XCON
Internet-Draft
Expires: July 7, 2005

H. Schulzrinne
Columbia U.

January 3, 2005

COMP: Conference Object Manipulation Protocol
draft-schulzrinne-xcon-comp-00

Status of this Memo

Copyright Notice

Abstract

The Conference Object Manipulation Protocol (COMP) allows to create, change and delete objects related to centralized conferences, including participants, their media and their roles. The protocol relies on web services and SIP event notification as its infrastructure, but can control conferences that use any signaling protocol to invite users.

Table of Contents

1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [1] and indicate requirement levels for compliant implementations.

This document uses the conferencing terminology defined in High-Level Requirements for Tightly Coupled SIP Conferencing [6] and the Framework and Data Model for Centralized Conferencing [9].

2. Introduction

The Conference Object Manipulation Protocol (COMP) allows administrators and authorized participants to create, change and delete multimedia conferences and their attributes. This includes adding and removing participants, changing their roles and privileges, as well as adding and removing media streams and associated end points. Following [9], we model conferences as objects that can be manipulated and that can inherit properties and attribute types from other conferences.

COMP is based on the three fundamental components of a centralized conference: the conference as a whole, users and media.

COMP implements a client-server model. The server is the Conference Control Server defined in the framework [9], while clients can either be signaling end points, such as SIP user agents, or control-only agents that do not contribute media to the conference.

COMP manipulates conferences based on their semantic properties. It does not address creating the user interface. If a user-interface-based design is desired, technologies such as XForms or possibly the combination of techniques popularly known as Ajax is likely to be more appropriate. The user-interface-based approach makes it easy to add new parameters to deployed systems, as the client implementation does not have to understand the meaning of such new parameters. However, it does not lend itself to scripting and other control by non-human users as the naming and semantics of controls is likely to differ among implementations.

COMP attempts to simplify implementation by re-using components developed earlier. It consists of two components, namely a control protocol based on standard remote procedure call techniques and HTTP that creates and changes conference-related objects, and an event notification protocol that notifies interested and authorized entities when conference-related information has changed. It is possible to build client and server implementations easily with

existing commodity tools, without writing parsing code, retransmission algorithms, security implementations or other low-level components. For simple applications, web-retrieval applications are sufficient.

The remote procedure call component utilizes SOAP [5], as that allows the re-use of libraries, servers and other infrastructure and provides a convenient mechanism for the formal definition of protocol syntax, WSDL. To simplify client implementations, implementations SHOULD support GET-based retrieval, as described in Section 3 of the SOAP introduction.

The latter uses the conference event package [2], with extensions. In the future, web services event notification may also be used, but this is left for future study.

It is likely that implementations and future standardization work will add more conference attributes and parameters. There are three types of extensions. The first, simplest type of extension adds elements to the overall conference, 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 options (Section 3.1) mechanism allows clients and servers to exchange their capabilities.

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.

3. Protocol Operations

We first describe the generic behavior of the four core operations on conference-related objects. These operations will perform similarly if new objects are defined later. Since there is a querying mechanism to ascertain the namespaces understood 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.)

3.1 Options

The 'options' operation does not pertain to a particular conference or other conference-related object but rather queries the capabilities of the Conference Control Server as a whole. In this document, the response returns the XML namespaces that the server understands and the namespaces to be used in responses that it requires the client to understand. Future work may add more global capabilities. [TBD: Should this also be made available via an HTTP OPTIONS request?]

3.2 Create

The 'create' operation creates a new object, either a conference node or some other object that is related to a conference object, e.g., a new user within an existing conference.

3.3 Get

The 'get' operation returns the full XML document describing the 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. To simplify operations, the HTTP/SOAP GET method can also be used directly on these URLs, so that simple systems that need to only obtain data about conference objects do not need a full SOAP implementation. Similarly, a PUT operation can be used to create a new objects.

3.4 Change

The 'change' operation updates the object identified. Trying to change a non-existing object is an error, as is trying to change a parameter that is inherited from a protected element.

3.5 Delete

The 'delete' operation removes an object. Trying to delete a conference object that is being referenced by a child object is an error.

4. Conference Control Objects

Conference objects are referenced by unique identifiers, typically URLs, chosen by the conference server. Implementations MAY choose GUIDs for this purpose, but do not have to. The identifiers are opaque to the client.

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.

Each object has four basic operations: create, change, delete and get, as described in Section 3. Object properties that are not explicitly replaced, remain as-is. This approach makes it possible to manipulate objects created by another application even if the manipulating application does not understand all object properties.

To simplify operations, a server treats certain parameters as suggestions, as noted in the object description. If the 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.

4.1 Conference Object

Conferences use the <conference> object defined in [2]. A client MAY add a <parent> element that indicates the parent that it wants the conference to inherit values from. When creating conferences, the conference URIs are only suggestions by the client. To avoid identifier collisions and to conform to local server policy, the server MAY choose different identifiers. These identifiers are returned in the response.

In addition, the conference description MAY contain a <calendar> element, in the iCal format in XML rendition defined in CPL [7] or (preferable, if available as stable reference) xCal [10]. This description indicates when the conference is active. As discussed above, the conference server may be only offer a subset of the dates, indicated by the 203 response.

Sidebars, i.e., conferences made up of a subset of the participants in the main conference, can be set up by creating a new conference that inherits its properties from the main conference.

4.2 Users (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.

It is believed that there is no need to define end points in the conference control mechanism as these are defined at call-in or call-out time.

Note that 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 <language> element is defined in Section 5.6.4 of [2]. The <type> element defines how the caller is to be reached, 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. If the conference is currently active, dial-out users are contacted immediately; otherwise, they are contacted at the start of the conference. <dial-in> is the default.

In many conferences, users dial in if they know the conference URI and an access code shared by all conference participants. We represent this user by a <user> element without entity attribute. Only the (default) type of <dial-in> is permitted for this type of user. The Conference Control Server then creates individual users as users dial in, identified, in the entity attribute, by their call signaling URL, such as their SIP URL, tel URI [8] or similar. In cases where there is no such URI, e.g., because a PSTN caller has blocked caller-ID delivery, the server assigns a locally-unique URI, such as a locally-scoped tel URI.

The system uses the entity identifier or access code to change or delete user elements.

Three examples for user elements are shown below; the second user element is a dial-in user with access code and only listen capability, while the other access code allows full participant access.

```
<user entity="sip:alice@example.com">
  <roles>moderator sending receiving</roles>
  <languages>en</languages>
  <type><dial-in/></type>
  <media>
    <mediagroup status="sendonly">questions</mediagroup>
    <mediagroup status="recvonly">lecture</mediagroup>
  </media>
</user>

<user access-code="12345">
  <roles>passiveParticipant</roles>
</user>

<user access-code="23456">
  <roles>activeParticipant</roles>
</user>
```

4.3 Roles

While the conference package allows to associate a role with each user, it does not offer a mechanism to define those roles. This document provides an initial mechanism to associate roles with a set of associated permitted activities, i.e., rights. An empty list designates no permissions. An initial set of rights is described below:

sending: The user can send media.
receiving: The user can receive media.
changeConference: The user can change conference characteristics.
createConference: The user can create a child conference for this conference.
deleteConference: The user can delete this conference.
getUsers: The user can get the status of conference participants.
addUser: The user can add users to the current conference.
changeUser: The user can change the attributes of existing users in a conference, e.g., add media.
deleteUser: The user has the permission to delete users from the conference, removing them from the conference.
addMedia: The user has the right to add media to a conference.
deleteMedia: The user has the right to delete media from a conference.

moderator: The person can be a floor-control moderator.
designateModerator: Confers the right to designate a new floor moderator.

Note: Muting and unmuting could simply be expressed by changing roles, but moderators and others with multiple roles would then likely require four roles each. In addition, roles cannot be changed by normal users and are likely to be media-specific. Thus, we use the concept of media groups instead.

In addition, there is a media permission list for each role, identified by the media bus or label (see below).

Active participants can always subscribe to conference events and see their own status.

It is expected that the conference roles are defined in conference documents that are then inherited by most locally-defined conferences so that conferences would typically not have to define new roles.

```
<role id="moderator">
  <rights>getUsers moderator addUser</rights>
</role>
```

4.4 Media Groups

The concept of a media bus/floor describes all the media sources that are controlled together and mixed together. If the conference has floor control such as via BFCP, read and write permission are governed by the floor control protocol. If not, the static configuration modified via COMP can be used to control read and write access to media groups. SDP labels [3] are used to identify media streams.

In addition to the name, the media group may also designate rendering properties of the floor. Initially, we designate the grid (e.g., 4x4) for video streams and how many squares the active speaker occupies. For audio streams, the stereo position is expressed as a number from -1 (leftmost) to +1 (rightmost). Additional properties can be added later by extensions from additional namespaces.

```
<mediagroup label="lecture">
  <media>
    <audio max-bandwidth="100">pcmu l16 dvi</audio>
    <video max-bandwidth="200" picture="3x3"
      speaker="4">mjpeg</video>
  </media>
```

```
    </mediagroup>
```

5. Responses and Error Conditions

Specific error conditions are described below, but there are several general conditions that can occur for any object and operation. Errors are described by a combination of a status code and a reason phrase. As in SIP and HTTP, responses contain a three-digit numeric status code and a textual response, possibly in different languages. The numeric status codes are described below. For easy recognition, they correspond to SIP response codes where this makes sense, but the name spaces are otherwise distinct.

Code	Reason phrase	Explanation
200	OK	successful
202	Pending	notification to follow
203	Modified	The object was created, but may differ from the request.
302	Moved temporarily	An object should be referenced by a different identifier.
400	Bad request	
401	Unauthorized	
403	Forbidden	
404	Object not found	
405	Method not allowed	user is not allowed to perform this method, such as 'create', on the object
408	Request timeout	
409	Cannot delete since it is a parent for another node	
410	Cannot change since it is marked as protected	
500	Server internal error	

501	Not implemented	The function or object has not been implemented.

Table 1

Note that the HTTP request may also return its normal status codes, for example, if a particular HTTP method is not available on the server or if HTTP-level authorization failed.

6. Examples

The examples below omits the standard SOAP header and wrappers, i.e., the part below is simply the <body> of the response.

The first example creates a new conference. The conference URIs are proposals by the client to the server; the server makes the final decision as to whether it will honor those requests.

```
<method>create</method>

<object>
  <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>
</object>
```

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.

```
<result>
  <status>200</status>
  <reason>OK</status>
</result>
<response>
[... modified conference description ...]
</response>
```

The request below adds a user to the conference identified by the conference URI.

```
<method conference="http://example.com/conf233">create</method>
<user entity="sip:bob@example.com">
 <roles>receiving</roles>
 <type><dial-out/></type>
</user>
```

7. WSDL

TBD

8. Security Considerations

Access to conference control functionality needs to be tightly controlled to avoid attackers disrupting conferences, adding themselves to conferences or engaging in theft of services. Implementors needs to deploy standard HTTP and SOAP 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).

9. References

9.1 Normative References

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

[2] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Conference State", draft-ietf-sipping-conference-package-12 (work in progress),

July 2005.

[3] Levin, O. and G. Camarillo, "The SDP (Session Description Protocol) Label Attribute", draft-ietf-mmusic-sdp-media-label-01 (work in progress), January 2005.

[4] Yergeau, F., Paoli, J., Sperberg-McQueen, C., Bray, T., and E. Maler, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C REC REC-xml-20040204, February 2004.

[5] Nielsen, H., Mendelsohn, N., Gudgin, M., Hadley, M., and J. Moreau, "SOAP Version 1.2 Part 1: Messaging Framework", W3C REC REC-soap12-part1-20030624, June 2003.

9.2 Informative References

[6] Levin, O. and R. Even, "High-Level Requirements for Tightly Coupled SIP Conferencing", RFC 4245, November 2005.

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

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

[9] Barnes, M., "A Framework and Data Model for Centralized Conferencing", draft-ietf-xcon-framework-02 (work in progress), October 2005.

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

Author's Address

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

Phone: +1 212 939 7004
Email: hgs+xcon@cs.columbia.edu
URI: http://www.cs.columbia.edu

Appendix A. Acknowledgments

This document is based on discussions within the IETF XCON working group. ?? provided helpful comments.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.