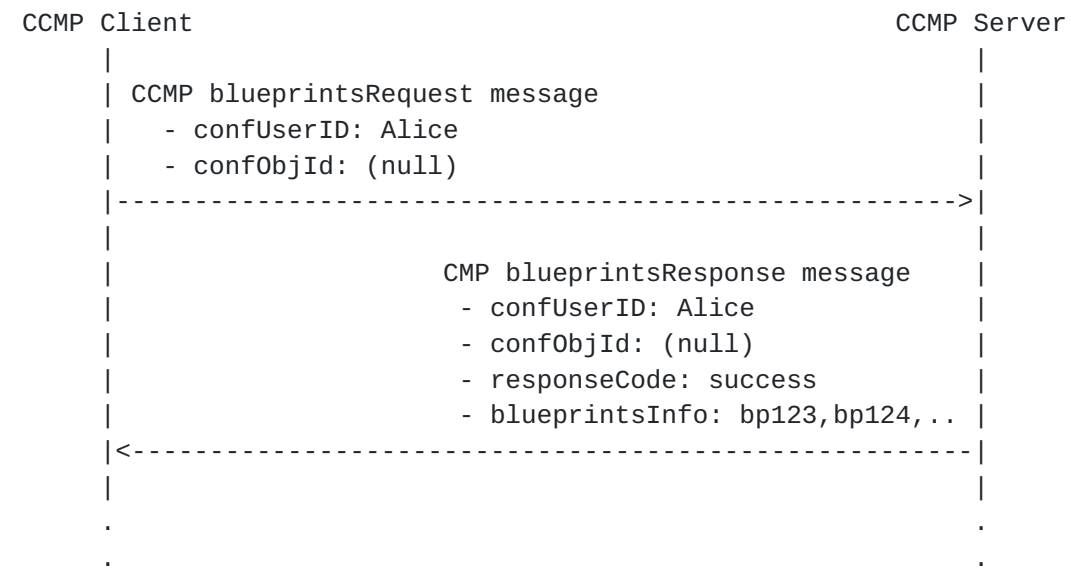
Note, the examples do not include any details beyond the basic operation.

In the following sections we deal with each of the above mentioned actions separately.

8.1. 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>Alice</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>Alice</confUserID>
    <ccmp:response-code>success</ccmp:response-code>
     <ccmp:blueprintsResponse>
      <blueprintsInfo>
       <info:entry>
        <info:uri>xcon:AudioRoom@meetecho.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@meetecho.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@meetecho.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@meetecho.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@meetecho.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 16: Getting blueprints from the server

8.2. 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@meetecho.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">
        <confObjID>xcon:AudioRoom@meetecho.com</confObjID>
        <confUserID>Alice</confUserID>
        <ccmp:blueprintRequest>
            <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">
      <confObjID>xcon:AudioRoom@meetecho.com</confObjID>
      <confUserID>Alice</confUserID>
      <ccmp:response-code>success</ccmp:response-code>
      <ccmp:blueprintResponse>
        <blueprintInfo entity="AudioRoom">
          <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 17: Getting info about a specific blueprint

8.3. 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@meetecho.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@meetecho.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@meetecho.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">
       <confObjID>xcon:AudioRoom@meetecho.com</confObjID>
       <confUserID>Alice</confUserID>
       <ccmp:confRequest>
           <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">
     <confObjID>xcon:8977794@meetecho.com</confObjID>
     <confUserID>Alice</confUserID>
     <ccmp:response-code>success</ccmp:response-code>
     <ccmp:confResponse>
          <confInfo entity="xcon:8977794@meetecho.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@meetecho.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>
```

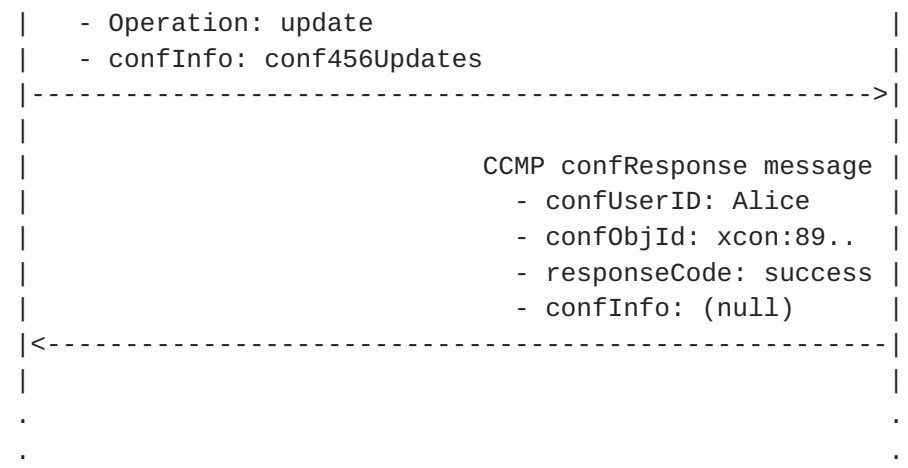
Figure 18: Creating a new conference by cloning a blueprint

8.4. 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 'success', which acknowledges the modifications requested by the client.

Alice updates information about the conference she just created:

```
  CCMP Client                                          CCMP Server
       |                                                    |
       | CCMP confRequest message                           |
       |   - confUserID: Alice                              |
       |   - confObjId: xcon:8977794@meetecho.com           |
```



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">
    <confObjID>xcon:8977794@meetecho.com</confObjID>
    <confUserID>Alice</confUserID>
    <ccmp:confRequest>
       <operation>update</operation>
         <confInfo entity="xcon:8977794@meetecho.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">
```

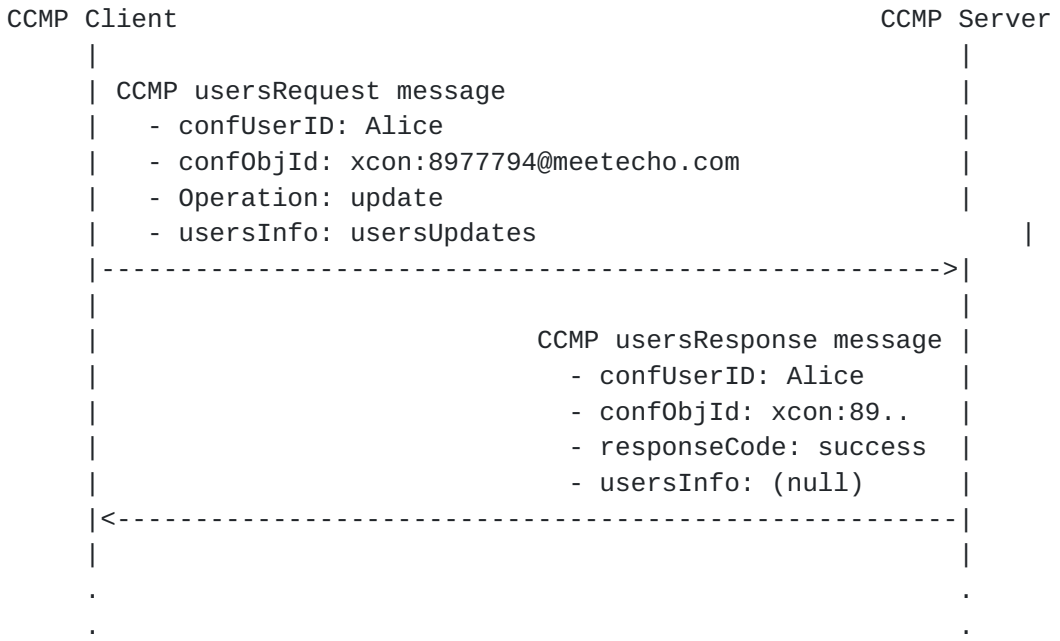
```
    <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:type="ccmp:ccmp-conf-response-message-type">
       <confObjID>xcon:8977794@meetecho.com</confObjID>
       <confUserID>Alice</confUserID>
       <ccmp:response-code>success</ccmp:response-code>
       <ccmp:confResponse/>
    </ccmpResponse>
  </ccmp:ccmpResponse>
```

Figure 19: Updating conference information

8.5. 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">
        <confObjID>xcon:8977794@meetecho.com</confObjID>
        <confUserID>Alice</confUserID>
        <ccmp:usersRequest>
          <operation>update</operation>
            <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:

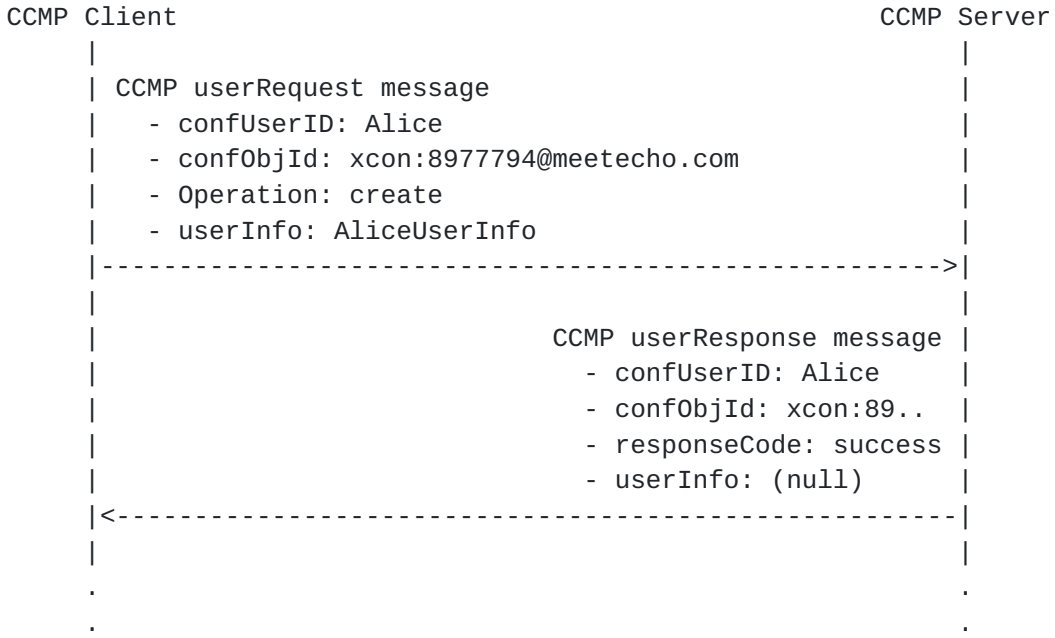
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
                  xmlns:info="urn:ietf:params:xml:ns:conference-info"
                        xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp">
    <ccmpResponse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

Figure 20: Updating the list of allowed users for the conference 'xcon:8977794@meetecho.com'

8.6. 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-confUser-request-message-type">
        <confObjID>xcon:8977794@meetecho.com</confObjID>
        <confUserID>Alice</confUserID>
```

```
          <ccmp:confUserRequest>
              <operation>create</operation>
              <userInfo entity="Alice">
                  <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:confUserRequest>
      </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-confUser-response-message-type">
        <confObjID>xcon:8977794@meetecho.com</confObjID>
        <confUserID>Alice</confUserID>
        <ccmp:response-code>success</ccmp:response-code>
        <ccmp:confUserResponse/>
    </ccmpResponse>
</ccmp:ccmpResponse>
```

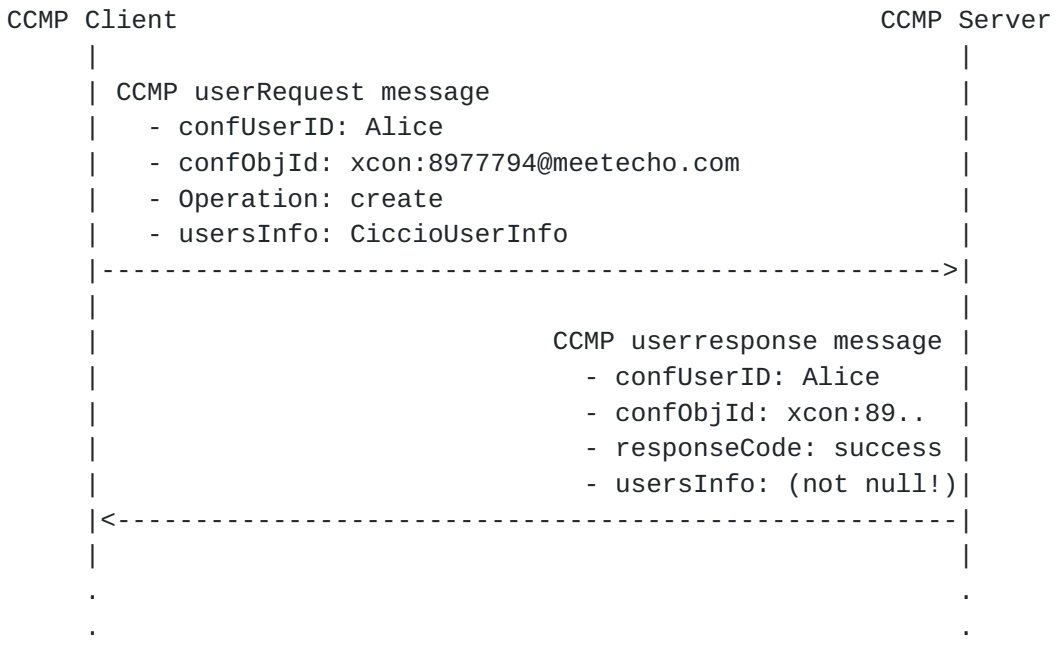
Figure 21: Alice joins the conference through the CCMP

8.7. 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 user to the 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 his current end-point. The picture below shows the transaction. Notice how the 'confUserId' parameter in the request is Alice's id,

whereas the <userInfo> element has no "entity" attribute and contains information about a different user, thus indicating that the request issued by the client is a third-party one. This is also reflected in the response coming from the server, which this time contains a non-void <userInfo> element, whose "entity" attribute has been set by the server to the value of the newly created conference user id.

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-confUser-request-message-type">
        <confObjID>xcon:8977794@meetecho.com</confObjID>
        <confUserID>Alice</confUserID>
        <ccmp:confUserRequest>
            <operation>create</operation>
            <userInfo>
                <info:associated-aors>
                    <info:entry>
                        <info:uri>
```

```
                        mailto:ciccio@pernacchio.com
                    </info:uri>
                    <info:display-text>email</info:display-text>
                </info:entry>
            </info:associated-aors>
            <info:endpoint entity="sip:ciccio@pernacchio.com"/>
        </userInfo>
    </ccmp:confUserRequest>
</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-confUser-response-message-type">
        <confObjID>8977794</confObjID>
        <confUserID>Alice</confUserID>
        <ccmp:response-code>success</ccmp:response-code>
        <ccmp:confUserResponse>
            <confUserInfo entity="bn90ujbkj">
                <info:associated-aors>
                    <info:entry>
                        <info:uri>
                           mailto:ciccio@pernacchio.com
                        </info:uri>
                        <info:display-text>email</info:display-text>
                    </info:entry>
                </info:associated-aors>
                <info:endpoint entity="sip:ciccio@pernacchio.com"/>
            </confUserInfo>
        </ccmp:confUserResponse>
    </ccmpResponse>
</ccmp:ccmpResponse>
```

Figure 22: Alice adds a new user to the conference through the CCMP

9. 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 23 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 23: 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.

10. 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="confUserID" type="xs:string"
          minOccurs="1" maxOccurs="1" />
       <xs:element name="confObjID" type="xs:string"
          minOccurs="0" maxOccurs="1" />
     </xs:sequence>
  </xs:complexType>

    <!-- blueprintsRequest -->
  <xs:complexType name="ccmp-blueprints-request-message-type">
      <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type"/>
      </xs:complexContent>
  </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="operation" type="operationType"
                        minOccurs="1" maxOccurs="1" />
            <xs:element name="blueprintInfo"
                   type="info:conference-type" minOccurs="0"/>
        </xs:sequence>
  </xs:complexType>

  <!-- confsRequest -->
  <xs:complexType name="ccmp-confs-request-message-type">
      <xs:complexContent>
        <xs:extension base="tns:ccmp-request-message-type"/>
      </xs:complexContent>
  </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="operation" type="operationType"
                        minOccurs="1" maxOccurs="1" />
            <xs:element name="confInfo"
                type="info:conference-type" minOccurs="0"/>
        </xs:sequence>
       </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="operation" type="operationType"
                                          minOccurs="1" maxOccurs="1" />
                   <xs:element name="usersInfo"
              type="info:users-type" minOccurs="0" />
            </xs:sequence>
          </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="operation"
          type="operationType"
                           minOccurs="1" maxOccurs="1" />
            <xs:element name="userInfo"
          type="info:user-type"
          minOccurs="0" />
          </xs:sequence>
  </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="operation" type="operationType"
                      minOccurs="1" maxOccurs="1" />
          <xs:element name="sidebarsByValInfo"
              type="info:sidebars-by-val-type" minOccurs="0"/>
```

```
      </xs:sequence>
  </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="operation" type="operationType"
                        minOccurs="1" maxOccurs="1" />
            <xs:element name="sidebarsByRefInfo"
               type="info:uris-type" minOccurs="0"/>
         </xs:sequence>
       </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="operation" type="operationType"
             minOccurs="1" maxOccurs="1" />
         <xs:element name="sidebarByValInfo"
             type="info:conference-type" minOccurs="0"/>
      </xs:sequence>
     </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="operation" type="operationType"
                minOccurs="1" maxOccurs="1" />
            <xs:element name="sidebarByRefInfo"
                type="info:conference-type" minOccurs="0"/>
         </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="0" maxOccurs="1" />
         <xs:element name="confObjID" type="xs:string"
           minOccurs="0" maxOccurs="1" />
         <xs:element ref="response-code" minOccurs="1"
           maxOccurs="1" />
      </xs:sequence>
  </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:sequence>
     </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"/>
         </xs:sequence>
     </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:sequence>
    </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"/>
        </xs:sequence>
     </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"/>
        </xs:sequence>
    </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"/>
        </xs:sequence>
     </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"/>
         </xs:sequence>
     </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"/>
            </xs:sequence>
  </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"/>
         </xs:sequence>
     </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"/>
         </xs:sequence>
     </xs:complexType>

          <!-- response-code -->

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

          <xs:simpleType name="response-codeType">
            <xs:restriction base="xs:token">
              <xs:enumeration value="success"/>
              <xs:enumeration value="pending"/>
              <xs:enumeration value="modified"/>
              <xs:enumeration value="badRequest"/>
              <xs:enumeration value="unauthorized"/>
              <xs:enumeration value="forbidden"/>
              <xs:enumeration value="objectNotFound"/>
              <xs:enumeration value="forbiddenDeleteParent"/>
```

```
            <xs:enumeration value="forbiddenChangeProtected"/>
            <xs:enumeration value="requestTimeout"/>
            <xs:enumeration value="serverInternalError"/>
            <xs:enumeration value="notImplemented"/>
          </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>
    </xs:schema>
```

Figure 24

11. Managing notifications

This section is still "Under Construction" and currently contains some views on handling notifications.

One proposal is to stick with SIP notification. Another alternative, which is commonly done in other web-based systems, is a "call back", i.e., the CCMP client provides the conference server with an HTTP URL which is invoked when a change occurs. This is apparently how most credit card shopping cards work, having implemented one. This works well for our scenario since a CCMP "client" is likely to be a web server that provides the graphical HTML user interface and uses CCMP as the backend to talk to the conference server. In that particular case, there doesn't seem to be a problem of having both models. PC-based clients behind NATs would provide a SIP event URI, web servers would probably find the HTTP model much easier to program with.

Another option being considered is BOSH (http://xmpp.org/extensions/xep-0124.html), which is basically an extension to XMPP designed with the following aim: "...a transport protocol that emulates a bidirectional stream between two entities (such as a client and a server) by efficiently using multiple synchronous HTTP request/response pairs without requiring the use of polling or asynchronous chunking."

A final consideration (under discussion only) is basic XMPP.

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 10 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. 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: This content type provides a basis for a protocol

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 Location 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 Location Server Application Protocol Tag for HELD

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 7 and defined in the XML schema in Section 10. 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:

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

blueprintsResponse: The optionsResponse 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 7:

success: This code indicates that the request was successfully processed.

modified: This code indicates that the object was created, but may differ from the request.

badRequest: This code indicates that the request was badly formed in some fashion.

unauthorized: This code indicates that the user was not authorized for the specific operation on the conference object.

forbidden: This code indicates that the specific operation is not valid for the target conference object.

objectNotFound: This code indicates that the specific conference object was not found.

operationNotAllowed: This code indicates that the specific operation is not allowed for the target conference object (e.g.., due to policies, etc.)

deleteFailedParent: This code indicates that the conferencing system cannot delete the specific conference object because it is a parent for another conference object.

changeFailedProtected: This code indicates that the target conference object cannot be changed (e.g., due to policies, roles, privileges, etc.).

requestTimeout: This code indicates that the request could not be processed within a reasonable time, with the time specific to a conferencing system implementation.

serverInternalError: This code indicates that the conferencing system experienced some sort of internal error.

notImplemented: This code indicates that the specific operation is not implemented on that conferencing system.

13. Security Considerations

Access to conference control functionality needs to be tightly controlled to keep attackers from disrupting conferences, adding themselves to conferences or engaging in theft of services. In the case of a RESTful implementation of the CCMP, implementors need to deploy standard HTTP authentication and authorization mechanisms. Since conference information may contain secrets such as participant lists and dial-in codes, all conference control information SHOULD be carried over TLS (HTTPS).

14. Acknowledgments

The authors appreciate the feedback provided by Dave Morgan, Pierre Tane, Lorenzo Miniero and Tobia Castaldi. 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 8 have been extracted.

15. 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 01 and the 02:

1. Changed the basic approach from REST to HTTP as a transport. This impacted most of the document - i.e., a major rewrite - 02 is closer to 00 than the 01.

2. Added full example based on prototype.

The following summarizes the changes between the WG 00 and the 01:

1. Changed the basic approach from using SOAP to REST - the fundamentals are the same in terms of schema, basic operations. This impacted most sections, in particular introduction and motivation.

2. Added new request types - blueprintsRequest, blueprintRequest and confsRequest. The first replaces the optionsRequest and the latter allows the client to get a list of all active conferences.

3. Merged all requests into the basic operations table. Added summary of RESTful examples (referenced by the basic operations table.

4. Added examples showing RESTful approach - i.e., HTTP methods for message exchange.

5. Removed requestID from the schema (it should be handle by the transport - e.g., HTTP). Updated schema (based on current prototype - it still needs another revision.

6. Added placeholders for Notifications and Role Based Access Control.

7. Added some text for discovery using DNS (including IANA registrations)

8. Updated References: updated XCON FW RFC, SOAP/W3C moved to informational section.

16. References

16.1. 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.

[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.

[I-D.ietf-xcon-common-data-model] Novo, O., Camarillo, G., Morgan, D., Even, R., and J. Urpalainen, "Conference Information Data Model for Centralized Conferencing (XCON)", draft-ietf-xcon-common-data-model-12 (work in progress), October 2008.

16.2. 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.

[RFC3880] Lennox, J., Wu, X., and H. Schulzrinne, "Call Processing Language (CPL): A Language for User Control of Internet Telephony Services", RFC 3880, October 2004.

[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.

[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.royer-calsch-xcal]
Royer, D., "iCalendar in XML Format (xCal-Basic)", draft-royer-calsch-xcal-03 (work in progress), October 2005.

[W3C.REC-soap12-part1-20030624]
Gudgin, M., Hadley, M., Mendelsohn, N., Nielsen, H., and J. Moreau, "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]
Mendelsohn, N., Nielsen, H., Hadley, M., Gudgin, M., and J. Moreau, "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
2201 Lakeside Blvd
Richardson, TX

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