XCON Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2009

M. Barnes
Nortel
C. Boulton
Avaya
S P. Romano
University of Napoli
H. Schulzrinne
Columbia University

November 3, 2008

Centralized Conferencing Manipulation Protocol
draft-ietf-xcon-ccmp-01

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 4
2. Conventions 5
3. Terminology 5
4. Rationale and Motivation 5
5. System Architecture 7
6. Conference Object and User Identifiers 9
6.1. Conference Object 9
6.2. Conference Users and Participants 9
7. Protocol Operations 10
7.1. Retrieve 10
7.2. Create 11
7.3. Change 12
7.4. Delete 12
8. Protocol Operations on Conference Objects 13
8.1. Locating a Conference Control Server 14
8.2. Constructing a CCMP Request 15
8.2.1. blueprintsRequest 16
8.2.2. confsRequest 16
8.2.3. Operations Requests 16
8.3. Handling a CCMP Response 18
8.3.1. blueprintsResponse 20
8.3.2. confsResponse 20
8.3.3. Operation Responses 21
9. Managing sidebars 25
9.1. Sidebars by value 25
9.2. Sidebars by reference 26
10. Protocol Parameters 26
10.1. Operation Parameter 26
10.2. ConfObjID Parameter 26
10.3. ConfUserID Parameter 27
10.4. ResponseCode Parameter 27
10.5. Blueprints Parameter 28
10.6. Conference-info Parameter 28
10.7. User Parameter 29
10.8. Users Parameter 29
10.9. Sidebar Parameters 29
11. Examples 30
11.1. HTTP methods for realizing a RESTful CCMP 30

```
     11.2. CCMP Detailed Message Body Examples . . . . . . . . . . . . 33
       11.2.1. Creating a New Conference  . . . . . . . . . . . . . . . 33
       11.2.2. Creating a New Conference User . . . . . . . . . . . . . 36
       11.2.3. Adding a User to a Conference  . . . . . . . . . . . . . 36
   12. XML Schema . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
   13. Managing notifications . . . . . . . . . . . . . . . . . . . . . 45
   14. Role based access control  . . . . . . . . . . . . . . . . . . . 46
   15. IANA Considerations  . . . . . . . . . . . . . . . . . . . . . . 46
     15.1. URN Sub-Namespace Registration . . . . . . . . . . . . . . . 46
     15.2. XML Schema Registration  . . . . . . . . . . . . . . . . . . 47
     15.3. MIME Media Type Registration for 'application/ccmp+xml'  . 47
     15.4. DNS Registrations  . . . . . . . . . . . . . . . . . . . . . 48
       15.4.1. Registration  of a Location Server Application
               Service Tag  . . . . . . . . . . . . . . . . . . . . . . 48
       15.4.2. Registration of a Location Server Application
               Protocol Tag for HELD  . . . . . . . . . . . . . . . . . 48
     15.5. CCMP Protocol Registry . . . . . . . . . . . . . . . . . . . 49
       15.5.1. CCMP Message Types . . . . . . . . . . . . . . . . . . . 49
       15.5.2. CCMP Response Codes  . . . . . . . . . . . . . . . . . . 50
   16. Security Considerations  . . . . . . . . . . . . . . . . . . . . 51
   17. Acknowledgments  . . . . . . . . . . . . . . . . . . . . . . . . 51
   18. Changes since last Version . . . . . . . . . . . . . . . . . . . 51
   19. References . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
     19.1. Normative References . . . . . . . . . . . . . . . . . . . . 52
     19.2. Informative References . . . . . . . . . . . . . . . . . . . 52
   Authors' Addresses . . . . . . . . . . . . . . . . . . . . . . . . . 53
   Intellectual Property and Copyright Statements . . . . . . . . . . . 55
```

1. Introduction

The Framework for Centralized Conferencing [RFC5239] (XCON FW) defines a signaling-agnostic framework, naming conventions and logical entities required for building advanced conferencing systems. The XCON FW 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.

CCMP implements the client-server model within the XCON FW, with the conferencing client and conference control server acting as client and server, respectively. CCMP is an instance of conference control protocol (CCP).

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.

CCMP can 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].

CCMP follows the well-known REST (REpresentational State Transfer) architectural style [REST] This document describes how the CCMP specification maps 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.

Section 4 motivates the design of 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, with the sequence of protocol requests and responses discussed in Section 8 and examples provided in Section 11. The protocol parameters are detailed in Section 10. Section 12 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].

W3C: World Wide Web Consortium, the organization that developed the SOAP and WSDL specifications referenced within this document.

4. Rationale and Motivation

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.

The operations on these objects can be implemented in at least two different ways, namely as remote procedure calls and by defining resources. A remote procedure call (RPC) mechanism 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.

Alternatively, conference objects can 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 a natural URIs.

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. The coarser-grained approach in 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.

We assume that each update operation 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.

In summary, a REST-style approach must ensure sure that all operations can be mapped to HTTP operations, while 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.

For CCMP, the resource (REST) model appears more attractive, since the conference operations fit the CRUD approach.

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

```
   .........................................................
   .  Conferencing System                                  .
   .                                                       .
   .        +----------------------------------------+     .
   .        |   C O N F E R E N C E   O B J E C T    |     .
   .      +-+--------------------------------------+ |     .
   .      |   C O N F E R E N C E   O B J E C T    | |     .
   .    +-+--------------------------------------+ | |     .
   .    |   C O N F E R E N C E   O B J E C T    | | |     .
   .    |                                        | | |     .
   .    |                                        | |-+     .
   .    |                                        |-+       .
   .    +----------------------------------------+         .
   .                        ^                              .
   .                        |                              .
   .                        v                              .
   .               +-------------------+                   .
   .               | Conference Control|                   .
   .               | Server            |                   .
   .               +-------------------+                   .
   .                        ^                              .
   .........................|...............................
                            |
                            |Conference
                            |Control
                            |Protocol
                            |
                            |
   .........................|...............................
   .                        V                              .
   .                +----------------+                     .
   .                | Conference     |                     .
   .                | Control        |                     .
   .                | Client         |                     .
   .                +----------------+                     .
   .                                                       .
   .  Conferencing Client                                  .
   .........................................................
```

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 framework for centralized conferencing document 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 users 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

The primary function of the protocol defined within this document is to provide a conference control client with the ability to carry out operations on a conference object as a whole and on specific elements within a conference object. This section describes the four basic operations on a conference object: retrieve, create, change and delete. The recommended HTTP method for each of the basic operations is described. The XCON-URI as discussed in Section 6.1 is the primary target for each of these operations. The normative protocol details as to the applicability of each of the operations for the various CCMP requests and responses are provided in Section 8.

7.1. Retrieve

The "retrieve" operation is used by a client to query a system for a specific template in the form of a blueprint prior to the creation of a conference. In this case, the "retrieve" operation often follows a "blueprintsRequest" operation, although a conferencing control client may be pre-configured to perform the "retrieve" operation on a specific blueprint.

The "retrieve" operation is also used to get the current representation of a specific conference object (or specific parameters in the conference object) for a conference reservation or an active conference. The unique conference identifier (XCON-URI) is included in the CCMP request.

The "retrieve" operation returns the XML document describing the conference object in its current state including all inherited values or the specific parameters per the specific request type. Elements may be marked by attributes, in particular, whether they are specific to this instance or have been inherited from the parent node.

In the case of a RESTful implementation of the protocol, HTTP GET MUST be used on XCON-URIs, so that clients can obtain data about conference objects in the form of XML data model documents.

7.2. Create

The "create" operation is used by a client to create and reserve a conference object or a new conference user. The creation of a conference object can be explicit by requesting it to be created based upon a specific blueprint, based on an existing conference object (e.g., cloning a conference reservation or active conference object) or based on the data included in the request. In the first two cases, a specific XCON-URI MUST be included in the request.

When the creation of a conference object is implicit, with no conference object for a blueprint or existing conference specified and no data included in the request (i.e., an "empty" request sent to a specific conference server), the creation and reservation of the conference instance is based on the default conference object. The default conference object is specific to a conference control server and its specification is outside the scope of this document.

A client may first send a request with "retrieve" operation in order to obtain all the data as defined in [I-D.ietf-xcon-common-data-model] for the specific blueprint or existing conference object. This would allow the client to modify the data prior to sending the request with the "create" operation. In this case, the request would also include all the data. If the client wants to create the new conference by cloning the blueprint or existing conference object, there would be no data included in the request. The client may later modify this data by sending a request with a "change" operation.

When creating conferences, any XCON-URI included by the client is considered as the target conference object from which the new conference is to be created. To avoid identifier collisions and to

conform to local server policy, the conference control server typically chooses a different identifier for the newly created conference object. The identifier is returned in the response.

In addition, the conference description MAY contain a calendar element, in the iCal format in XML rendition defined in CPL [RFC3880] or (preferable, if available as stable reference) xCal [I-D.royer-calsch-xcal]. This description indicates when the conference is active.

The "create" operation may also be used to create a new conference user with the "userRequest" message. In this case, the "userResponse" to this operation includes an XCON-USERID.

In the case of a RESTful implementation of the protocol, HTTP PUT MUST be used to create a new object as identified by the XCON-URI or XCON-USERID.

7.3. Change

The "change" operation updates the conference object as referenced by the XCON-URI included in the request. A request which attempts to change a non-existing object is an error, as is a request which attempts to change a parameter that is inherited from a protected element.

During the lifetime of a conference, this operation is used by a conference control client to manipulate a conference object. This includes the ability to manipulate specific elements in the conference object through element specific requests such as "userRequest" or "sideBarRequest", etc.

Upon receipt of a "change" operation, the conference control server updates the specific elements in the referenced conference object. Object properties that are not explicitly changed, remain as-is. This approach allows a conference control client to manipulate objects created by another application even if the manipulating application does not understand all object properties.

In the case of a RESTful implementation of the protocol, either HTTP PATCH or HTTP POST MUST be used to change the conference object identified by the XCON-URI.

7.4. Delete

This conference control operation is used to delete the current representation of a conference object or a specific parameter in the conference object and requires the unique conference identifier

(XCON-URI) be provided by the client.

A request which attempts to delete a conference object that is being referenced by a child object is an error.

In case of a RESTful implementation of the protocol, HTTP DELETE MUST be used to delete conference objects and parameters within conference objects identified by the XCON-URI.

8. Protocol Operations on Conference Objects

The primary function of CCMP is to provide a conference control client with the ability to carry out specific operations (Section 7) on a conference object through the protocol requests and responses. In case of a RESTful implementation of the protocol, the CCMP requests (Section 8.2)and responses (Section 8.3) MUST be represented as HTTP requests and responses. The basic CCMP request/response pairs defined in this document are:

blueprintsRequest/blueprintsResponse: The blueprintsRequest is used to ascertain the list of blueprints available at the conference server. The blueprintsResponse returns a list of the requested blueprints, in the form of XCON URIs.

confsRequest/confsResponse: The confsRequest is used to ascertain conference reservations and active conferences supported by the server. The confsResponse returns a list of the requested types of conference objects (i.e. Conference Reservations and/or Active Conferences) supported by the specific conference server.

blueprintRequest/blueprintResponse: The blueprintRequest is used to request an operation on a specific blueprint.

confRequest/confResponse: The confRequest is used to request an operation on the conference object as a whole.

userRequest/userResponse: The userRequest is used to request an operation on the "user" element in the conference object. [Editor's Note: we may want to add more discrete user requests/ responses as this is a very broad parameter]

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

sidebarRequest/sidebarResponse: This sidebarRequest is used to retrieve the information related to a sidebar or to create, change or delete a specific sidebar. [Editor's Note: the data model defines a byVal and byRef sidebar type. Rather than define two root operations, the preference is to have these two types reflected by a parameter in the request.]

With respect to the above mentioned operations, we remark that the

difference between "blueprintsRequest" and "confsRequest" only exists at the semantic level. They both ask for a list of XCON-URIs and they have exactly the same format. The returned XCON-URIs, though, represent blueprints in the former case, real (i.e. either active or reserved) conferences in the latter. The fact that blueprints and conferences share the same representation (a conference object compliant with the XCON data model) is a mere coincidence. The same holds for "confRequest/blueprintRequest", which aim at managing, respectively, a specific conference object and a specific blueprint.

To simplify operations, a conference control server treats certain parameters as suggestions (e.g., for the "create" and "change" operations), as noted in the object description. If the conference control server cannot set the parameter to the values desired, it picks the next best value, according to local policy and returns the values selected in the response. If the client is not satisfied with these values, it simply deletes the object.

As illustrated above, along with the protocol requests and responses for manipulating the conference object, there are also querying mechanisms ("blueprintsRequest"/"blueprintsResponse" and "confsRequest/confsResponse") to get information about either blueprints or scheduled/active conferences supported by the server. Any elements with namespaces not understood by the server are to be ignored by the server. This allows a client to include optional elements in requests without having to tailor its request to the capabilities of each server.

A conference client must first discover the conference control server as described in Section 8.1. The conference control server is the recipient of the CCMP requests.

8.1. 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 15.4.1 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 15.4.2, is used to identify an XCON server that understands the CCMP protocol.

The NAPTR records in the following example Figure 2 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 2: Sample XCON:CCMP Service NAPTR Records

Details for the "XCON" Application Service tag and the "CCMP" Application Protocol tag are included in Section 15.4.

8.2. Constructing a CCMP Request

Construction of a valid CCMP request is based upon the operations defined in Section 7, depending upon the function and associated information desired by the conference control client. The next two sections provide details of the "blueprintsRequest" and "confsRequest" messages, which differ from the other CCMP messages in that they are only used to ask the conference system for general information (blueprints and conferences). Subsequent sections

summarize the CCMP requests related to the specific operations in Section 7.

8.2.1. blueprintsRequest

The "blueprintsRequest" is used by a client to query a system for its capabilities in terms of types of conferences supported and isn't targeted toward a particular conference object. Detailed information about a specific blueprint, can be subsequently obtained through the blueprintRequest operation, which is used to retrieve a whole XCON blueprint (in the form of a conference object) available at the server.

The "blueprintsResponse" returns the XML namespaces that the server understands and the namespaces to be used in responses that it requires the client to understand. Within the conferencing system, the namespaces correlate with blueprints, as specified in the XCON framework. The blueprints are comprised of conference information initialized to specific values and ranges. Each blueprint has a corresponding XCON-URI.

8.2.2. confsRequest

The "confsRequest" is used by a client to query a system for information about reserved/active conferences and isn't targeted toward a particular conference object. Detailed information about a specific conference, can be subsequently obtained through the confRequest operation, which can be used to retrieve a whole XCON conference (in the form of a conference object) available at the server.

The "confsResponse" returns the XCON-URIs of all reserved and active conferences currently hosted by the server.

8.2.3. Operations Requests

Construction of other valid CCMP requests is based upon the operations defined in Section 7, depending upon the function and associated information desired by the conference control client. The following table summarizes specific request type and processing for each of the "operations". A value of "N/A" indicates the specific operation is not valid for the specific CCMP request. Following the table examples for each of the HTTP operations for each of the request types is provided.

Editors' Notes:

1. Sidebars need additional consideration - e.g., due to the byVal and byRef options, it's messy. Operations approach may need additional consideration (or we need separate request types).

Operation (HTTP method) ------------- Request Type	Retrieve (GET)	Create (PUT)	Change (PATCH or POST)	Delete (DELETE)
blueprintsRequest	Gets list of available blueprints	N/A	N/A	N/A
confsRequest	Gets list of active or reserved confs	N/A	N/A	N/A
blueprintRequest	Gets a specific blueprint	Creates a blueprint (needs admin privileges)	Changes a blueprint (needs admin privileges)	Deletes a blueprint (needs admin privileges)
confRequest	Gets conference object	Creates conference object	Changes conference object	Deletes conference Object as a whole
userRequest	Gets a specific user element	Creates a user and associated XCON-UserID	Modifies the specified user element	Deletes a user element as a whole
usersRequest	Gets a specific users element	N/A	Modifies the specified users element	Deletes a users element as a whole

sidebarRequest	Gets a sidebar element by Val or by Ref	Creates a new sidebar by Val	Modifies a sidebar by Val	Removes/del etes the entire sidebar by Val

Table 1: Request Type Operation Specific Processing

The following provides HTTP examples for each of the valid operations for each request type in the above table Table 1

- o blueprintsRequest: GET /blueprints
- o confsRequest: GET /confs
- o blueprintRequest
  - * GET /blueprint/blueprintId
  - * PUT /blueprint/blueprintId
  - * POST /blueprint/blueprintId
  - * DELETE /blueprint/blueprintId
- o confRequest
  - * GET /confs/confObjId
  - * PUT /confs/confObjId
  - * POST /confs/confObjId
  - * DELETE /confs/confObjId
- o userRequest
  - * GET /user/confUserId
  - * PUT /user/confUserId
  - * POST /user/confUserId
  - * DELETE /user/confUserId
- o usersRequest
  - * GET /confs/confObjId/users
  - * POST /confs/confObjId/users
  - * DELETE /confs/confObjId/users
- o sidebarRequest
  - * By val: GET /confs/confObjId/sidebars/entityAttribute
  - * By val: N/A (use a "confRequest" message with a "change" operation for this)
  - * By val: N/A (use a "confRequest" message with a "change" operation for this)
  - * By val: N/A (use a "confRequest" message with a "change" operation for this)
  - * By ref: GET /sidebars/sidebarId

8.3. Handling a CCMP Response

A response to the CCMP request MUST contain a response code and may contain other elements depending upon the specific request and the value of the response code.

In case of a RESTful implementation, the CCMP response message MUST be enclosed in a HTTP response message. CCMP-related error codes will be carried in the body of the response: no mapping is proposed in this document regarding the potential association between CCMP and HTTP error codes. For the sake of adhering to the principle of separation of concerns, HTTP maintains its own semantics, while delegating to the CCMP response message (which is in the body of the HTTP response) the task of informing the CCMP client about error conditions. This means that, in case of a CCMP error, the client receives a 200 OK in the HTTP response, but a CCMP-specific response code in the body of such response.

All response codes are application-level, and MUST only be provided in successfully processed transport-level responses. For example where HTTP is used, CCMP Response messages MUST be accompanied by a 200 OK HTTP response.

The set of CCMP Response codes currently contain the following tokens:

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.., when trying to make a "confRequest" operation with a request type equal to "delete" on a conference object representing a blueprint, 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.

CCMP Response codes are defined to allow for extensibility. A conference control client SHOULD treat unrecognized response codes as it handles a Response code of "notImplemented".

8.3.1. blueprintsResponse

A "blueprintsResponse" message containing a response code of "success" MUST include the XML namespaces that the server understands and the namespaces to be used in subsequent responses that it requires the client to understand. Future work may add more global capabilities rather than conferencing system specific. Within the conferencing system, the namespaces correlate with blueprints, as specified in the XCON framework. The blueprints are comprised of conference information initialized to specific values and ranges.

Upon receipt of a successful "blueprintsResponse" message, a conference control client may then initiate a "blueprintRequest" with a "retrieve" operation per Section 7.1 to get a specific conference blueprint.

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

The response codes of "modified", "deleteParentFailed" and "changeFailedProtected" are not applicable to a "blueprintsRequest" and should be treated as "serverInternalError", the handling of which is specific to the conference control client.

A "blueprintsResponse" message containing any other response code is an error and the handling is specific to the conference control client. Typically, an error for a "blueprintsRequest" indicates a configuration problem in the conference control server or in the client.

8.3.2. confsResponse

A "confsResponse" message containing a response code of "success" MUST include the list of XCON-URIs associated with reserved/active conferences at the server.

Upon receipt of a successful "confsResponse" message, a conference

control client may then initiate a "confRequest" with a "retrieve" operation per Section 7.1 to get a specific conference object.

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

The response codes of "modified", "deleteParentFailed" and "changeFailedProtected" are not applicable to a "confsRequest" and should be treated as "serverInternalError", the handling of which is specific to the conference control client.

A "confsResponse" message containing any other response code is an error and the handling is specific to the conference control client. Typically, an error for a "blueprintsRequest" indicates a configuration problem in the conference control server or in the client.

8.3.3. Operation Responses

The following sections detail the operation specific handling of the response codes, including details associated with specific types of responses in the cases where the response handling is not generic.

8.3.3.1. Retrieve Operation Responses

A confResponse for a "retrieve" operation containing a response code of "success" MUST contain the full XML document describing the conference object in its current state including all inherited values. Elements may be marked by attributes, in particular, whether they are specific to this instance or have been inherited from the parent node.

A blueprintResponse for a "retrieve" operation containing a response code of "success" MUST contain the full XML document describing the conference object associated with the requested blueprint

Any other CCMP response message (e.g., userResponse, usersResponse, etc.) for a "retrieve" operation containing a response code of "success" MUST contain the XML document describing the specific target parameter (as indicated by the specific type of Request) from the conference object.

If a response code of "objectNotFound" is received in a "blueprintResponse" message to a "blueprintRequest" to get the initial blueprint, it is RECOMMENDED that a conference control client attempt to retrieve another conference blueprint if more than one had

been received in the "blueprintsResponse" message. If there was only one blueprint in the "blueprintsResponse" initially, then the client should send another "blueprintsRequest" message to determine if there may be new or additional blueprints for the specific conferencing system. If this "blueprintsResponse" message contains no blueprints, the handling is specific to the conference control client. This might indicate, for example, that something is going wrong at the server, since no more blueprints are now available at it. In such case, the client MAY interpret the new answer as a 'serverInternalError' and assume that no more service associated with blueprints (e.g. creation of a new conference starting from a server-side template) is available.

If a response code of "requestTimeout" is received in the CCMP response, a conference control client MAY re-attempt the request within a period of time that would be specific to a conference control client or conference control server.

Response codes such as "notImplemented" and "forbidden" indicate that a subsequent "retrieve" would not likely be sucessful. Handling of these and other response codes is specific to the conference control client. For example, in the case of some clients a "blueprintsRequest" operation might be performed again or another conference control server may be accessed.

The response codes of "modified", "deleteParentFailed" and "changeFailedProtected" are not applicable to the "retrieve" operation and SHOULD be treated as "serverInternalError", the handling of which is specific to the conference control client.

8.3.3.2. Create Operation Responses

The only valid responses containing a "create" operation are a "confResponse", a "blueprintResponse" and the "userResponse". The "blueprintRequest" containing a "create" operation has to be considered a special operation, used by a conference server administrator wishing to remotely add a new blueprint to the conference server. The operation requires that the new blueprint is associated with an XCON-URI. Such URI is provided by the administrator in the request, but has to be considered as a suggestion. The conference server MAY change such identifier and create a new one. The new identifier MUST be returned to the client as part of the "blueprintResponse" message. If the CCMP response contains a response code of "success", a "confResponse" message MUST contain the XCON-URI for the conference object and a "userResponse" message MUST contain the XCON-USERID.

If the confResponse to a "create" operation contains a response code

of "modified", along with the XCON-URI for the conference object, the response MUST also contain the entire XML document associated with that conference object for a "confRequest". For example, in the case where the conference object contained a calendar element, the conference server may only offer a subset of the dates requested, thus the updated dates are included in the returned XML document.

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

Response codes such as "unauthorized", "forbidden" and "operationNotAllowed" indicate the client does not have the appropriate permissions, there is an error in the permissions, or there is a system error in the client or conference control server, thus re-attempting the request would likely not succeed.

The response codes of "deleteParentFailed" and "changeFailedProtected" are not applicable to the "create" operation and SHOULD be treated as "serverInternalError", the handling of which is specific to the conference control client.

Any other response code indicates an error in the client or conference control server (e.g., "forbidden", "badRequest") and the handling is specific to the conference control client.

8.3.3.3. Change Operation Responses

If the CCMP response to the "change" operation contains a response code of "success", the response SHOULD also contain the XCON-URI for the conference object that was changed.

The "blueprintRequest" containing a "change" operation has to be considered a special operation, used by a conference server administrator wishing to remotely an existing blueprint in the conference server.

If the CCMP response to the "change" operation contains a response code of "modified", the response MUST contain the XCON-URI for the conference object and the appropriate XML document (either the full XML document for a confResponse or specific paramaters for the other CCMP request types) associated with that conference object. For example, a conferencing system may not have the resources to support specific capabilities that were changed, such as <codecs> in the <available-media>, thus the <codecs> supported are included in the returned XML document.

If the CCMP response code of "requestTimeout" is received, a conference control client MAY re-attempt the request within a period of time that would be specific to a conference control client or conference control server.

Response codes such as "unauthorized", "forbidden", "operationNotAllowed" and "changeFailedProtected" indicate the client does not have the appropriate permissions, the conference is locked, there is an error in the permissions, or there is a system error in the client or conference control server, thus re-attempting the request would likely not succeed.

The response code of "deleteParentFailed" is not applicable to the "change" operation and SHOULD be treated as "serverInternalError", the handling of which is specific to the conference control client.

Any other response code indicates an error in the client or conference control server (e.g., "forbidden", "badRequest") and the handling is specific to the conference control client.

[Note by spromano: In case of "change" with a userRequest, the server first has to change the user's information stored; then, it has to update all conference objects which include that user. The association between the user and the conferences in which she/he is participating is guaranteed through the "entity" attribute of the <user> element. IMO, after doing all that, the server just answers with a userResponse message; then, if it is also using notifications, it might raise events towards the interested subscribers, to notify them about the changes in the updated conference objects. Is this right??]

8.3.3.4. Delete Operation Responses

If the CCMP response to the "delete" operation contains a response code of "success", the response MUST contain the XCON-URI for the conference object that was deleted for a "confResponse" or whose data element(s) were deleted for the other response types.

The "blueprintRequest" containing a "delete" operation has to be considered a special operation, used by a conference server administrator wishing to remotely remove a blueprint from the conference server.

The response code of "deleteParentFailed" indicates that the conference object could not be deleted because it is the Parent of another conference object that is in use. In this case, the response also includes the XCON-URI for the conference object and is only applicable to a "confResponse". If this response code is received

for any other type of CCMP response, it should be treated as "serverInternalError", the handling of which is specific to the conference control client.

If a response code of "requestTimeout" is received, a conference control client MAY re-attempt the request within a period of time that would be specific to a conference control client or conference control server.

Response codes such as "unauthorized", "forbidden" and "operationNotAllowed" indicate the client does not have the appropriate permissions, the conference is locked, the object that the client is trying to delete is actually a blueprint, there is an error in the permissions, or there is a system error in the client or conference control server, thus re-attempting the request would likely not succeed.

The response code of "changeFailedProtected" is not applicable to the "delete" operation and SHOULD be treated as "serverInternalError", the handling of which is specific to the conference control client.

Any other response code indicates an error in the client or conference control server (e.g., "forbidden", "badRequest") and the handling is specific to the conference control client.

[Note by spromano (same comment as for "change"): In case of "delete" with a userRequest, the server first has to delete the user's information stored; then, it has to update all conference objects which include that user. The association between the user and the conferences in which she/he is participating is guaranteed through the "entity" attribute of the <user> element. IMO, after doing all that, the server just answers with a userResponse message; then, if it is also using notifications, it might raise events towards the interested subscribers, to notify them about the changes in the updated conference objects. Is this right??]

9. Managing sidebars

Sidebars can be either "by reference" or "by value". The management of sidebars differs in the two cases, as discussed below

9.1. Sidebars by value

Sidebars by value represent an inner part of the conference object associated with the root conference from which they stem. One or more sidebars by value are then created by using the "confRequest" message with an operation of "change". The conference description

provided in the request MUST contain the desired sidebars information, in the form of a sequence of one or more <entry> elements under the <sidebars-by-val> element. Information about a sidebar by value can be accessed directly through a "sidebarRequest" message containing the identifier of the required sidebar (i.e. its "entity" attribute value).

9.2. Sidebars by reference

Sidebars by reference represent semi-independent conference objects, i.e. objects that exist on their own, but which are strictly coupled to the conference object from which they stem. A sidebar by reference is then created by using the "confRequest" message with an operation of "create".

Editor's Note: should we have a means to indicate that the object we are creating is actually a sidebar? This would go in the confRequest/create message. Otherwise, we might add a sidebarRequest/create operation which basically does a conference creation, but, e.g., stores it in a different repository (/sidebars rather than /confs).

Once the sidebar has been created, you can add it to a conference by issuing a "confRequest" message with a "change" operation on the conference object which the sidebar belongs to. Information about a sidebar by reference can be accessed directly through a "sidebarRequest" message containing the identifier of the required sidebar (i.e. the value of its <uri> element).

10. Protocol Parameters

This section describes in detail the parameters that are used for the CCMP protocol.

10.1. Operation Parameter

The "operation" attribute is a mandatory token included in all CCMP request and response messages. This document defines four possible values for this parameter: "retrieve", "create", "change" and "delete".

10.2. ConfObjID Parameter

The "confObjID" attribute is an optional URI included in the CCMP request and response messages. This attribute is required in the case of an "operation" of "retrieve", "change", and "delete" in the CCMP request and response messages. The attribute is optional for an

"operation" of "create" in the "confRequest" message. The "create" cases for which this parameter is REQUIRED are described in Section 7.2. This attribute is the XCON-URI which is the target for the specific operation. [Editor's Note: it might be good to re-iterate the normative text here.]

This attribute is not included in the "userRequest" message for an operation of "create". In this case, the conference control client is requesting the creation of a new conference user, as detailed in Section 10.3.

In the cases where the "conference-info" parameter Section 10.6 is also included in the requests and responses, the "confObjID" MUST match the XCON-URI in the "entity" attribute.

10.3. ConfUserID Parameter

The "confUserID" attribute is optional URI included in the CCMP request and response messages. This is the XCON-USERID for the conference control client initiating the request. The attribute is required in the CCMP request and response messages with the exception of the "userRequest" message. The "confUserID" parameter is used to determine if the conference control client has the authority to perform the operation. Note that the details for authorization and related policy are specified in a separate document [TBD].

This attribute is optional only for an "userRequest" message with a "create" operation. In this case, the request MUST include information about the user in the "user" element. At a minimum, the request MUST include the "user" element with an "entity" attribute. For this case, the conference control server MUST create a new conference user and return the associated confUserID in the response, if the allocation of a new XCON-USERID is succesful.

In the case where there is a confUserID in the request that has already been allocated, this request may be the creation of a confUserID for the conference control client to take on an additional role.

This attribute is required in the "userResponse" message in the case of an "operation" of "create" and for all other responses.

10.4. ResponseCode Parameter

The "responseCode" attribute is a mandatory parameter in all CCMP response messages. The values for each of the "responseCode" values are detailed in Section 8.3 with the associated processing described in Section 8.3.3.

10.5. Blueprints Parameter

The "blueprints" attribute is an optional parameter in the CCMP blueprintsResponse message. In the case of a "blueprintsRequest" message, the "blueprintsResponse" message with a "responseCode" of "success" SHOULD include the "blueprints" supported by the conference control server. The "blueprints" attribute is comprised of a list of blueprints supported by the specific conference server and includes a conference system specific "blueprintName" and a "confObjID" in the form of an XCON-URI for each of the blueprints.

10.6. Conference-info Parameter

The "conference-info" element is optional in the CCMP confRequest and confResponse messages.

The "conference-info" element contains the data for the conference object that is the target for the "confRequest" operations for "create", "change" and "delete" operations. It is returned in a "confResponse" if the "confResponse" contains a responseCode of "modified" or if the original CCMP request for the "create" operation did not contain a "conference-info" element. The latter case occurs when a conference control client sends a "confRequest" containing any of the following: - a "confObjID" associated with a specific blueprint - a "confObjID associated with a specific active conference or conference reservation that was included in a "confsResponse" message - no "confObjID" (or "conference-info") element, in which case the request is to create a conference object based on a default provided by a conferencing system.

The "conference-info" element is also returned in a "userResponse" message, in the case of a "change" operation. In such case, in fact, the request contains the <user> element to be added to the conference indicated in the <confObjID> parameter; the associated answer SHOULD carry the updated conference object in its body.

The details on the information that may be included in the "conference-info" element MUST follow the rules as specified in the XCON Data Model document [I-D.ietf-xcon-common-data-model]. The conference control client and conference control server MUST follow those rules in generating the "conference-info" in any of the CCMP request and response messages.

Note that the "conference-info" element is not explicitly shown in the XML schema (Section 12) due to XML schema constraints.

10.7. User Parameter

The "user" element contains the data for the conference user that is the target for the CCMP request operations. It is REQUIRED for all "userRequest" messages.

The details on the information that may be included in the "user" element MUST follow the rules as specified in the XCON Data Model document [I-D.ietf-xcon-common-data-model]. The conference control client and conference control server MUST follow those rules in generating the "user" in any of the CCMP request and response messages.

Note that the "user" element is not explicitly shown in the XML schema Section 12 due to XML schema constraints.

10.8. Users Parameter

The "users" element contains the data for the conference users that are the target for the CCMP request operations. It is REQUIRED for all "usersRequest" messages.

The details on the information that may be included in the "users" element MUST follow the rules as specified in the XCON Data Model document [I-D.ietf-xcon-common-data-model]. The conference control client and conference control server MUST follow those rules in generating the "users" in any of the CCMP request and response messages.

Note that the "users" element is not explicitly shown in the XML schema Section 12 due to XML schema constraints.

10.9. Sidebar Parameters

The "sidebar" parameter contains the data for the sidebar that is the target for the CCMP request operations. It is REQUIRED for all "sidebarRequest" messages. There are two elements associated with a sidebar: "sidebar-by-val" and "sidebar-by-ref". The elements relate to whether the data for the sidebar is in the same conference object for which it serves as a sidebar or whether a new conference object is created for the sidebar.

The details on the information that may be included in the "sidebar-by-val" or "sidebar-by-ref" element MUST follow the rules as specified in the XCON Data Model document [I-D.ietf-xcon-common-data-model]. The conference control client and conference control server MUST follow those rules in generating the "sidebar-by-val" or "sidebar-by-ref" element in any of the CCMP

request and response messages.

11. Examples

Examples on the use of HTTP as the CCP based on a RESTful implementation are provided in Section 11.1. The body of the HTTP methods contains the CCMP operations and data. Examples of the CCMP operations and related data are provided in section Section 11.2

11.1. HTTP methods for realizing a RESTful CCMP

This section provides a series of examples using the HTTP methods for realization of the CCMP. The examples provide a sequence of operations that a typical user might invoke in activating a conference, adding users to a conference, retrieving conference data and then deleting an active conference. Note, the examples do not include any details beyond the basic operation. For example, the "Host" that would be the result of discovery of the conference server per Section 8.1 would be included in the HTTP messages.

Alice retrieves info about active/scheduled CCMP 'conferences':

```
 CCMP client "Alice"                              ConfS
      |                                             |
      | GET /confs                                  |
      |-------------------------------------------->|
      |                                             |--+ Prepare
      |                                             |  | formatted
      |                          200 OK (w/ body)   |<-+ conf info
      |<--------------------------------------------|    (list of
      |                                             |     conf objs)
```

Figure 3: Getting a List of Active Coferences

Alice is now able to retrieve info about a specific conference:

```
   CCMP client "Alice"                               ConfS
   |                                              |
   | GET /confs/confid-34fg67h                    |
   |--------------------------------------------->|
   |                                              |--+ Prepare
   |                                              |  | formatted
   |                             200 OK (w/ body) |<-+ XML info
   |<---------------------------------------------|
   |                                              |
```

Figure 4: Getting a Specific Coference

Alice decides to add a new user to this conference:

```
    CCMP client "Alice"                               ConfS
     |                                              |
     | PUT /confs/confid-34fg67h/users/pippo876     |
     | (w/ body=new user info)                      |
     |--------------------------------------------->|
     |                                              |--+ Add new user
     |                                              |  | to data model
     |                             200 OK (w/ body) |<-+ and update
     |<---------------------------------------------|   user in system
     |                                              |
     |                                              | (event triggered
     |                                              |  e.g. RFC4575)
     |                                              |---------------->
     |                                              |
```

Figure 5: Adding a New User to an Active Conference

Subsequent GETs on both the conference object as a whole and the users portion reflect the addition of the New User:

```
    CCMP client "Alice"                               ConfS
          |                                              |
          | GET /confs/confid-34fg67h/users/pippo876     |
          |--------------------------------------------->|
          |                                              |--+ Prepare
          |                                              |  | formatted
          |                             200 OK (w/ body) |<-+ XML info
          |<---------------------------------------------|
          |                                              |
          | GET /users/pippo876                          |
          |--------------------------------------------->|
          |                                              |--+ Prepare
          |                                              |  | formatted
          |                             200 OK (w/ body) |<-+ XML info
          |<---------------------------------------------|
          |                                              |
```

Figure 6: Getting a Specific Conference Object after Changes

Alice updates some info related to the same user:

```
    CCMP client "Alice"                               ConfS
         |                                              |
         | POST /confs/confid-34fg67h/users/pippo876    |
         | (w/ body=updated user info)                  |
         |--------------------------------------------->|
         |                                              |--+ Update user
         |                                              |  | in data
         |                             200 OK (w/ body) |<-+ and in
         |<---------------------------------------------|    system
         |                                              |
         |                                              |Event trigger
         |                                              |e.g. RFC4575
         |                                              |------------->
         |                                              |
```

Figure 7: Updating a User's Information

Alice destroys the running conference: when trying to access it, the server returns an error:

```
 CCMP client "Alice"                              ConfS
      |                                              |
      | DELETE /confs/confid-34fg67h                 |
      |--------------------------------------------->|
      |                                              |--+ Prepare
      |                                              |  | formatted
      |                            200 OK (w/ body)  |<-+ XML info
      |<---------------------------------------------|
      |                                              |
      | GET /confs/confid-34fg67h                    |
      |--------------------------------------------->|
      |                                              |--+ ConfS can't
      |                                              |  | find the
      |                            200 OK (w/body:   |<-+ conference
      |<---------------------------------------------|
      |                responseCode=objectNotFound   |
      |                                              |
```

Figure 8: Deleting an Active Coference

11.2. CCMP Detailed Message Body Examples

The examples below contain simply the <body> of the requests and responses. In the case that HTTP serves as the transport, the HTTP methods as identified in Table 1 (and per the examples in Section 11.1) would include the CCMP requests and Responses as the body of the HTTP methods.

11.2.1. Creating a New Conference

The first example creates a new conference.

```
          <confRequest xmlns="urn:ietf-params:xml:ns:xcon:ccmp">
          <operation>create</operation>
          <confUserID> userA-confxyz987 </confUserID>

          <conference-info
            xmlns="urn:ietf:params:xml:ns:conference-info"
            version="1">
            <conference-description>
             <parent>http://example.com/conf200</parent>
             <subject>Agenda: This month's goals</subject>
             <conf-uris>
               <entry>
                 <uri>sips:conf223@example.com</uri>
                 <purpose>participation</purpose>
               </entry>
             </conf-uris>
             <service-uris>
               <entry>
                <uri>http://sharep/salesgroup/</uri>
                <purpose>web-page</purpose>
               </entry>
               <entry>
                <uri>http://example.com/conf233</uri>
                <purpose>control</purpose>
               </entry>
              </service-uris>
             </conference-description>
          </conference-info>

        </confRequest>
```

Figure 9: Create Request Example

The response to this request is shown below; it returns the object identifier as a URL and the final conference description, which may modify the description offered by the user.

```
    <confResponse xmlns="urn:ietf-params:xml:ns:xcon:ccmp"
      <operation>create</operation>
      <responseCode> modified </responseCode>
      <confObjID> xcon:confxyz987@example.com </confObjID>
      <confUserID> userA-confxyz987 </confUserID>

      <conference-info
        xmlns="urn:ietf:params:xml:ns:conference-info"
        version="1">
        <entity> xcon:confxyz987@example.com </entity>
        <conference-description>
         <parent>http://example.com/conf200</parent>
         <subject>Agenda: This month's goals</subject>
         <conf-uris>
           <entry>
             <uri>sips:conf223@example.com</uri>
             <purpose>participation</purpose>
           </entry>
         </conf-uris>
         <service-uris>
           <entry>
            <uri>http://sharep/salesgroup/</uri>
            <purpose>web-page</purpose>
           </entry>
           <entry>
            <uri>http://example.com/conf233</uri>
            <purpose>control</purpose>
           </entry>
          </service-uris>

      <!-- Addt'l modified conference description including users alice,
           bob and userA... -->

          <allowed-users-list>
           <target uri="sip:alice@example.com" method="dial-out"/>
           <target uri="sip:bob@example.com" method="dial-out"/>
           <target uri="sip:userA@example.com" method="dial-in"/>
          </allowed-users-list>

         </conference-description>
      </conference-info>

      </confResponse>
```

Figure 10: Create Response Example

11.2.2. Creating a New Conference User

The request below creates a new conference user, independent of a specific conference object.

```
    <userRequest xmlns="urn:ietf-params:xml:ns:xcon:ccmp">
       <operation>create</operation>

       <user entity="sip:bob@example.com">
            <role>observer</role>
       </user>

    </userRequest>
```

Figure 11: Create User Example

The response to this request is shown below; it returns the conference user identifier.

```
   <userResponse xmlns="urn:ietf-params:xml:ns:xcon:ccmp">
     <operation>create</operation>
     <responseCode> success </responseCode>
     <confUserID>userC-confxyz987</confUserID>
   </userResponse>
```

Figure 12: Create Response Example

11.2.3. Adding a User to a Conference

The request below adds a user to the conference identified by the XCON-URI. Note that the user in "confUserID" element is the user requesting that the user "sip:claire@example.com" be added to the conference. The user may or may not be "claire" (i.e., a user, such as the moderator, can add another user to the conference.

Editor's note: Do we need to consider users adding users OBO of other users or in that case do we just change the conference object as a whole?

```
      <userRequest xmlns="urn:ietf-params:xml:ns:xcon:ccmp">
          <operation>change</operation>
          <confObjID> xcon:confxyz987@example.com </confObjID>
          <confUserID> userC-confxyz987 </confUserID>

          <user entity="sip:claire@example.com">
            <role>participant</role>
            <type>dial-out</type>
          </user>

       </userRequest>
```

Figure 13: Add User Example

The response to this request is shown below.

```
      <userResponse xmlns="urn:ietf-params:xml:ns:xcon:ccmp">
        <operation>change</operation>
        <responseCode> success </responseCode>
        <confObjID> xcon:confxyz987@example.com </confObjID>
        <confUserID> userC-confxyz987 </confUserID>

                           <user entity="sip:claire@example.com">
                   <role>participant</role>
                   <type><dial-out/></type>
              </user>
            </users>
          </conference-info>

      </userResponse>
```

Figure 14: Add User Response Example

12. XML Schema

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

Editor's Note: the schema currently matches the prototype - it needs updating to include changes/additions to request names (e.g., optionsRequest -> blueprintsRequest, addition of blueprintRequest and confsRequest.

```
   <?xml version="1.0" encoding="utf-8"?>
      <xs:schema
          targetNamespace="urn:ietf:params:xml:ns:xcon:ccmp"
          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:xs="http://www.w3.org/2001/XMLSchema">

          <!-- Import data model schema (as per the latest draft) -->
          <xs:import
              namespace="urn:ietf:params:xml:ns:xcon-conference-info"
              schemaLocation="DataModel-11.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:attribute name="xconURI" type="xs:string"
                                      use="optional" />
   </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:attribute name="xconURI" type="xs:string"
                                      use="optional" />
            </xs:complexType>

           <!-- Definition of ccmp-request-message-type as an
                        abstract complex type -->

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

```
         <xs:element name="confUserID" 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>

                   <!-- 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>

     <!-- 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>

    <!-- 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>

      <!-- [TODO: sidebarRequest -->

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

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

     <!-- 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>

       <!-- 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>

        <!-- 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>

       <!-- 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>

       <!-- [TODO: sidebarResponse -->


       <!-- 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="operationNotAllowed"/>
              <xs:enumeration value="deleteFailedParent"/>
              <xs:enumeration value="modifyFailedProtected"/>
              <xs:enumeration value="requestTimeout"/>
              <xs:enumeration value="serverInternalError"/>
              <xs:enumeration value="notImplemented"/>
            </xs:restriction>
          </xs:simpleType>

     <!-- blueprintsResponse -->

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

          <xs:complexType name="blueprintsResponseType">
            <xs:sequence>
               <xs:element ref="namespace"
                                minOccurs="1"
                                maxOccurs="unbounded" />
            </xs:sequence>
          </xs:complexType>

          <xs:element name="namespace">
                  <xs:simpleType>
                          <xs:restriction base="xs:string" />
                  </xs:simpleType>
          </xs:element>


         <!-- confsResponse -->

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

          <xs:complexType name="confsResponseType">
            <xs:sequence>
               <xs:element ref="namespace"
                           minOccurs="1"
                           maxOccurs="unbounded" />
            </xs:sequence>
          </xs:complexType>
```

```
           <!-- confRequest -->

           <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="dm:conference-info"
                            minOccurs="0"/>
             </xs:sequence>
           </xs:complexType>

          <!-- confResponse -->

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

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

           <!-- userRequest -->

           <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="dm:user"
                            minOccurs="0"/>
             </xs:sequence>
           </xs:complexType>
```

```
          <!-- userResponse -->

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

             <xs:complexType name="userResponseType">
             <xs:sequence>
               <xs:element name="operation"
                           type="operationType"
                           minOccurs="1"
                           axOccurs="1" />
               <xs:element name="userInfo"
                           type="dm:conference-info"
                           minOccurs="0"/>
             </xs:sequence>
           </xs:complexType>

      <!-- usersRequest -->

           <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="dm:users"
                            minOccurs="0"/>
             </xs:sequence>
           </xs:complexType>

          <!-- confResponse -->

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

             <xs:complexType name="usersResponseType">
             <xs:sequence>
               <xs:element name="operation"
                           type="operationType"
                           minOccurs="1"
                           maxOccurs="1" />
               <xs:element name="usersInfo"
                           type="dm:users"
                           minOccurs="0"/>
```

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

            <!-- operationType -->

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

Figure 15

13. 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.

14. Role based access control

Editors' Note: this section is also under construction. This topic is planned to be described in a separate document that will be reference here. XACML is the current proposed direction for which the authors would like feedback.

15. 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.

15.1. 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
```

15.2. 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 12 of this document.

15.3. 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.

15.4. DNS Registrations

Section 15.4.1 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 15.4.2, is used to identify an XCON server that understands the CCMP protocol.

15.4.1. **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

15.4.2. 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

15.5. 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.

15.5.1. CCMP Message Types

The CCMP messages are described in Section 8 and defined in the XML schema in Section 12. 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.

15.5.2. 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 8.3:

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.

16. 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).

17. Acknowledgments

The authors appreciate the feedback provided by Dave Morgan, Pierre Tane, Lorenzo Miniero and Tobia Castaldi

18. 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 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.

19. References

19.1. 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.

19.2. 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., Moreau, J., and H. Nielsen, "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]
Moreau, J., Mendelsohn, N., Hadley, M., Nielsen, H., and M. Gudgin, "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>.

Authors' Addresses

Mary Barnes
Nortel
2201 Lakeside Blvd
Richardson, TX

Email: mary.barnes@nortel.com

Chris Boulton
Avaya
Building 3
Wern Fawr Lane
St Mellons
Cardiff, South Wales CF3 5EA

Email: cboulton@avaya.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