XCON BOF
Internet-Draft
Expires: August 16, 2004

R. Mahy
N. Ismail
Cisco Systems, Inc.
February 16, 2004

Media Policy Manipulation in the Conference Policy Control Protocol
draft-mahy-xcon-media-policy-control-01.txt

Abstract

The SIP conferencing framework defines a model for tightly-coupled conferencing signaled via the Session Initiation Protocol (SIP), in which a Conference Policy Control Protocol is used to manipulate policies relevant to a specific conference, such as conference membership policy, authorization policy, and media layout. This document describes a logical model, which can apply to any session setup protocol, to describe media processing in a tightly-coupled conference. It also defines specific protocol semantics and a specific syntax to manipulate that model.

Table of Contents

1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

2. Overview

The SIP conferencing framework [13] defines a model for tightly-coupled conferences setup via SIP [8], in which a Conference Policy Control Protocol is used to manipulate policies which are relevant to a specific conference instance, such as conference membership policy, authorization policy, and media layout. (As discussed later, the bulk of this model is applicable to tightly-coupled conferences accessed using almost any session setup protocol.) While the conference policy control protocol provides many non-media specific functions [4] such as membership policy and authorization policy, this document specifically addresses requirements [3] to manipulate the way in which media in such a conference is selected, combined, and modified. It defines a logical model of media processing using a "media topology graph". By manipulating the graph, authorized users can change the media processing behavior of the mixers associated with a specific conference.

Here we will briefly summarize the terminology used in SIP conferencing framework in protocol-inspecific terms. Each "conference" is an instance of a multi-media conversation which has a unique protocol-specific identifier. Other (optional) identifiers can represent a conference-factory (an identifier which creates new conferences when contacted). Conferences can contain sub-conferences, which have a unique identifier within the conference, and optionally a unique, protocol-specific, external identifier as well. Each conference identifier is managed by a logical role called a focus, which manages session state for all sessions in the conference. The focus is responsible for coordinating media combining through logical mixers. Mixers perform the actual selection and combination operations. A logical Conference Policy server manages creation and deletion of conferences, authorization, conference longevity, and the media layout or topology. In addition, the focus can use protocol-specific notification mechanisms to provide access to a basic roster and changes in media or non-media aspects of conference policy. Finally, the conference policy may be configured such that mixers use the information returned dynamically by Floor control server(s) to affect media selection.

A media topology graph is a loop-free graph which consists of

individual media streams, logical groups of media streams, and functions or "operations" performed on those streams. These elements are typically associated with a specific subconference. A subconference simply defines a context which allows different groups of users to share a media topology and participant roster with a subset of the participants in a conference. Subconferences are defined in the conferencing framework, and are typically used to enable conferencing sidebars. For convenience purposes, subgraphs--called collections--of connected operators can be defined, instantiated, and manipulated just like individual elements. These elements and their properties are described below.

2.1 Streams

In the beginning there were Streams. These are the actual media streams sent and/or received by or on behalf of conference participants. Media streams are typically established when conference participants join a conference and are described by the SDP [9] media lines in the offer/answer [10] exchange between the participants and the focus, or the analogous exchange in other protocols (ex: H.245 [12] logical channel establishment). Within the media topology graph, each stream is described by a media type, direction and at least one identifier. Initially media types considered include audio, video or text. (Other media types can also be considered in the future.) The direction "in" corresponds to streams originating from the conference participants to the conference, and "out" for streams originating from the conference and terminating at the conference participant. Stream identifiers can be network identifiers or aliases. Network identifiers consist of an address family (IPv4 or IPv6), an IP address, and a port number.

Aliases can also be created for any of the streams, either automatically or when created manually. One such automatic alias consists of a participant identifier and a media stream instance (for example, in SDP, either the media stream identification "mid" as specified in RFC3388 [11] or the position of the media line describing the stream in SDP). Another set of automatic aliases can be created automatically when per media line i-lines (description lines) appear in the SDP.

Conference Policy servers provide clients with lists of stream descriptions as part of protocol-specific notification mechanisms such as the SIP conference package [15] and in response to inventory requests as specified in Section 5.3. Clients use the stream identifier that is part of a stream description to associate and connect (or disconnect) a specific stream with a specific group. (Stream identifiers also play an important role in the naming of the logical internal streams which make up the "bundles" described later

in this section.)

Editors Note: The distinction between external streams and internal (logical) streams may be confusing. If this becomes a problem, one or both terms will be renamed.

2.2 Groups and Bundles

Media groups (hereafter just "groups") are created automatically by servers within the context of a sub-conference as specified in Section 5.3 and have a media type and a direction. Input groups take individual streams and aggregate them into a bundle of named streams. Likewise, output groups accept a bundle of named streams, and distribute these as appropriate to individual output streams. One motivation for naming streams in a bundle is described shortly. Also, the process used to distribute output streams is described in the server behavior section. Groups do not connect directly to other groups.

Bundles are a logical concept which represent a set of individually tagged (named) logical streams. Input bundles contain tags which describe which identifier or participant is contributing to a logical stream. Output bundles contain tags which describe which identifiers or participants should receive a logical stream. This distinction allows participants to receive different streams even when their logical description of the topology is the same. For example, in most audio conferences participants do not hear their own input. Most output bundles also contain a default logical stream.

2.3 Operators

Next are Operators. Operators are basic elements that perform simple media operations. They select among media streams, combine streams, or perform other media processing. Each operator has a type, one or more inputs, one logical output, and an optional set of parameters. The type uniquely identifies the operator and specifies the media service offered.

Selection operators typically accept an input bundle and generate an ordered Set of names of logical streams. These sets can be further manipulated by other operators, but typically they are used as input to a mixing or combining operator. Mixing operators typically receive an input bundle and an ordered list and generate an output bundle. Obviously at least one mixer in the topology graph must be present which can switch the orientation of the streams. Other types of mixers may receive one or more output bundles, perform the appropriate content manipulation, and return a bundle which preserves

the sense of the original tags.

For example, the simplest type of mixer is a promiscuous media mux. It receives an input bundle and generates a bundle consisting of a single default stream (all of the original streams appended to each other). In another simple variation, a media mux generates a named output stream in the output bundle which contains all the other output except that of the sender, for each named input stream in the input bundle. Most mixing operations actually combine input streams in some media-specific way (for example: tiling for video). Other types of operators can provide other arbitrary media or set manipulations such as adjust volume, cross-fade, etc. Operators cannot connect directly to input or output streams. Each type of operator defines the semantics of the operation and any parameters. Parameters define aspects of the operator's function that can differ from one instance of the operator to another.

This document defines a set of standard operators (see Section 3). Each standard operator has a unique type registered with IANA and an XML schema describing the operator. Server implementations can support any of the set of standard operators. As well, implementors can define their own operators and operator types. Clients can discover which operators are supported by making inventory requests to the Server. Authorized clients can then instantiate operators using the method specified in Section 5.2.

2.4 Collections

Finally there are Collections. Collections are subgraphs created by connecting different operators together. Each collection can provide a specific, potentially sophisticated, media service. Like operators, a collection has a type that uniquely identifies it and specifies its function. Each collection has one or more inputs, one logical output and an optional set of parameters. As with operators, this specification defines a set of standard collections that offer the most common mixing and switching media functions available. Each standard collection has a unique type that will be registered with IANA and an XML schema describing the collection. Server implementations can support any of the set of standard collections and they can also define their own proprietary collections. Each newly defined collection needs a unique type and a published XML schema. Clients can make inventory requests to Servers to get the set of collections supported by the server. Clients can then instantiate collections using the method specified in Section 5.2. Clients can also make their own collections to provide new media services by using the method specified in Section 4.

Below follows an example diagram of a media topology graph for a

simple audio conference using the default audio collection.

```
                                 Input Streams

                            A     B     C     D     E

                            |     |     |     |     |
                            |     |     |     |     |
                            v     v     v     v     v
                     +-----------------------------------+
                     |                                   |
                     |  Subconference 0 (Main conf)      |
                     |  Audio Input Group                |
                     |                                   |
                     +-----------------------------------+
                                   ||
                                   \/
               .............................................
               :          ||                ||             :
               :   Input  ||       Input    ||             :
               :   Bundle ||       Bundle   ||             :
               :          ||                \/             :
               :          ||       +--------------+        :
               :          ||       |  Speaker     |        :
               :          ||       |  Selection   |        :
               :          ||       |  Operator    |        :
       Default :          ||       |              |        :
       Audio   :          ||       +--------------+        :
       Collec- :          ||          /                    :
        tion   :          ||         /  Ordered List of    :
               :          \/        /          Speakers    :
               :      +---------------+                    :
               :      |  Audio        |                    :
               :      |  MixMinus     |                    :
               :      |  Operator     |                    :
               :      |               |                    :
               :      +---------------+                    :
               :          ||                               :
               :          || Output Bundle                 :
               :          \/                               :
               .............................................
                                    ||
                                    \/
                      +-----------------------------------+
                      |                                   |
                      |  Subconference 0 (Main conf)      |
                      |  Audio Input Group                |
                      |                                   |
```

```
      +-----------------------------------+
             |     |     |     |     |
             |     |     |     |     |
             v     v     v     v     v

             A     B     C     D     E

                  Output Streams
```

2.5 Using these Elements

This document defines numerous standard operators (in Section 3) to facilitate interoperability. Implementors are free to extend this list of operators, and an IANA registration process is defined for this purpose. Note that specific conference servers may (MAY) support as few or as many operators as they choose, however each conference server needs to (MUST) support at least one standard collection per media type (these are defined in Section 4) which the conference server is capable of handling.

Media manipulation is generally media-specific. When a subconference is created, an input group and an output group are automatically created for each media type supported by the conference server, and a specific collection can be instantiated (again, for each media type). Once instantiated, collections are simply a subgraph of operators connected in some specific way. The resulting graph can be modified, attached, detached, and deleted without affecting the collection from which the graph was copied. Note also that more than one collection can be incorporated into the topology graph for a given subconference and media type.

Manipulating the topology graph for a tightly-coupled conference enables a number of useful features, many of which are described in the XCON scenarios [16] and SIP conferencing high-level requirements [14] documents.

For example, noisy participants can be "muted" from a conference by disconnecting their audio from the appropriate input group. Participants can be moved to a sidebar by disconnecting their media streams (some or all of them) and reconnecting them to the input and output groups created for the corresponding subconference. Interaction with floor control [17] is coordinated by including an operator which selects only media streams corresponding to participants who have the appropriate floor. The resulting logical output stream or group of streams can be connected to a suitable filtering, mixing, or combining operator (for example tiling for video).

Obviously, authorization is required to allow manipulation of media topology by multiple parties (participants and non-participants alike). The effects of manipulating the media topology graph can range from simple, benign changes which only affect the participant requesting the change, to complete failure of the conference. Clearly no one-size-fits-all policy can be applied. However it is useful to recognize several different categories or severities of impact.

- connecting and disconnecting your own streams to a group
- connecting and disconnecting another participants streams
- creating subconferences
- instantiating arbitrary operators or collections
- connecting and disconnecting operators and collections to your own groups
- connecting and disconnecting operators and collections which affect an existing conference or subconference

The rest of the functions of the Conference Policy Control Protocol (CPCP for brevity) are mostly orthogonal to media manipulation and so they are described in a separate document [4]. However it is important to mention the interaction between the media topology-specific and other aspects of the policy. Conferences and subconferences can be created and deleted by CPCP. Although not topology dependent, when these are created the media topology will change automatically to reflect this. Also, one participant may wish to invite several other participants to a subconference (sidebar), but the initiating participant may not have permission to change the stream connection properties of all of the participants. In this case, the initiator places the participant in a pending state. This informs the participant that the initiator would like the participant to join the sidebar. Then the participant (or an agent acting on his or her behalf) either makes the requested change to the media topology by connecting his or her streams to the appropriate groups (a media topology task), or removes himself or herself from the pending list (a non-media related task). Finally, in many cases authorized users can set authorization policy related to a variety of aspects of conference policy. While setting these policies is non-media related, many uses of these policies do affect the media topology. Note that because of this separation, it is possible to produce an implementation of CPCP which runs on two separate servers, one responsible for media topology and the other responsible for the balance of conference policy functions.

3. Some Standard Operators

This sections specifies a set of operators that are needed to provide the most common media processing operators used in conferencing today. Each operator performs a specific function. Each type of operator is registerd with IANA and has an XML Schema [7] that defines how to use the operator. Server implementations are free to support any number of these operators (or none of them) as well as define their own operators.

The operators described below are logical operators which are useful for describing conference features. Implementations may use any internal representation which generates externally identical functionality. The formal syntax for using these operators is described in Section 6.

The "audioSelectSpeakers" operator takes an audio input bundle and generates an ordered list of names of streams. This list is ordered by the priority for including them in an audio mix. No specific algorithm is specified for selecting which speakers are the "best", but commercial implementations typically use a combination of last, loudest, and longest speakers. The actual list of selected speakers is dynamically calculated by a conference mixer. A generically vague definition was intentionally chosen to allow most implementations to offer this operator.

The "audioMixMinus" operator takes an audio input bundle and an ordered list of names of streams and generates an audio output bundle. It selects the first <n> of the streams from the ordered list, where <n> is an implementation-specific integer. The output bundle contains a default stream (which mixes all <n> logical streams) and one logical stream for each stream present in the original input bundle which contains a mix of all <n> logical streams except for input streams corresponding to the same participant as that output stream. In general this property of a mixer is called an exclusive property because it causes participant ouputs to be excluded from their own inputs. With these two operators, you can build the default audio collection described in Section 4.1 and illustrated in the figure in Section 2.4.

The "allParticipantsSet" operator takes an input bundle and generates an unordered list of all the stream names which could conceivably contribute to that bundle.

The "videoSelectSpeakers" operator takes an audio input bundle (to determine who is speaking) and generates an ordered list of names of streams. This list is ordered by the priority for including any corresponding video streams in a video mix. Note that at a given

instant the output of videoSelectSpeakers and audioSelectSpeakers may be different. For example, video speaker selection algorithms typically delay their selection to avoid swapping speakers in the presence of noise such as coughs.

The "setIntersection" operator takes an (optionally) ordered list and an unordered list and generates a new list in the same order as the first list. The new list contains the intersection of the members of the two lists.

The "streamMux" operator takes an input bundle and an ordered list of streams, and generates an output bundle where each output stream contains at least <n> and at most <m> of the input streams muxed in priority order. (<n> and <m> are attributes which specify the minimum and maximum number of streams respectively). This operator also takes an attribute which indicates if the operator should include input streams corresponding to the output stream's participant. With these additional four operators you can build the default multipoint video collection described in Section 4.2. A client using these operators directly to create the same effect would follow these steps. (Note that in most cases the correct "connector" to use is implicit from the direction and type of the connection.)

1. Instantiate a streamMux operator with the following parameters: n=1, m=1, exclusive=true.

2. Instantiate an allParticipants operator, a setIntersection operator, and a videoSelectSpeakers operator.

3. Connect the video input group for this conference to the allParticipants operator

4. Connect the audio input group for this conference to the videoSpeakerSelection operator

5. Connect the allParticipants operator to the "unordered" input of the setIntersection operator

6. Connect the videoSelectSpeakers operator to the "ordered" input of the setIntersection operator

7. Connect the video input group for this conference to the streamMux operator

8. Connect the (output of the) setIntersection operator to the streamMux operator

9. Connect the streamMux operator to the video output group for this

conference

The "selectFloorHolders" operator takes an input bundle and a mandatory attribute which names the floor, and generates an unordered list of names of streams which have been granted the named floor. With this additional operator you can build the floor controlled audio collection in Section 4.3 and the floor controlled video collection in Section 4.4.

The "volume" operator takes an audio bundle and generates an audio bundle which has been adjusted to modify the volume of all streams according to the attributes provided. Either a qualitative or quantitative attribute can be provided. The quantitative attribute is an integer percentage compared to the input volume. The qualitative attributes are "normal", "soft", "softer", "very soft", "loud", "louder", and "very loud".

The "audioMix" operator takes in one or more output bundles and generates a new output bundle. This operator preserves tags. In other words, the output bundle contains streams for each member in the intersection of the participants in the input bundles. With these additional two operators, you can build the audio sidebar collection in Section 4.5 which addresses both sidebar and coaching scenarios.

The "tile" operator takes at least one input video bundle and an ordered list of names of streams. It generates a video output bundle where each output stream consists of tiled windows with a fixed orientation and in priority order as described in Appendix A. One attribute to this operator selects the number of tiles, and another selects if the tile operator is an exclusive or non-exclusive mix. If an exclusive operator is chosen, whenever a tile would display the input of the current participant the next video source is selected instead from the ordered list. Bundles can be connected to a specific tile of the tile operator. For example, tile 4 may be connected to a bundle which shows one of the current floor holders, or to a stream corresponding to a named participant in an input bundle. With this additional operator, you can build a fixed tile continuous presence video layout.

Is there anyway to do this with one input bundle and set or list manipulation? Possibly use weighted lists or position-based manipulation? We should be able to use setSubtraction and/or subSets to enable this functionality.

The "autotile" operator dynamically selects a number of tiles between a minimum and maximum number of streams and incorporates them in a tiled layout automatically. Like the tile operator, this operator can be exclusive or non-exclusive and specific bundles may be connected

to specific tiles. With this additional operator, you can build the an automatically tiled continuous presence video layout.

In addition to those operators just listed, future versions of this document will contain additional standard operators. Some other operators for consideration are listed below.

- o textMux
- o textMuxExclusive
- o explicitList
- o explicitWeightedList
- o sortSet
- o setIntersection
- o setAddition
- o setSubtraction
- o subSet
- o volumeWeighted
- o smilLayout (apply a W3C SMIL stylesheet)
- o textStylesheet
- o xsltLayout
- o selectExplicitParticipants
- o containsContributor
- o doesNotContainContributor
- o crossFade
- o invertSet
- o playUrl
- o selectLast
- o selectLoudest

- o selectLongest
- o stereo2mono
- o pan
- o text2speech
- o speech2text
- o speech2gesture
- o speech2signlanguage

4. More about Collections

To create a new collection, a client defines a list of "connectors" which form the interface between the collection and external graphs. These connectors are strongly typed as input or output bundles or sets, and may be further restricted to media type. Then the "interior" subgraph is created by connecting operators and these connectors to each other. It is even possible to make use of existing collections inside a collection, although this makes loop detection more difficult for the server. Once a new collection is defined, the XML description is stored on the conference policy server as a collection template. These are stored in a context completely removed from individual conferences. Templates persist until they are removed.

Collections are instantiated just like operators. In some cases however, the conference policy server may hide the internal structure of a collection. Also, some conference policy servers may choose to implement only collections (individual operators cannot be instantiated). Conference policy server MUST implement at least one standard collection for each media type they support. Of course they MAY implement as many other standard or vendor-specific collections as desired.

Below we list some of these standard collections. For each collection we give a short textual description and describe the media topology subgraph which describes the behavior of that collection.

- o The basicAudioCollection (see Section 4.1)
- o basicMpVideoCollection (see Section 4.2)
- o sidebarAudioCollection (see Section 4.5)

o audioStreamSelectionCollection

o videoStreamSelectionCollection

o basicTextCollection

o textWithStylesheetCollection

o smilLayoutVideoCollection

o stereoAudioCollection

And a subset of these collections which are floor control enabled...

o audioWithFloorControlCollection (see Section 4.3)

o mpVideoWithFloorControlCollection (see Section 4.4)

o audioStreamSelectionWithFloorControlCollection

o videoStreamSelectionWithFloorControlCollection

o textWithFloorControlCollection

o textWithStylesheetWithFloorControlCollection

4.1 The Basic Audio Collection

```
<connectionTemplate name="basicAudioCollection">
  <connectors>
    <connector name="input" type="bundle"
        media="audio" direction="in"/>
    <connector name="output" type="bundle"
        media="audio" direction="out"/>
  </connectors>
  <operators>
    <operator type="audioSelectSpeakers"/>
    <operator type="audioMixMinus"/>
  </operators>
  <connections>
    <connection>
      <from element="connector" name="input"/>
      <to element="operator" type="audioSelectSpeakers"/>
    </connection>
    <connection>
      <from element="connector" name="input"/>
      <to element="operator" type="audioMixMinus"/>
```

```
        </connection>
        <connection>
          <front element="operator" type="audioSelectSpeakers"/>
          <to element="operator" type="audioMixMinus"/>
        </connection>
        <connection>
          <from element="operator" type="audioMixMinus"/>
          <to element="connector" name="output"/>
        </connection>
      </connections>
    </connectionTemplate>
```

4.2 Basic Video MP Collection

```
    <connectionTemplate name="basicMpVideoCollection">
      <connectors>
        <connector name="in.audio" type="bundle"
            media="audio" direction="in"/>
        <connector name="in.video" type="bundle"
            media="video" direction="in"/>
        <connector name="output" type="bundle"
            media="video" direction="out"/>
      </connectors>
      <operators>
        <operator type="allParticipants"/>
        <operator type="videoSelectSpeakers"/>
        <operator type="setIntersection"/>
        <operator type="streamMux" n="1" m="1" exclusive="true"/>
      </operators>
      <connections>
        <connection>
          <from element="connector" name="in.audio"/>
          <to element="operator" type="videoSelectSpeakers"/>
        </connection>
        <connection>
          <from element="connector" name="in.video"/>
          <to element="operator" type="allParticipants"/>
        </connection>
        <connection>
          <from element="connector" name="in.video"/>
          <to element="operator" type="streamMux"/>
        </connection>
        <connection>
          <front element="operator" type="videoSelectSpeakers"/>
          <to element="operator" type="setIntersection"
```

```
              port="ordered"/>
        </connection>
        <connection>
          <front element="operator" type="allParticipants"/>
          <to element="operator" type="setIntersection"
              port="unordered"/>
        </connection>
        <connection>
          <front element="operator" type="setIntersection"/>
          <to element="operator" type="streamMux"/>
        </connection>
        <connection>
          <from element="operator" type="streamMux"/>
          <to element="connector" name="output"/>
        </connection>
      </connections>
    </connectionTemplate>
```

4.3 Basic Audio Collection with Floor Control

OPEN ISSUE: How do we pass parameters (like the name of the floor) into the interior of a collection?

```
    <connectionTemplate name="audioWithFloorControlCollection">
      <connectors>
        <connector name="input" type="bundle"
            media="audio" direction="in"/>
           <parameter name="floor" value="$floor"/>
        <connector name="output" type="bundle"
            media="audio" direction="out"/>
      </connectors>
      <operators>
        <operator type="audioSelectSpeakers"/>
        <operator type="selectFloorHolders" floor="$floor"/>
        <operator type="setIntersection"/>
        <operator type="audioMixMinus"/>
      </operators>
      <connections>
        <connection>
          <from element="connector" name="input"/>
          <to element="operator" type="audioSelectSpeakers"/>
        </connection>
        <connection>
          <from element="connector" name="input"/>
          <to element="operator" type="selectFloorHolders"/>
```

```
      </connection>
      <connection>
        <from element="connector" name="input"/>
        <to element="operator" type="audioMixMinus"/>
      </connection>
      <connection>
        <front element="operator" type="audioSelectSpeakers"/>
        <to element="operator" type="setIntersection"
            port="ordered"/>
      </connection>
      <connection>
        <front element="operator" type="selectFloorHolders"/>
        <to element="operator" type="setIntersection"
            port="unordered"/>
      </connection>
      <connection>
        <front element="operator" type="setIntersection"/>
        <to element="operator" type="audioMixMinus"/>
      </connection>
      <connection>
        <from element="operator" type="audioMixMinus"/>
        <to element="connector" name="output"/>
      </connection>
    </connections>
  </connectionTemplate>
```

4.4 Basic Video Collection with Floor Control

```
  <connectionTemplate name="mpVideoWithFloorControlCollection">
    <connectors>
      <connector name="in.audio" type="bundle"
          media="audio" direction="in"/>
      <connector name="in.video" type="bundle"
          media="video" direction="in"/>
         <parameter name="floor" value="$floor"/>
      <connector name="output" type="bundle"
          media="video" direction="out"/>
    </connectors>
    <operators>
      <operator type="allParticipants"/>
      <operator type="selectFloorHolders" floor="$floor"/>
      <operator type="videoSelectSpeakers"/>
      <operator type="setIntersection" instance="1"/>
      <operator type="setIntersection" instance="2"/>
      <operator type="streamMux" n="1" m="1" exclusive="true"/>
```

```
      </operators>
      <connections>
        <connection>
          <from element="connector" name="in.audio"/>
          <to element="operator" type="videoSelectSpeakers"/>
        </connection>
        <connection>
          <from element="connector" name="in.video"/>
          <to element="operator" type="allParticipants"/>
        </connection>
        <connection>
          <from element="connector" name="in.video"/>
          <to element="operator" type="streamMux"/>
        </connection>
        <connection>
          <front element="operator" type="videoSelectSpeakers"/>
          <to element="operator" type="setIntersection"
              port="ordered" instance="1"/>
        </connection>
        <connection>
          <front element="operator" type="allParticipants"/>
          <to element="operator" type="setIntersection" instance="2"/>
        </connection>
        <connection>
          <front element="operator" type="selectFloorHolders"/>
          <to element="operator" type="setIntersection" instance="2"/>
        </connection>
        <connection>
          <front element="operator" type="setIntersection" instance="2"/>
          <to element="operator" type="setIntersection"
              port="unordered" instance="1"/>
        </connection>
        <connection>
          <front element="operator" type="setIntersection" instance="1"/>
          <to element="operator" type="streamMux"/>
        </connection>
        <connection>
          <from element="operator" type="streamMux"/>
          <to element="connector" name="output"/>
        </connection>
      </connections>
    </connectionTemplate>
```

4.5 Sidebar Audio Collection

```
<connectionTemplate name="sidebarAudioCollection">
  <connectors>
    <connector name="in.thisconf" type="bundle"
        media="audio" direction="in"/>
    <connector name="in.mainconf" type="bundle"
        media="audio" direction="in"/>
       <parameter name="volume" value="$vol"/>
    <connector name="output" type="bundle"
        media="audio" direction="out"/>
  </connectors>
  <operators>
    <operator type="volume" level="$vol"/>
    <operator type="audioSelectSpeakers"/>
    <operator type="audioMixMinus"/>
    <operator type="audioMix"/>
  </operators>
  <connections>
    <connection>
      <from element="connector" name="in.thisconf"/>
      <to element="operator" type="audioSelectSpeakers"/>
    </connection>
    <connection>
      <from element="connector" name="in.mainconf"/>
      <to element="operator" type="volume"/>
    </connection>
    <connection>
      <front element="operator" type="audioSelectSpeakers"/>
      <to element="operator" type="audioMixMinus"/>
    </connection>
    <connection>
      <front element="operator" type="audioMixMinus"/>
      <to element="operator" type="audioMix"/>
    </connection>
    <connection>
      <front element="operator" type="volume"/>
      <to element="operator" type="audioMix"/>
    </connection>
    <connection>
      <from element="operator" type="audioMix"/>
      <to element="connector" name="output"/>
    </connection>
  </connections>
</connectionTemplate>
```

5. Semantics

5.1 Transactions

Manipulations of a "live" media topology graph are performed as transactions. This insures that the media graph transitions from one consistent state to another. It should never be in a partially connected or disconnected state. Loop detection is always performed by the server before a transaction is accepted.

Note that operators are automatically deleted unless they have at least one input connection and at least one output connection. As a result, a transaction which instantiates an operator must connect it to an input source and an output source during the same transaction, otherwise adding the operator would have no effect.

A transaction encloses one or more topology graph manipulations which must all succeed or all fail. Within the transaction, individual steps consist of either creating or instantiating elements or connecting them together. Note that there is an important distinction between groups and aliases and collections and operators. Groups and aliases are created (they don't exist before they are created), while collections and operators are instantiated (a copy of the original is placed in the media topology graph).

While nearly any RPC-style protocol could be used to express media policy transactions, this document describes an XCAP [2] profile for manipulating media policy. XCAP is a usage of HTTP [5] which uses XPath [6] to address fragments of an XML document in the Request URI. Two XML schemas are defined--one for managing collections for later use, and another for real-time manipulation of media policy graphs.

> Note that support for transactions is currently an open issue in XCAP.

5.2 Client Behavior

To query the media policy for a particular conference, a client merely fetches the media policy document (or document fragment) of interest. In some cases the document will be filtered to remove hidden or private information. Similarly, if the client is authorized, it can view the internal structure of a collection template by just fetching its definition document. When filtered, a collection template may just describe the connectors associated with it and a textual description.

A client connects a stream to a group merely by writing the stream

into the appropriate group structure in the target conference or subconference. Likewise a client disconnects a stream by deleting the stream from the appropriate group structure. The client permissions determine if this request fails, requires confirmation from the affected target, or succeeds immediately. Since a stream can only exist in one group at a time, if a write operation succeeds and the stream is already connected it results in a reassignment rather than the same stream in multiple groups.

To instantiate a new operator or collection, just append an XML fragment of code which describes the parameters for that operator to the appropriate XPath (the operators or collections XPath). To make a connection, just append the appropriate XML fragment describing that connection to the connections XPath. Deleting an XPath, removes the operation, collection, or connection. Once an connection is removed this may cause one or more operations to be automatically deleted. Likewise, when an operation is deleted, all its connections are deleted as well. Just using these simple mechanisms allow authorized clients to perform arbitrary manipulations of the media topology.

Finally, to create a new collection, the client writes an XML description of the collection into the collectionTemplates XPath.

5.3 Server Behavior

Servers must maintain a list of all operator and collection types that can be used by Clients within a conference. Servers must return such a list to all authorized Clients in response to inventory queries. For operators and collections that have parameters, a list of acceptable parameter values must also be specified for each parameter.

For each transaction received by the Server it must proceed with the steps that follow. For each request within the transaction the Server must verify that the party initiating the request is authorized to initiate this specific request in the context of the sub-conference specified within the request. If the initiator is not authorized, the Server must not execute any part of the transaction and return the appropriate "Authorization Failure" response to the initiator. An example if user A requests to connect the input audio stream of user B to group X in sub-conference "sidebar-1" and the output audio stream of user B to group Y in sub-conference "sidebar-1". The Server must verify that user A is authorized to manipulate the media policy of user B and is authorized to manipulate "sidebar-1".

For each request the Server must verify that any changes in the media policy of any participant as a result of the execution of the request is authorized by the conference policy. If any party is not

authorized for the media policy changes that result from the execution of any request within the transaction then the server must not execute any part of the transaction and return the appropriate "Authorization Failure" response to the initiator. In the example used in the previous point, the Server must verify that user B is authorized to join "sidebar-1".

The Server should verify that all requests to instantiate, create and/or connect elements are conforming to the XML schema and descriptions of the elements. If any request does not conform to the XML schema of the elements that it is operating on then the Server must not execute any part of the transaction and return the appropriate "XML Schema Error" response to the initiator. For example an operator that takes one video input bundle can not be connected to an audio bundle.

The Server should verify that all the relevant mixers have enough resources to perform the actual media processing required as a result of the execution of the transaction. If not enough resources are available the Server must not execute any part of the transaction and return the appropriate "No Available Resources" response to the initiator. Note that resources needed for trans-coding and trans-rating should be accounted for. Editor Note: More details and some examples need to be provided to explain this section and specifically the last bullet.

5.4 Notifications of media policy changes

Media topology changes should result in an appropriate protocol-specific notification to those (authorized) parties who have requested (subscribed for) them. In the case of SIP, this notification will be a notification from the SIP conference package, but will send an application/media-policy+xml MIME type in the notification body in addition to, or instead of the basic roster information normally provided by that event package. Note that the protocol should allow hidden transactions for which no notifications will be sent as a result of the media policy change.

Editors Note: Need to describe how pending operations are handled with notifications.

6. Formal Syntax

Below is an XCAP encoding (using XML Schema) for media-topology manipulation of an active conference (or subconference):

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

```
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="media-policy">
         <xs:complexType>
            <xs:sequence>
               <xs:element maxOccurs="1" minOccurs="0"
                           name="groups" type="groupsType"/>
               <xs:element maxOccurs="1" minOccurs="0"
                           name="collections" type="collectionsType"/>
               <xs:element maxOccurs="1" minOccurs="0"
                           name="operators" type="operatorsType"/>
               <xs:element maxOccurs="1" minOccurs="0"
                           name="connections" type="connectionsType"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:complexType name="groupsType">
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0"
                        name="group" type="groupType"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="groupType">
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0"
                        name="stream" type="streamType"/>
         </xs:sequence>
         <xs:attribute name="direction" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="in"/>
                  <xs:enumeration value="out"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="media" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="audio"/>
                  <xs:enumeration value="video"/>
                  <xs:enumeration value="text"/>
                  <!-- add extensibility of the media type? -->
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
      <xs:complexType name="streamType">
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0"
```

```
                        name="alias" type="xs:string"/>
         </xs:sequence>
         <xs:attribute name="fam" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="ipv4"/>
                  <xs:enumeration value="ipv6"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="addr" type="xs:string" use="required"/>
         <xs:attribute name="proto" type="xs:string" use="optional"/>
         <xs:attribute name="sock" type="xs:integer" use="optional"/>
         <xs:attribute name="demux" type="xs:string"/>
      </xs:complexType>
      <xs:complexType name="collectionsType">
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0"
                        name="collection" type="operatorType"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="operatorsType">
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0"
                        name="collection" type="operatorType"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="operatorType">
         <xs:attribute name="type" type="xs:string" use="required"/>
         <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:complexType>
      <xs:complexType name="connectionsType">
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0"
                        name="connection" type="connectionType"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="connectionType">
         <xs:sequence>
            <xs:element maxOccurs="1" minOccurs="1"
                        name="to" type="connectType"/>
            <xs:element maxOccurs="1" minOccurs="1"
                        name="from" type="connectType"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="connectType">
         <xs:attribute name="element" use="required">
            <xs:simpleType>
```

```
              <xs:restriction base="xs:string">
                 <xs:enumeration value="group"/>
                 <xs:enumeration value="collection"/>
                 <xs:enumeration value="operator"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="type" type="xs:string" use="optional"/>
        <xs:attribute name="conf" type="xs:string" use="optional"/>
        <xs:attribute name="media" use="optional">
           <xs:simpleType>
              <xs:restriction base="xs:string">
                 <xs:enumeration value="audio"/>
                 <xs:enumeration value="video"/>
                 <xs:enumeration value="text"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="direction" use="optional">
           <xs:simpleType>
              <xs:restriction base="xs:string">
                 <xs:enumeration value="in"/>
                 <xs:enumeration value="out"/>
              </xs:restriction>
           </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="port" type="xs:string" use="optional"/>
        <xs:attribute name="instance" type="xs:string" use="optional"/>
     </xs:complexType>
  </xs:schema>
```

And here is an XML schema for describing collection templates:

```
     <?xml version="1.0" encoding="UTF-8"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
     <xs:element name="collectionTemplates">
        <xs:complexType>
           <xs:sequence>
              <xs:element maxOccurs="1" minOccurs="0"
                  name="connectors" type="connectorsType"/>
              <xs:element maxOccurs="1" minOccurs="0"
                  name="collections" type="collectionsType"/>
              <xs:element maxOccurs="1" minOccurs="0"
                  name="operators" type="operatorsType"/>
              <xs:element maxOccurs="1" minOccurs="0"
                  name="connections" type="connectionsType"/>
```

```
          </xs:sequence>
          <xs:attribute name="name" type="xs:string"/>
       </xs:complexType>
    </xs:element>
    <xs:complexType name="connectorsType">
       <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0"
              name="connector" type="connectorType"/>
       </xs:sequence>
    </xs:complexType>
    <xs:complexType name="connectorType">
       <xs:attribute name="name" use="required"/>
       <xs:attribute name="type" use="required">
          <xs:simpleType>
             <xs:restriction base="xs:string">
                <xs:enumeration value="bundle"/>
                <xs:enumeration value="set"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="direction" use="required">
          <xs:simpleType>
             <xs:restriction base="xs:string">
                <xs:enumeration value="in"/>
                <xs:enumeration value="out"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:attribute>
    </xs:complexType>
    <xs:complexType name="collectionsType">
       <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0"
              name="collection" type="operatorType"/>
       </xs:sequence>
    </xs:complexType>
    <xs:complexType name="operatorsType">
       <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0"
              name="collection" type="operatorType"/>
       </xs:sequence>
    </xs:complexType>
    <xs:complexType name="operatorType">
       <xs:attribute name="type" type="xs:string" use="required"/>
       <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
    <xs:complexType name="connectionsType">
       <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0"
```

```
                name="connection" type="connectionType"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="connectionType">
         <xs:sequence>
            <xs:element maxOccurs="1" minOccurs="1"
                name="to" type="connectType"/>
            <xs:element maxOccurs="1" minOccurs="1"
                name="from" type="connectType"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="connectType">
         <xs:attribute name="element" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="connector"/>
                  <xs:enumeration value="collection"/>
                  <xs:enumeration value="operator"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="type" type="xs:string" use="optional"/>
         <xs:attribute name="conf" type="xs:string" use="optional"/>
         <xs:attribute name="media" use="optional">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="audio"/>
                  <xs:enumeration value="video"/>
                  <xs:enumeration value="text"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="direction" use="optional">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="in"/>
                  <xs:enumeration value="out"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="port" type="xs:string" use="optional"/>
         <xs:attribute name="instance" type="xs:string" use="optional"/>
      </xs:complexType>
   </xs:schema>
```

7. Examples

Below is a diagram which shows a sample media topology (with streams, collections, and groups) for an audio and video conference with an audio sidebar.

```
        Audio and Video Conference with one Audio Sidebar

           (streams)              (streams)                (streams)

       A B   D E F  H  J      A   C D   F G H I           B   E   J
       | |   | | |  |  |      |   | |   | | | |           |   |   |
       | |   | | |  |  |      |   | |   | | | |           |   |   |
       V V   V V V  V  V      V   V V   V V V V           V   V   V
     +------------------+   +------------------+   +-------------------+
     | Main Video In    |   | Main Audio In    |   | Sidebar Audio Out |
     |  (group)         |   | (group)          |   | (group)           |
     +------------------+   +------------------+   +-------------------+
              ||            //        ||                      ||
              ||           //         ||          +------+    ||
              ||          //          ||          |+----+|    ||
              ||         //           ||          ||    ||    ||
              \/        //            \/          ||    \/    \/
     ...................V.   ...................  ||  ..................
     :                   :   :                 :  ||  :                :
     :                   :   :                 :  ||  :                :
     :   vendor          :   :   standard      :  ||  :   standard     :
     :   defined         :   :   conference    :  ||  :   sidebar      :
     :   video           :   :   audio         :  ||  :   audio        :
     :   collection      :   :   collection    :  ||  :   collection   :
     :                   :   :                 :  ||  :                :
     :                   :   :                 :  ||  :                :
     .....................   ...................  ||  ..................
               ||                     ||     ||   ||          ||
               ||                     ||     |+---+|          ||
               ||                     ||     +-----+          ||
               \/                     \/                      \/
     +------------------+   +------------------+   +-------------------+
     | Main Video Out   |   | Main Audio Out   |   | Sidebar Audio Out |
     | (group)          |   | (group)          |   | (group)           |
     +------------------+   +------------------+   +-------------------+
       | | | | | |  |  |       |  | |  | | | |             |   |   |
       | | | | | |  |  |       |  | |  | | | |             |   |   |
       V V V V V V  V  V       V  V V  V V V V             V   V   V
       A B C D E F  H  J       A  C D  F G H I             B   E   J

           (streams)              (streams)                (streams)
```

Here we have the media topologies description documents for the combined audio/video conference in the figure above. The first media topology is for the main conference, and the second is for the subconference used by the audio sidebar. Specific streams are omitted for brevity.

```
<media-topology>
  <groups>
    <group dir="in" media="audio"/>
    <group dir="out" media="audio"/>
    <group dir="in" media="video"/>
    <group dir="out" media="video"/>
  </groups>
  <collections>
    <collection type="basicAudioCollection"/>
    <collection type="example.com.videoCollection" size="7"/>
  </collections>
  <connections>
    <connection>
      <from element="group" direction="in" media="audio"/>
      <to element="collection" type="basicAudioCollection"/>
    </connection>
    <connection>
      <from element="group" direction="in" media="video"/>
      <to element="collection" type="example.com.videoCollection"/>
    </connection>
    <connection>
      <from element="collection" type="basicAudioCollection"/>
      <to element="group" direction="out" media="audio"/>
    </connection>
    <connection>
      <from element="collection" type="example.com.videoCollection"/>
      <to element="group" direction="out" media="video"/>
    </connection>
  </connections>
</media-topology>
```

Below is the media topology description document for the subconference. Note that conf=".." refers to the parent of the current conference

```
<media-topology>
  <groups>
    <group dir="in" media="audio"/>
    <group dir="out" media="audio"/>
  </groups>
  <collections>
    <collection type="sidebarAudioCollection" volume="soft"/>
```

```
    </collections>
    <connections>
      <connection>
        <from element="group" direction="in" media="audio"/>
        <to element="collection" type="sidebarAudioCollection"
            port="in.thisconf"/>
      </connection>
      <connection>
        <from element="group" direction="out" media="audio" conf=".."/>
        <to element="collection" type="sidebarAudioCollection"
            port="in.mainconf"/>
      </connection>
      <connection>
        <from element="collection" type="sidebarAudioCollection"/>
        <to element="group" direction="out" media="audio"/>
      </connection>
    </connections>
  </media-topology>
```

8. Security Considerations

Much needs to be written here. Authorization rules will be discussed in Section 5.3. Privacy and filtering rules will be discussed there as well.

9. IANA Considerations

This document defines an IANA registry of Media Operators, and another of Media Collections.

10. Acknowledgments

This work was the result of discussions among the SIP Conferencing Design Team.

Normative References

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

[2] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft-rosenberg-simple-xcap-00 (work in progress), May 2003.

[3] Even, R., Levin, O. and N. Ismail, "Conferencing media policy Requirements", draft-even-xcon-media-policy-requirements-00.txt (work in progress), June 2003.

[4] Koskelainen, P. and H. Khartabil, "XCAP Usage for Conference Policy Manipulation", draft-koskelainen-xcon-xcap-cpcp-usage-00.txt (work in progress), June 2003.

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

[6] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", W3C Recommendation xpath, November 1999, <http://www.w3.org/TR/xpath>.

[7] Thompson, H., Beech, D., Maloney, M. and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC-xmlschema-1, May 2001, <http://www.w3.org/TR/xmlschema-1/>.

[8] 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.

[9] Jacobson, V., Perkins, C. and M. Handley, "SDP: Session Description Protocol", draft-ietf-mmusic-sdp-new-13 (work in progress), May 2003.

[10] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[11] Camarillo, G., Eriksson, G., Holler, J. and H. Schulzrinne, "Grouping of Media Lines in the Session Description Protocol (SDP)", RFC 3388, December 2002.

[12] International Telecommunications Union, "CONTROL PROTOCOL FOR MULTIMEDIA COMMUNICATION", ITU Recommendation H.245, 1998.

Informational References

[13] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-00 (work in progress), May 2003.

[14] Levin, O. and R. Even, "High Level Requirements for Tightly Coupled SIP Conferencing", draft-ietf-sipping-conferencing-requirements-00 (work in progress), April 2003.

[15] Rosenberg, J. and H. Schulzrinne, "A Session Initiation

Protocol (SIP) Event Package for Conference State", draft-ietf-sipping-conference-package-00 (work in progress), June 2002.

[16] Even, R. and N. Ismail, "Conferencing Scenarios", draft-even-xcon-conference-scenarios-00.txt (work in progress), June 2003.

[17] Koskelainen, P., "Floor Control Requirements", draft-koskelainen-xcon-floor-control-reqs-00.txt (work in progress), June 2003.

Authors' Addresses

Rohan Mahy
Cisco Systems, Inc.
5617 Scotts Valley Drive
Scotts Valley, CA 95066
USA

EMail: rohan@cisco.com

Nermeen Ismail
Cisco Systems, Inc.
170 W Tasman Dr
San Jose, CA 95134
USA

EMail: nismail@cisco.com

Appendix A. Standard Tile Order

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

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