Network Working Group	A. Amirante	
Internet-Draft	T. Castaldi	
Expires: May 7, 2009	L. Miniero	
	S P. Romano	
	University of Napoli	
	November 03, 2008	

TOC

Media Control Channel Framework (CFW) Call Flow Examples
draft-miniero-mediactrl-escs-03

Status of this Memo

Abstract

This document provides with a list of more or less detailed Media Control Channel Framework [I-D.ietf-mediactrl-sip-control-framework] (Boulton, C., Melanchuk, T., and S. McGlashan, “Media Control Channel Framework,” October 2009.) call flows. It aims at being a simple guide throughout the use of the interface between Application Servers and MEDIACTRL-based Media Servers, as well as a hopefully helpful base reference documentation for both implementors and protocol researchers.

Table of Contents

---

1. Introduction

TOC

TBD. Discussion upon SIP/MEDIACTRL and separation of responsibilities between Application Servers (application logic) and Media Servers (media management and manipulation).
Requirements -> Architecture -> Framework (Control Packages)

---

2. Conventions

TOC

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] (Bradner, S., “Key words for use in RFCs

to Indicate Requirement Levels," March 1997.) and indicate requirement levels for compliant implementations.
Besides, note that due to RFC formatting conventions, this document often splits SIP/SDP and CFW across lines whose content would exceed 72 characters. A backslash character marks where this line folding has taken place. This backslash and its trailing CRLF and whitespace would not appear in the actual protocol contents.

---

3. Terminology

TOC

This document pretty much makes use of the same terminology as the one that can be found in the referenced documents. The following terms are only a summarization of the most commonly used ones in this context, mostly derived from the terminology used in the related documents:

Application Server: an entity that requests media processing and manipulation from a Media Server; typical examples are Back to Back User Agents (B2BUA) and endpoints requesting manipulation of a third-party's media stream.

Media Server: an entity that performs a service, such as media processing, on behalf of an Application Server; typical provided functionality are mixing, announcement, tone detection and generation, and play and record services.

Control Channel: a reliable connection between an Application Server and a Media Server that is used to exchange Framework messages.

---

4. Overview

TOC

This document provides with a list of more or less detailed MEDIACTRL Media Control Channel Framework [I-D.ietf-mediactrl-sip-control-framework] (Boulton, C., Melanchuk, T., and S. McGlashan, "Media Control Channel Framework," October 2009.) call flows. The motivation for this comes from our implementation experience with the framework and its protocol. This drove us to writing what could be both a simple guide throughout the use of the several interfaces between Application Servers and MEDIACTRL-based

Media Servers (and the related correlations between them) and a hopefully helpful base reference documentation for other implementors and protocol researchers.
Following this spirit, this document covers several aspects of the interaction between Application Servers and Media Servers. However, in the context of this document, the call flows almost always depict the interaction between a single Application Server (which, for the sake of conciseness, is called AS from now on) and a single Media Server (MS). To ease up the understanding of all the flows (for what concerns both SIP dialogs and CFW transactions), the domains hosting the AS and the MS in all the scenarios are called, respectively, 'cicciopernacchio.com' and 'pippozzoserver.org'.
In the next paragraphs a small overview of our implementation approaches and choices is described, with particular focus upon the protocol-related aspects. This involves state diagrams for what concerns both the client side (the AS) and the server side (the MS). Of course, this section is not at all to be considered a mandatory approach to the implementation of the framework. It is only meant to ease up the understanding of how the framework works from a practical point of view, and of the following examples.
Once done with this preliminary considerations, in the subsequent sections real-life scenarios are faced. In this context, first of all, the establishment of the Control Channel is dealt with: after that, some typical use case scenarios, involving the most typical multimedia applications, are depicted and described.

4.1. A Practical Approach

TOC

TBD. (What exactly is needed here? Implementation detail are not likely to belong in here...)

4.1.1. State Diagrams

TOC

TBD. (talk about both diagrams; explain why both diagrams have been separated considering the introduction of the new MS-generated CONTROL event for notifications; describe how transactions and events are correlated at package level, but not at framework level).

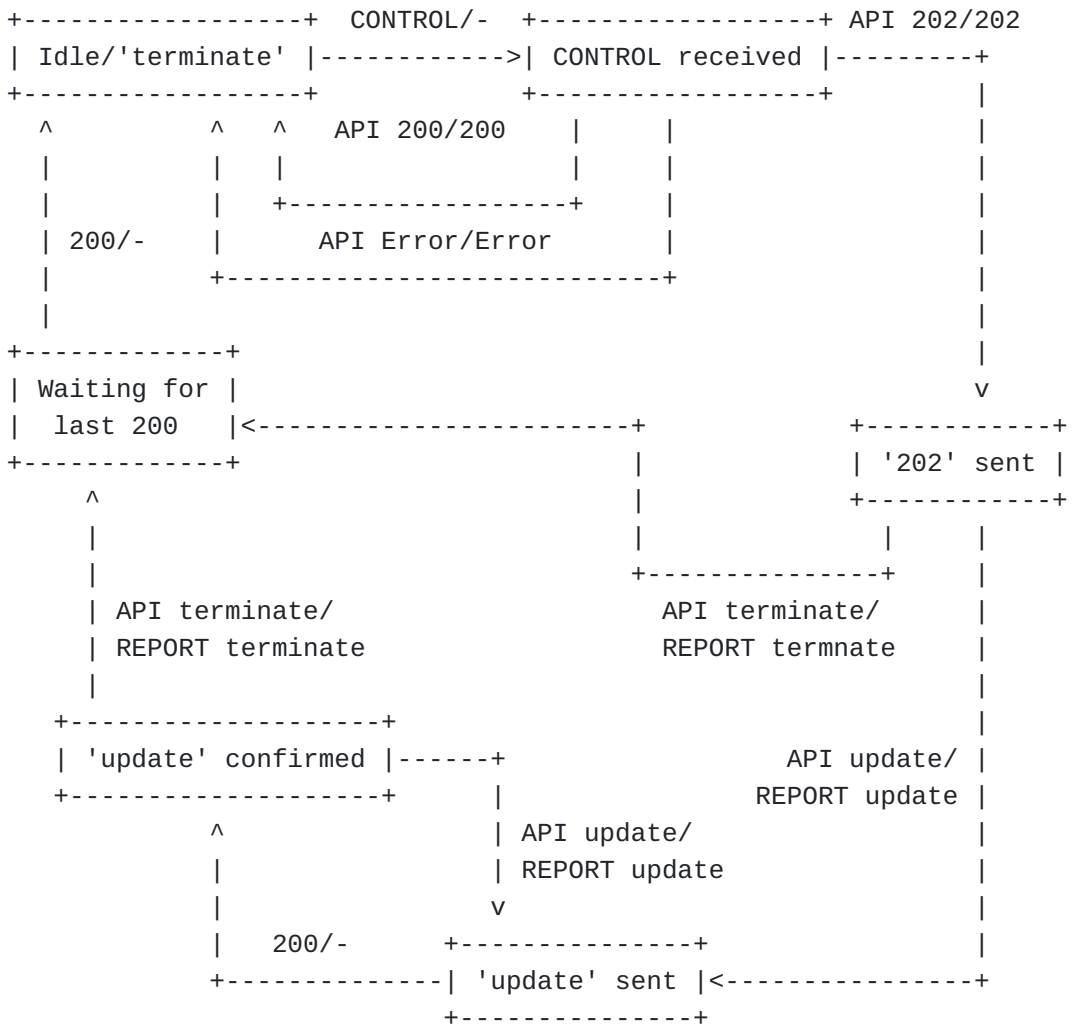


Figure 1: Media Server CFW State Diagram

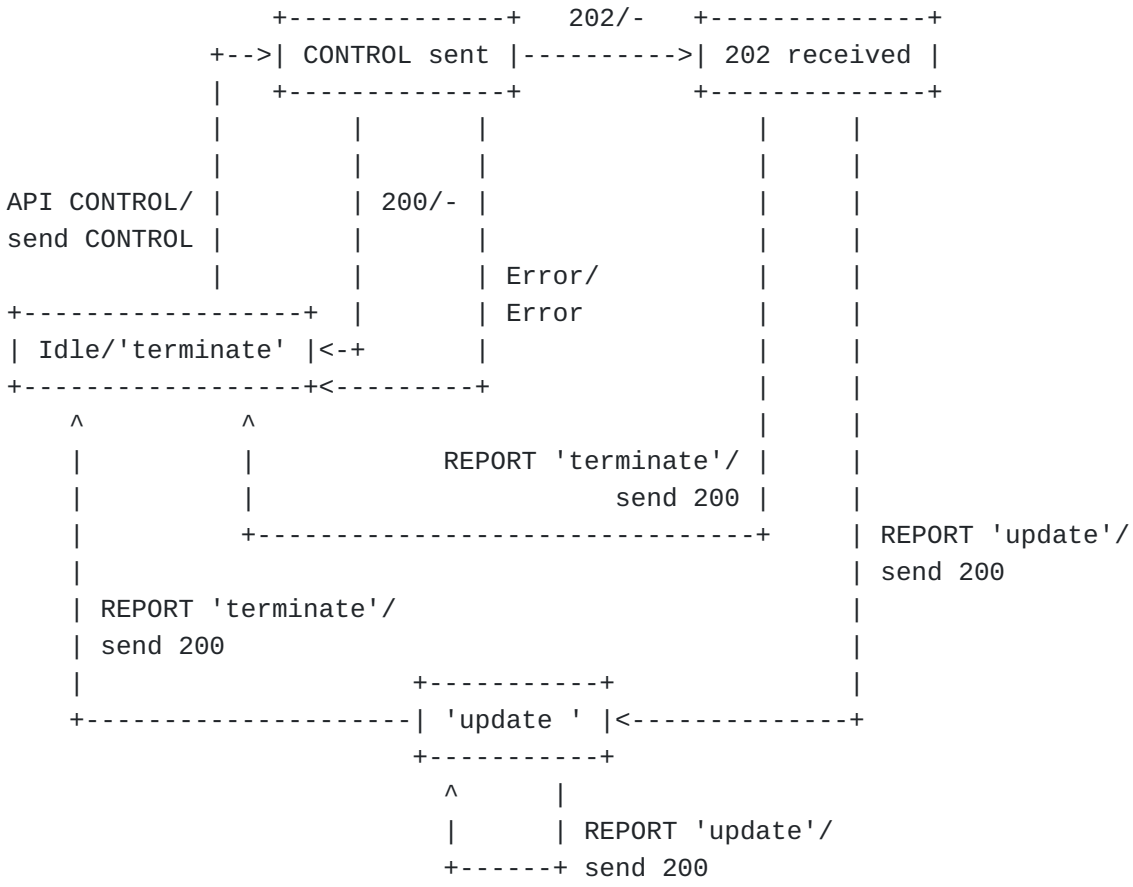


Figure 2: Application Server CFW State Diagram

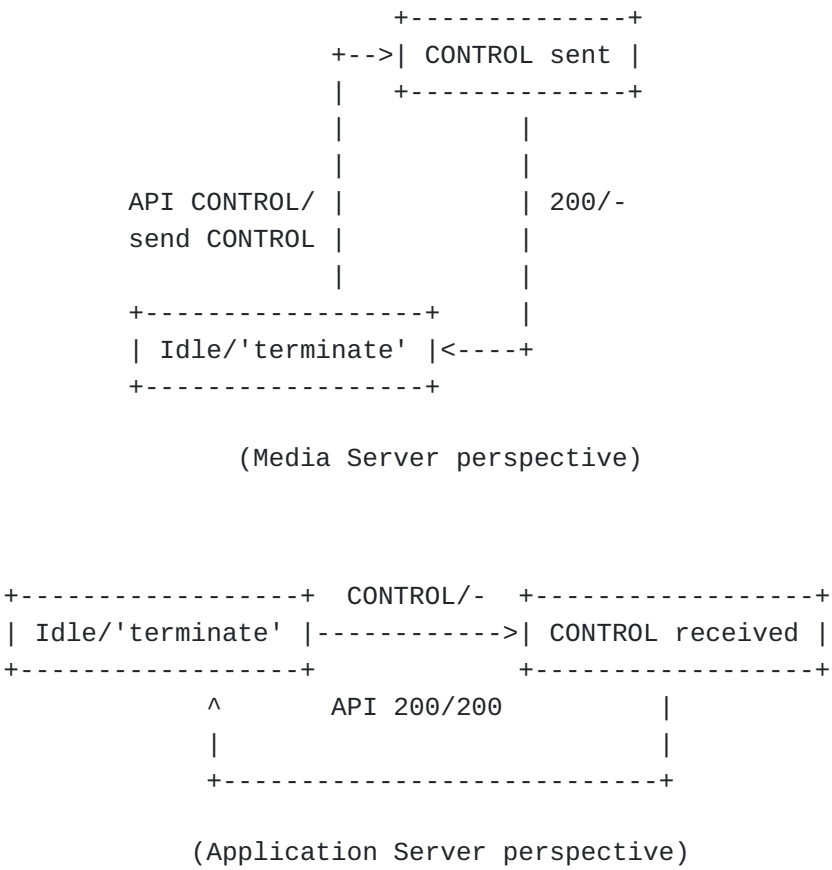


Figure 3: Event Notifications

4.1.2. Implementation

TOC

TBD. (media- and macro-connections, conferences, plugins)

---

5. Control Channel Establishment

TOC

As specified in [I-D.ietf-mediactrl-sip-control-framework] (Boulton, C., Melanchuk, T., and S. McGlashan, "Media Control Channel Framework," October 2009.), the preliminary step to any interaction between an AS and a MS is the establishment of a control channel between the two. As explained in the next subsection, this is accomplished by means of a so-called COMEDIA [RFC4145] (Yon, D. and G. Camarillo, "TCP-Based Media

Transport in the Session Description Protocol (SDP)," September 2005.)
negotiation. This negotiation allows for a TCP connection to be created between the AS and the MS: once they have connected, a SYNC message sent by the AS to the MS consolidates the control channel.

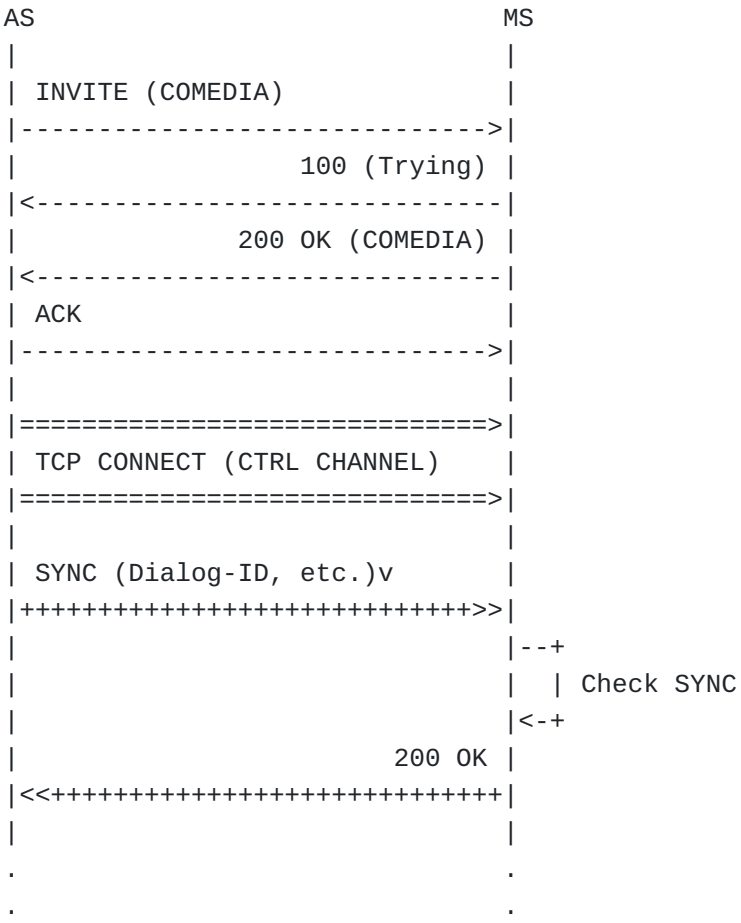


Figure 4: Control Channel Establishment

5.1. COMEDIA Negotiation

TOC

As a first step, the AS and the MS establish a Control SIP dialog. This is usually originated by the AS itself. The AS generates a SIP INVITE message containing in its SDP body information about the TCP connection it wants to establish with the MS. In the provided example (see

Figure 5 (COMEDIA Negotiation: Sequence Diagram) and the attached call flow), the AS wants to actively open a new TCP connection, which on his side will be bound to port 5757. If the request is fine, the MS answers with its own offer, by communicating to the AS the transport address to connect to in order to establish the TCP connection. In the provided example, the MS will listen on the port 7575. Once this negotiation is over, the AS can effectively connect to the MS.
The negotiation includes additional attributes, the most important being the 'cfw-id' attribute, since it specifies the Dialog-ID which will be subsequently referred to by both the AS and the MS, as specified in the core framework draft.
Note that the provided example also includes the indication, from both the AS and the MS, of the supported control packages. This is achieved by means of a series of 'ctrl-package' attributes as specified in [I-D.boulton-mmusic-sdp-control-package-attribute] (Boulton, C., “A Session Description Protocol (SDP) Control Package Attribute,” March 2009.). In the example, the AS supports (or is only interested to) two packages: IVR and the Audio Conferencing. The MS replies with the list of packages it supports, by adding the VoiceXML IVR package to the list provided by the AS. It is worth noting that this exchange of information is not meant as a strictly containing negotiation of packages: in case the AS gets to know that one or more packages it needs are not supported according to the indications sent by the MS, it MAY choose not to open a control channel with the MS at all, if its application logic leads to such a decision. The actual negotiation of control packages is done subsequenty through the use of the framework SYNC transaction.

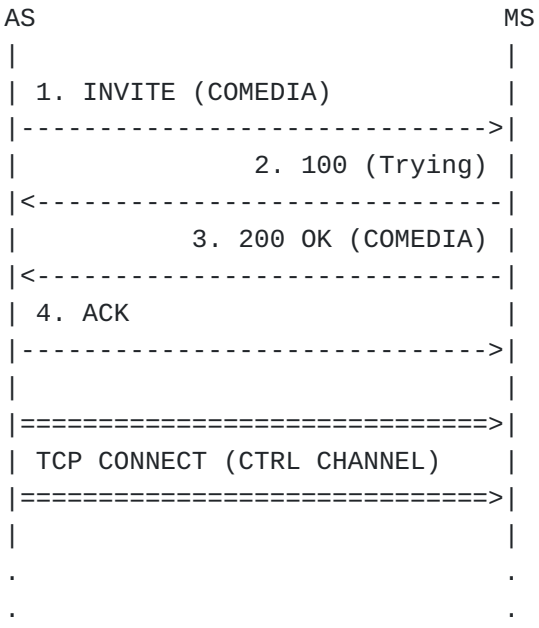


Figure 5: COMEDIA Negotiation: Sequence Diagram

```
1. AS -> MS (SIP INVITE)
------------------------
   INVITE sip:MediaServer@pippozzoserver.org:5060 SIP/2.0
   Via: SIP/2.0/UDP 1.2.3.4:5060;\
           branch=z9hG4bK-d8754z-9b07c8201c3aa510-1---d8754z-;rport=5060
   Max-Forwards: 70
   Contact: <sip:ApplicationServer@1.2.3.4:5060>
   To: <sip:MediaServer@pippozzoserver.org:5060>
   From: <sip:ApplicationServer@cicciopernacchio.com:5060>;tag=4354ec63
   Call-ID: MDk2YTk1MDU3YmVkZjgzYTQwYmJlNjE5NTA4ZDQ1OGY.
   CSeq: 1 INVITE
   Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, INVITE, REGISTER
   Content-Type: application/sdp
   Content-Length: 263

   v=0
   o=lminiero 2890844526 2890842807 IN IP4 cicciopernacchio.com
   s=MediaCtrl
   c=IN IP4 cicciopernacchio.com
   t=0 0
   m=application 5757 TCP/CFW *
   a=connection:new
   a=setup:active
   a=cfw-id:5feb6486792a
   a=ctrl-package:msc-ivr/1.0
   a=ctrl-package:msc-mixer/1.0


2. AS <- MS (SIP 100 Trying)
----------------------------
   SIP/2.0 100 Trying
   Via: SIP/2.0/UDP 1.2.3.4:5060; \
           branch=z9hG4bK-d8754z-9b07c8201c3aa510-1---d8754z-;rport=5060
   To: <sip:MediaServer@pippozzoserver.org:5060>;tag=499a5b74
   From: <sip:ApplicationServer@cicciopernacchio.com:5060>;tag=4354ec63
   Call-ID: MDk2YTk1MDU3YmVkZjgzYTQwYmJlNjE5NTA4ZDQ1OGY.
   CSeq: 1 INVITE
   Content-Length: 0


3. AS <- MS (SIP 200 OK)
------------------------
   SIP/2.0 200 OK
   Via: SIP/2.0/UDP 1.2.3.4:5060; \
           branch=z9hG4bK-d8754z-9b07c8201c3aa510-1---d8754z-;rport=5060
   Contact: <sip:MediaServer@pippozzoserver.org:5060>
   To: <sip:MediaServer@pippozzoserver.org:5060>;tag=499a5b74
```

```
   From: <sip:ApplicationServer@cicciopernacchio.com:5060>;tag=4354ec63
   Call-ID: MDk2YTk1MDU3YmVkZjgzYTQwYmJlNjE5NTA4ZDQ1OGY.
   CSeq: 1 INVITE
   Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, INVITE, REGISTER
   Content-Type: application/sdp
   Content-Length: 329

   v=0
   o=lminiero 2890844526 2890842808 IN IP4 pippozzoserver.org
   s=MediaCtrl
   c=IN IP4 pippozzoserver.org
   t=0 0
   m=application 7575 TCP/CFW *
   a=connection:new
   a=setup:passive
   a=cfw-id:5feb6486792a
   a=ctrl-package:msc-ivr-vxml/1.0
   a=ctrl-package:msc-ivr/1.0
   a=ctrl-package:msc-example-pkg/1.0
   a=ctrl-package:msc-mixer/1.0


4. AS -> MS (SIP ACK)
---------------------
   ACK sip:MediaServer@pippozzoserver.org:5060 SIP/2.0
   Via: SIP/2.0/UDP 1.2.3.4:5060; \
                branch=z9hG4bK-d8754z-22940f5f4589701b-1---d8754z-;rport
   Max-Forwards: 70
   Contact: <sip:ApplicationServer@1.2.3.4:5060>
   To: <sip:MediaServer@pippozzoserver.org:5060>;tag=499a5b74
   From: <sip:ApplicationServer@cicciopernacchio.com:5060>;tag=4354ec63
   Call-ID: MDk2YTk1MDU3YmVkZjgzYTQwYmJlNjE5NTA4ZDQ1OGY.
   CSeq: 1 ACK
   Content-Length: 0
```

5.2. SYNC

TOC

Once the AS and the MS have successfully established a TCP connection, an additional step is needed before the control channel can be used. In fact, as seen in the previous subsection, the first interaction between the AS and the MS happens by means of a SIP dialog, which in turns allows for the creation of the TCP connection. This introduces the need for a proper correlation between the above mentioned SIP dialog and TCP connection, so that the MS can be sure the connection came from the AS which requested it. This is accomplished by means of a dedicated

framework message called SYNC. This SYNC message makes use of a unique identifier called Dialog-ID to validate the control channel. This identifier, as introduced in the previous paragrah, is randomly generated by the caller (the AS in the call flow), and added as an SDP media attribute (cfw-id) to the COMEDIA negotiation in order to make both the entities aware of its value:

```
a=cfw-id:5feb6486792a
         ^^^^^^^^^^^^
```

Besides, it offers an additional negotiation mechanism. In fact, the AS uses the SYNC not only to properly correlate as explained before, but also to negotiate with the MS the control packages it is interested to, as well as to agree on a Keep-Alive timer needed by both the AS and the MS to understand if problems on the connection occur. In the provided example (see Figure 5 (COMEDIA Negotiation: Sequence Diagram) and the related call flow), the AS sends a SYNC with a Dialog-ID constructed as needed (using the 'cfw-id' attribute from the SIP dialog) and requests access to two control packages, specifically the IVR and the Audio Conferencing package (These are the same packages the AS previously indicated in its SDP as specified in [I-D.boulton-mmusic-sdp-control-package-attribute] (Boulton, C., “A Session Description Protocol (SDP) Control Package Attribute,” March 2009.), with the difference that this time they are reported in the context of a binding negotiation). Besides, it instructs the MS that a 100 seconds timeout is to be used for Keep-Alive messages. The MS validates the request by matching the received Dialog-ID with the SIP dialog values and, assuming it supports the control packages the AS requested access to (and for the sake of this document we assume it does), it answers with a 200 message. Additionally, the MS provides the AS with a list of other unrequested packages it supports (in this case the VoiceXML IVR package and a dummy package providing testing functionality).

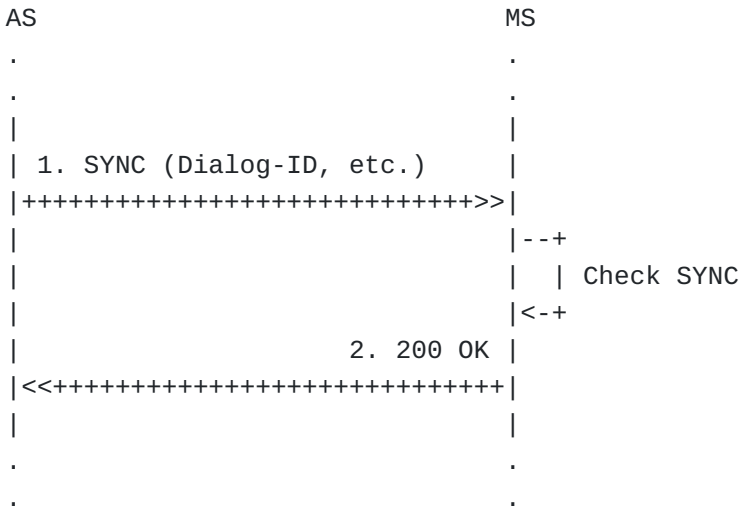


Figure 6: SYNC: Sequence Diagram

```
1. AS -> MS (CFW SYNC)
----------------------
   CFW 6e5e86f95609 SYNC
   Dialog-ID: 5feb6486792a
   Keep-Alive: 100
   Packages: msc-ivr/1.0,msc-mixer/1.0


2. AS <- MS (CFW 200)
---------------------
   CFW 6e5e86f95609 200
   Keep-Alive: 100
   Packages: msc-ivr/1.0,msc-mixer/1.0
   Supported: msc-ivr-vxml/1.0,msc-example-pkg/1.0
```

At this step, the control channel is finally established, and can be used by the AS to request services from the MS.

---

6. Use-case scenarios and examples

TOC

The following scenarios have been chosen for their common presence in many rich real-time multimedia applications. Each scenario is depicted

as a set of call flows, involving both the SIP/SDP signaling (UACs<->AS<->MS) and the Control Channel communication (AS<->MS).
All the examples assume that a Control Channel has already been correctly established and SYNCed between the reference AS and MS. Besides, unless stated otherwise, the same UAC session is referenced in all the above mentioned examples. The UAC session is assumed to have been created as the Figure 7 (3PCC Sequence Diagram) describes:

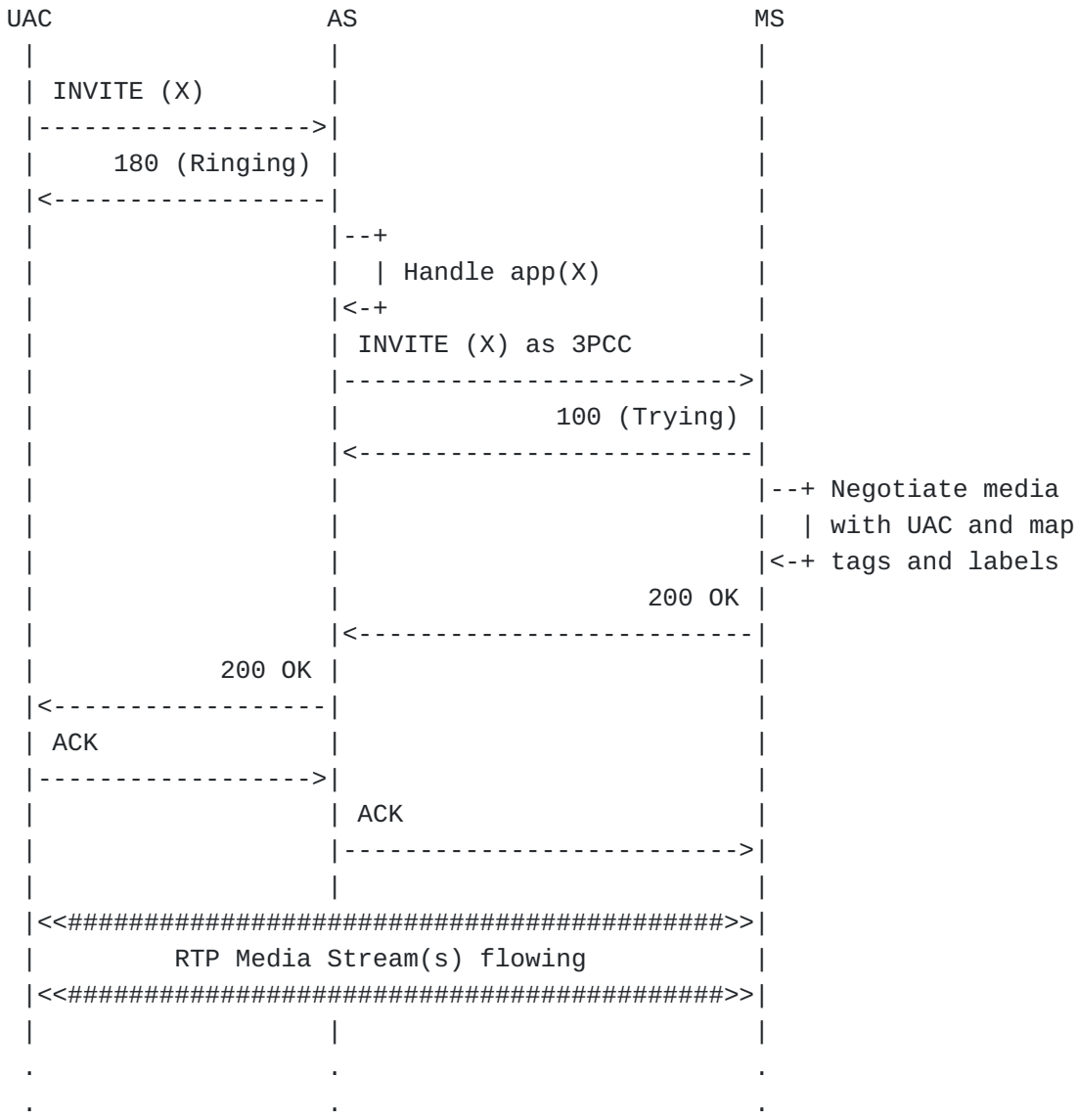


Figure 7: 3PCC Sequence Diagram

Note well: this is only an example of a possible approach involving a 3PCC negotiation among the UAC, the AS and the MS, and as such is not at all to be considered as the mandatory way or as best common practice either in the presented scenario. [RFC3725] (Rosenberg, J., Peterson, J., Schulzrinne, H., and G. Camarillo, “Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP),” April 2004.) provides several different solutions and many details about how 3PCC can be realized, with pros and cons.
The UAC first places a call to a SIP URI the AS is responsible of. The specific URI is not relevant to the examples, since the application logic behind the mapping between a URI and the service it provides is a matter that is important only to the AS: so, a generic 'sip:example@cicciopernacchio.com' is used in all the examples, whereas the service this URI is associated with in the AS logic is mapped scenario by scenario to the case under exam. The UAC INVITE is treated as envisaged in [I-D.ietf-mediactrl-architecture] (Melanchuk, T., “An Architectural Framework for Media Server Control,” November 2008.): the INVITE is forwarded by the AS to the MS in a 3PCC fashion, without the SDP provided by the UAC being touched, thus to have the session fully negotiated by the MS for what concerns its description. The MS matches the UAC's offer with its own capabilities and provides its answer in a 200 OK. This answer is then forwarded, again without the SDP contents being touched, by the AS to the UAC it is intended for. This way, while the SIP signaling from the UAC is terminated to the AS, all the media would start directly flowing between the UAC and the MS.
As a consequence of this negotiation, one or more media connections are created between the MS and the UAC. They are then addressed, when needed, by the AS and the MS by means of the tags concatenation as specified in [I-D.ietf-mediactrl-sip-control-framework] (Boulton, C., Melanchuk, T., and S. McGlashan, “Media Control Channel Framework,” October 2009.). How the identifiers are created and addressed is explained by making use of the sample signaling provided in Figure 8 (3PCC SIP Signaling).

```
1. UAC -> AS (SIP INVITE)
-------------------------
   INVITE sip:mediactrlDemo@cicciopernacchio.com SIP/2.0
   Via: SIP/2.0/UDP 4.3.2.1:5063;rport;branch=z9hG4bK1396873708
   From: <sip:lminiero@users.cicciopernacchio.com>;tag=1153573888
   To: <sip:mediactrlDemo@cicciopernacchio.com>
   Call-ID: 1355333098
   CSeq: 20 INVITE
   Contact: <sip:lminiero@4.3.2.1:5063>
   Content-Type: application/sdp
   Max-Forwards: 70
   User-Agent: Linphone/2.1.1 (eXosip2/3.0.3)
   Subject: Phone call
   Expires: 120
   Content-Length: 330

   v=0
   o=lminiero 123456 654321 IN IP4 4.3.2.1
   s=A conversation
   c=IN IP4 4.3.2.1
   t=0 0
   m=audio 7078 RTP/AVP 0 3 8 101
   a=rtpmap:0 PCMU/8000/1
   a=rtpmap:3 GSM/8000/1
   a=rtpmap:8 PCMA/8000/1
   a=rtpmap:101 telephone-event/8000
   a=fmtp:101 0-11
   m=video 9078 RTP/AVP 98
   a=rtpmap:98 H263-1998/90000
   a=fmtp:98 CIF=1;QCIF=1


2. UAC <- AS (SIP 180 Ringing)
------------------------------
   SIP/2.0 180 Ringing
   Via: SIP/2.0/UDP 4.3.2.1:5063;rport=5063; \
                                                branch=z9hG4bK1396873708
   Contact: <sip:mediactrlDemo@cicciopernacchio.com>
   To: <sip:mediactrlDemo@cicciopernacchio.com>;tag=bcd47c32
   From: <sip:lminiero@users.cicciopernacchio.com>;tag=1153573888
   Call-ID: 1355333098
   CSeq: 20 INVITE
   Content-Length: 0


3. AS -> MS (SIP INVITE)
------------------------
```

```
   INVITE sip:MediaServer@pippozzoserver.org:5060;transport=UDP SIP/2.0
   Via: SIP/2.0/UDP 1.2.3.4:5060; \
                branch=z9hG4bK-d8754z-8723e421ebc45f6b-1---d8754z-;rport
   Max-Forwards: 70
   Contact: <sip:ApplicationServer@1.2.3.4:5060>
   To: <sip:MediaServer@pippozzoserver.org:5060>
   From: <sip:ApplicationServer@cicciopernacchio.com:5060>;tag=10514b7f
   Call-ID: NzI0ZjQ0ZTBlMTEzMGU1ZjVhMjk5NTliMmJmZjE0NDQ.
   CSeq: 1 INVITE
   Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, INVITE, REGISTER
   Content-Type: application/sdp
   Content-Length: 330

   v=0
   o=lminiero 123456 654321 IN IP4 4.3.2.1
   s=A conversation
   c=IN IP4 4.3.2.1
   t=0 0
   m=audio 7078 RTP/AVP 0 3 8 101
   a=rtpmap:0 PCMU/8000/1
   a=rtpmap:3 GSM/8000/1
   a=rtpmap:8 PCMA/8000/1
   a=rtpmap:101 telephone-event/8000
   a=fmtp:101 0-11
   m=video 9078 RTP/AVP 98
   a=rtpmap:98 H263-1998/90000
   a=fmtp:98 CIF=1;QCIF=1


4. AS <- MS (SIP 100 Trying)
----------------------------
   SIP/2.0 100 Trying
   Via: SIP/2.0/UDP 1.2.3.4:5060; \
           branch=z9hG4bK-d8754z-8723e421ebc45f6b-1---d8754z-;rport=5060
   To: <sip:MediaServer@pippozzoserver.org:5060>;tag=6a900179
   From: <sip:ApplicationServer@cicciopernacchio.com:5060>;tag=10514b7f
   Call-ID: NzI0ZjQ0ZTBlMTEzMGU1ZjVhMjk5NTliMmJmZjE0NDQ.
   CSeq: 1 INVITE
   Content-Length: 0


5. AS <- MS (SIP 200 OK)
------------------------
   SIP/2.0 200 OK
   Via: SIP/2.0/UDP 1.2.3.4:5060; \
           branch=z9hG4bK-d8754z-8723e421ebc45f6b-1---d8754z-;rport=5060
   Contact: <sip:MediaServer@pippozzoserver.org:5060>
   To: <sip:MediaServer@pippozzoserver.org:5060>;tag=6a900179
   From: <sip:ApplicationServer@cicciopernacchio.com:5060>;tag=10514b7f
```

```
   Call-ID: NzI0ZjQ0ZTBlMTEzMGU1ZjVhMjk5NTliMmJmZjE0NDQ.
   CSeq: 1 INVITE
   Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, INVITE, REGISTER
   Content-Type: application/sdp
   Content-Length: 374

   v=0
   o=lminiero 123456 654322 IN IP4 pippozzoserver.org
   s=MediaCtrl
   c=IN IP4 pippozzoserver.org
   t=0 0
   m=audio 63442 RTP/AVP 0 3 8 101
   a=rtpmap:0 PCMU/8000
   a=rtpmap:3 GSM/8000
   a=rtpmap:8 PCMA/8000
   a=rtpmap:101 telephone-event/8000
   a=fmtp:101 0-15
   a=ptime:20
   a=label:7eda834
   m=video 33468 RTP/AVP 98
   a=rtpmap:98 H263-1998/90000
   a=fmtp:98 CIF=2
   a=label:0132ca2
```

6. UAC <- AS (SIP 200 OK)

```
   SIP/2.0 200 OK
   Via: SIP/2.0/UDP 4.3.2.1:5063;rport=5063; \
                                                 branch=z9hG4bK1396873708
   Contact: <sip:mediactrlDemo@cicciopernacchio.com>
   To: <sip:mediactrlDemo@cicciopernacchio.com>;tag=bcd47c32
   From: <sip:lminiero@users.cicciopernacchio.com>;tag=1153573888
   Call-ID: 1355333098
   CSeq: 20 INVITE
   Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, INVITE, REGISTER
   Content-Type: application/sdp
   Content-Length: 374

   v=0
   o=lminiero 123456 654322 IN IP4 pippozzoserver.org
   s=MediaCtrl
   c=IN IP4 pippozzoserver.org
   t=0 0
   m=audio 63442 RTP/AVP 0 3 8 101
   a=rtpmap:0 PCMU/8000
   a=rtpmap:3 GSM/8000
   a=rtpmap:8 PCMA/8000
   a=rtpmap:101 telephone-event/8000
```

```
   a=fmtp:101 0-15
   a=ptime:20
   a=label:7eda834
   m=video 33468 RTP/AVP 98
   a=rtpmap:98 H263-1998/90000
   a=fmtp:98 CIF=2
   a=label:0132ca2


7. UAC -> AS (SIP ACK)
----------------------
   ACK sip:mediactrlDemo@cicciopernacchio.com SIP/2.0
   Via: SIP/2.0/UDP 4.3.2.1:5063;rport;branch=z9hG4bK1113338059
   From: <sip:lminiero@users.cicciopernacchio.com>;tag=1153573888
   To: <sip:mediactrlDemo@cicciopernacchio.com>;tag=bcd47c32
   Call-ID: 1355333098
   CSeq: 20 ACK
   Contact: <sip:lminiero@4.3.2.1:5063>
   Max-Forwards: 70
   User-Agent: Linphone/2.1.1 (eXosip2/3.0.3)
   Content-Length: 0


8. AS -> MS (SIP ACK)
---------------------
   ACK sip:MediaServer@pippozzoserver.org:5060;transport=UDP SIP/2.0
   Via: SIP/2.0/UDP 1.2.3.4:5060; \
                branch=z9hG4bK-d8754z-5246003419ccd662-1---d8754z-;rport
   Max-Forwards: 70
   Contact: <sip:ApplicationServer@1.2.3.4:5060>
   To: <sip:MediaServer@pippozzoserver.org:5060;tag=6a900179
   From: <sip:ApplicationServer@cicciopernacchio.com:5060>;tag=10514b7f
   Call-ID: NzI0ZjQ0ZTBlMTEzMGU1ZjVhMjk5NTliMmJmZjE0NDQ.
   CSeq: 1 ACK
   Content-Length: 0
```

Figure 8: 3PCC SIP Signaling

As a result of the 3PCC negotiation depicted in Figure 8 (3PCC SIP Signaling), the following relevant information is retrieved:

1. The 'From' and 'To' tags (10514b7f and 6a900179 respectively) of the AS<->MS session:

```
From: <sip:ApplicationServer@cicciopernacchio.com:5060>;tag=10514b7f
                                                          ^^^^^^^^
To: <sip:MediaServer@pippozzoserver.org:5060>;tag=6a900179
                                                  ^^^^^^^^
```

2. the labels associated with the negotiated media connections, in this case an audio media stream (7eda834) and a video media stream (0132ca2).

```
m=audio 63442 RTP/AVP 0 3 8 101
[..]
a=label:7eda834
        ^^^^^^^
m=video 33468 RTP/AVP 98
[..]
a=label:0132ca2
        ^^^^^^^
```

These three identifiers allow the AS and MS to univocally and unambiguously address to each other the connections associated with the related UAC, specifically:

1. 10514b7f~6a900179, the concatenation of the 'From' and 'To' tags, addresses all the media connections between the MS and the UAC;

2. 10514b7f~6a900179~7eda834, the concatenation of the previous value with the label attribute, addresses only one of the media connections of the UAC session (in this case, the audio media stream).

The mapping the AS makes between the UACs<->AS and the AS<->MS SIP dialogs is instead out of scope for this document: we just assume that the AS knows how to address the right connection according to the related session it has with a UAC (e.g. to play an announcement to a specific UAC), which is obviously very important considering the AS is responsible for all the business logic of the multimedia application it provides.

TOC

6.1. Echo Test

The echo test is the simpliest example scenario that can be achieved by means of a Media Server. It basically consists of a UAC directly or indirectly "talking" to itself. A media perspective of such a scenario is depicted in Figure 9 (Echo Test: Media Perspective).

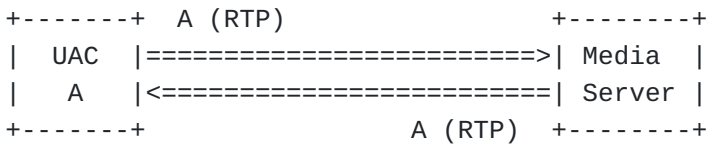


Figure 9: Echo Test: Media Perspective

From the framework point of view, when the UAC's leg is not attached to anything yet, what appears is described in Figure 10 (Echo Test: UAC Media Leg not attached): since there's no connection involving the UAC yet, the frames it might be sending are discarded, and nothing is sent to it (except for silence, if it is requested to be transmitted).

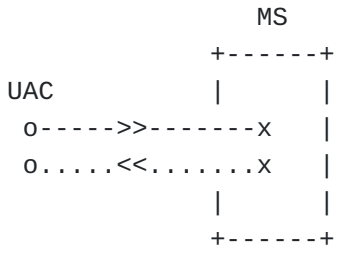


Figure 10: Echo Test: UAC Media Leg not attached

Starting from these considerations, two different approaches to the Echo Test scenario are explored in this document in the following paragraphes:

1. a Direct Echo Test approach, where the UAC directly talks to itself;

2. a Recording-based Echo Test approach, where the UAC indirectly talks to itself.

6.1.1. Direct Echo Test

TOC

In the Direct Echo Test approach, the UAC is directly connected to itself. This means that, as depicted in Figure 11 (Echo Test: Direct Echo (self connection)), each frame the MS receives from the UAC is sent back to it in real-time.

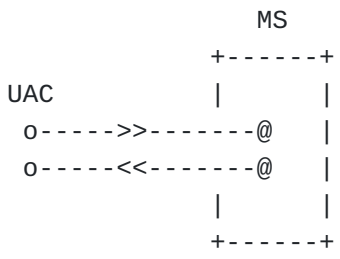


Figure 11: Echo Test: Direct Echo (self connection)

In the framework this can be achieved by means of the conference control package, which is in charge of the task of joining connections and conferences.
A sequence diagram of a potential transaction is depicted in Figure 12 (Self Connection: Framework Transaction):

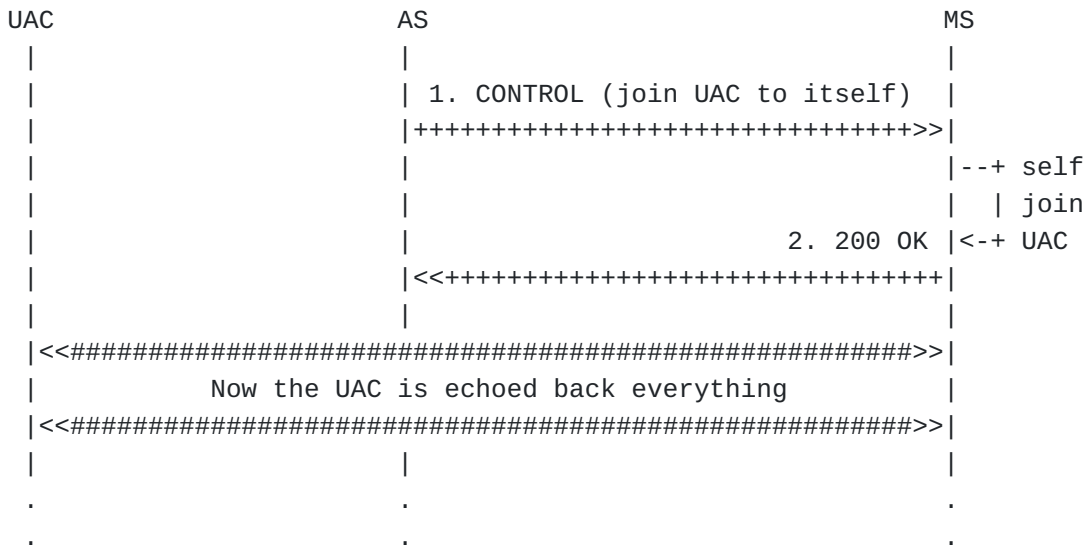


Figure 12: Self Connection: Framework Transaction

All the transaction steps have been numbered to ease up the understanding and the following of the subsequent explaination lines:

- The AS requests the joining of the connection to itself by sending a CONTROL request (1), specifically meant for the conferencing control package (msc-mixer/1.0), to the MS: since the connection must be attached to itself, the id1 and id2 attributes are set to the same value, i.e. the connectionid;

- The MS, having checked the validity of the request, enforces the joining of the connection to itself; this means that all the frames sent by the UAC are sent back to it; to report the result of the operation, the MS sends a 200 OK (2) in reply to the MS, thus ending the transaction; the transaction ended successfully, as testified by the body of the message (the 200 status code in the <response> tag).

The complete transaction, that is the full bodies of the exchanged messages, is provided in the following lines:

```
1. AS -> MS (CFW CONTROL)
-------------------------
   CFW 4fed9bf147e2 CONTROL
   Control-Package: msc-mixer/1.0
   Content-Type: application/msc-mixer+xml
   Content-Length: 90

   <mscmixer version="1.0">
      <join id1="10514b7f~6a900179" \
            id2="10514b7f~6a900179"/>
   </mscmixer>


2. AS <- MS (CFW 200 OK)
------------------------
   CFW 4fed9bf147e2 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" \
                reason="Join successful"/>
   </mscmixer>
```

6.1.2. Echo Test based on Recording

TOC

In the Recording-based Echo Test approach, instead, the UAC is NOT directly connected to itself, but indirectly. This means that, as depicted in Figure 13 (Echo Test: Recording involved), each frame the MS receives from the UAC is first recorded: then, when the recording process is ended, the whole recorded frames are played back to the UAC as an announcement.

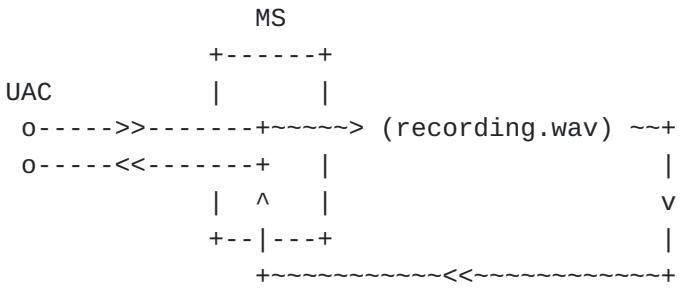


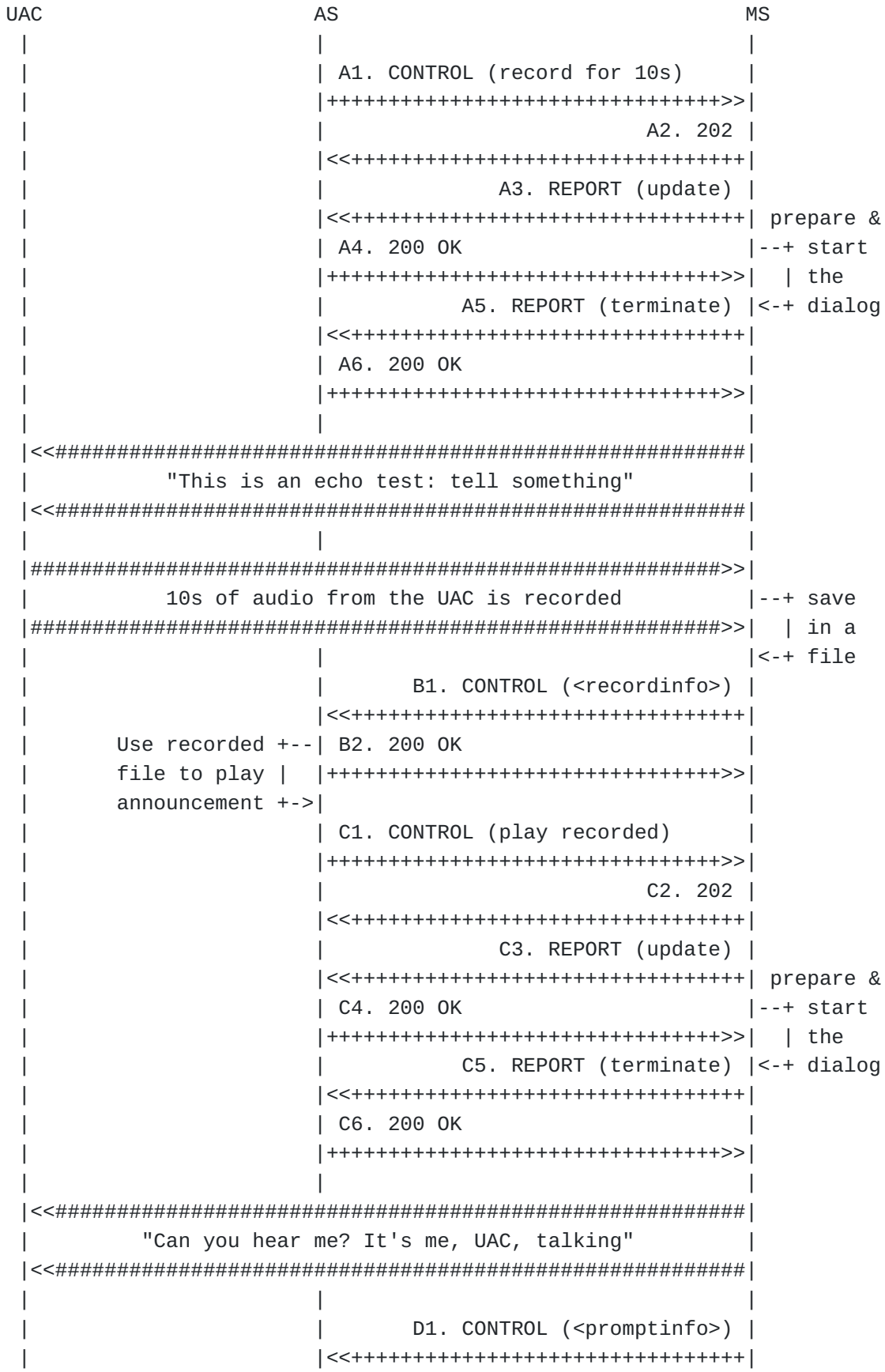
Figure 13: Echo Test: Recording involved

In the framework this can be achieved by means of the IVR control package, which is in charge of the task of recording and playout. However, the whole scenario cannot be accomplished in a single transaction; at least two steps, in fact, need to be followed:

1. first, a recording (preceded by an announcement, if requested) must take place;

2. then, a playout of the previously recorded media must occur.

This means that two separate transactions need to be invoked. A sequence diagram of a potential multiple transaction is depicted in Figure 14 (Recording-based Echo: Two Framework Transactions):


UAC
AS
MS
A1. CONTROL (record for 10s)
A2. 202
A3. REPORT (update)
A4. 200 OK
prepare & start the dialog
A5. REPORT (terminate)
A6. 200 OK
"This is an echo test: tell something"
10s of audio from the UAC is recorded
save in a file
B1. CONTROL (<recordinfo>)
B2. 200 OK
Use recorded file to play announcement
C1. CONTROL (play recorded)
C2. 202
C3. REPORT (update)
C4. 200 OK
prepare & start the dialog
C5. REPORT (terminate)
C6. 200 OK
"Can you hear me? It's me, UAC, talking"
D1. CONTROL (<promptinfo>)

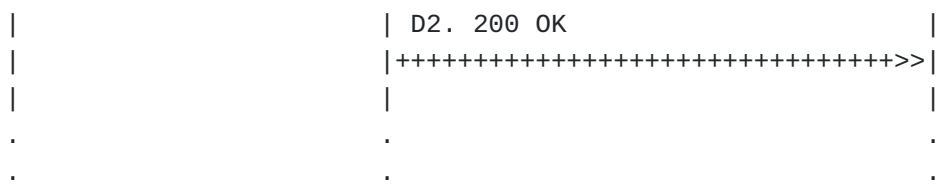


Figure 14: Recording-based Echo: Two Framework Transactions

Notice how the AS-originated CONTROL transactions are terminated as soon as the requested dialogs start: as specified in [I-D.ietf-mediactrl-ivr-control-package] (McGlashan, S., Melanchuk, T., and C. Boulton, "An Interactive Voice Response (IVR) Control Package for the Media Control Channel Framework," February 2010.), the MS makes use of a framework CONTROL message to report the result of the dialog and how it has proceeded. The two transactions (the AS-generated CONTROL request and the MS-generated CONTROL event) are correlated by means of the associated dialog identifier, as it will be clearer from the following lines. As before, all the transaction steps have been numbered to ease up the understanding and the following of the subsequent explaination lines. Besides, the two transactions are distinguished by the preceding letter (A,B=recording, C,D=playout).

*The AS, as a first transaction, invokes a recording on the UAC connection by means of a CONTROL request (A1); the body is for the IVR package (msc-ivr/1.0), and requests the start (dialogstart) of a new recording context (<record>); the recording must be preceded by an announcement (<prompt>), must not last longer than 10s (maxtime), and cannot be interrupted by a DTMF tone (dtmfterm=false); this has only to be done once (repeatCount), which means that if the recording does not succeed the first time, the transaction must fail; a video recording is requested (type), which is to be feeded by both the negotiated media streams; a beep has to be played (beep) right before the recording starts to notify the UAC;

*As seen before, the first responses to the request start flowing: the provisional 202 (A2), the subsequent REPORT update (A3), and its ack (A4) from the AS;

*In the meanwhile, the MS prepares the dialog (e.g. by retrieving the announcement file, for which a HTTP URL is provided, and by checking that the request is well formed) and if all is fine it starts it, notifying the AS about it with a new REPORT (A5) with a terminated status: the connection is then passed to the IVR package, which first plays the announcement on the connection, followed by a beep, and then records all the incoming frames to a

buffer; the MS also provides the AS with an unique dialog identifier (dialogid) which will be used in all subsequent event notifications concerning the dialog it refers to;

*The AS acks the latest REPORT (A6), thus terminating this transaction, waiting for the result to come;

*Once the recording is over, the MS prepares a notification CONTROL (B1); the <event> body is prepared with an explicit reference to the previously provided dialogid identifier, in order to make the AS aware of the fact that the notification is related to that specific dialog; the event body is then completed with the recording related information (<recordinfo>) , in this case the path to the recorded file (here a HTTP URL) which can be used by the AS for whatever it needs to; the payload also information about the prompt (<promptinfo&gT;), which is however not relevant to the scenario;

*The AS concludes this first recording transaction by acking the CONTROL event (B2).

Now that the first transaction has ended, the AS has the 10s recording of the UAC talking. It can let the UAC hear it by having the MS play it to the MS as an announcement:

*The AS, as a second transaction, invokes a playout on the UAC connection by means of a new CONTROL request (C1); the body is once againg for the IVR package (msc-ivr/1.0), but this time it requests the start (dialogstart) of a new announcement context (<prompt>); the file to be played is the one recorded before (prompts), and has only to be played once (iterations);

*Again, the usual provisional 202 (C2), the subsequent REPORT update (C3), and its ack (C4) from the AS take place;

*In the meanwhile, the MS prepares the new dialog and starts it, notifying the AS about it with a new REPORT (C5) with a terminated status: the connection is then passed to the IVR package, which plays the file on it;

*The AS acks the terminating REPORT (C6), now waiting for the announcement to end;

*Once the playout is over, the MS sends a CONTROL event (D1) which contains in its body (<promptinfo>) information about the just concluded announcement; as before, the proper dialogid is used as a reference to the correct dialog;

*The AS concludes this second and last transaction by acking the CONTROL event (D2).

As in the previous paragraph, the whole CFW interaction is provided for a more in depth evaluation of the protocol interaction.

```
A1. AS -> MS (CFW CONTROL, record)
---------------------------------
   CFW 796d83aa1ce4 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 245

   <mscivr version="1.0">
     <dialogstart connectionid="10514b7f~6a900179">
       <dialog>
         <prompt>
           <media \
             src="http://www.cicciopernacchio.com/demo/echorecord.mpg"/>
         </prompt>
         <record beep="true" maxtime="10s" vadinitial="false"/>
       </dialog>
     </dialogstart>
   </mscivr>


A2. AS <- MS (CFW 202)
----------------------
   CFW 796d83aa1ce4 202


A3. AS <- MS (CFW REPORT update)
--------------------------------
   CFW 796d83aa1ce4 REPORT
   Seq: 1
   Status: update
   Timeout: 10


A4. AS -> MS (CFW 200, ACK to 'REPORT update')
----------------------------------------------
   CFW 796d83aa1ce4 200
   Seq: 1


A5. AS <- MS (CFW REPORT terminate)
-----------------------------------
   CFW 796d83aa1ce4 REPORT
   Seq: 2
   Status: terminate
   Timeout: 25
   Content-Type: application/msc-ivr+xml
   Content-Length: 137
```

```
   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <response status="200" reason="Dialog started" \
                dialogid="68d6569"/>
   </mscivr>


A6. AS -> MS (CFW 200, ACK to 'REPORT terminate')
-------------------------------------------------
   CFW 796d83aa1ce4 200
   Seq: 2


B1. AS <- MS (CFW CONTROL event)
--------------------------------
   CFW 0eb1678c0bfc CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 385

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
     <event dialogid="68d6569">
       <dialogexit status="1" reason="Dialog successfully completed">
         <promptinfo duration="5759" termmode="bargein"/>
         <recordinfo recording=\
      "http://www.pippozzoserver.org/recordings/recording-68d6569.mpg" \
                    type="video/mpeg" duration="10006" size="1245184" \
                    termmode="maxtime"/>
       </dialogexit>
     </event>
   </mscivr>


B2. AS -> MS (CFW 200, ACK to 'CONTROL event')
----------------------------------------------
   CFW 0eb1678c0bfc 200


C1. AS -> MS (CFW CONTROL, play)
--------------------------------
   CFW 1632eead7e3b CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 204

   <mscivr version="1.0">
     <dialogstart connectionid="10514b7f~6a900179">
       <dialog>
         <prompt>
           <media \
  src="http://www.pippozzoserver.org/recordings/recording-68d6569.mpg"/>
```

```
        </prompt>
      </dialog>
    </dialogstart>
  </mscivr>
```

C2. AS <- MS (CFW 202)

```
  CFW 1632eead7e3b 202
```

C3. AS <- MS (CFW REPORT update)

```
  CFW 1632eead7e3b REPORT
  Seq: 1
  Status: update
  Timeout: 10
```

C4. AS -> MS (CFW 200, ACK to 'REPORT update')

```
  CFW 1632eead7e3b 200
  Seq: 1
```

C5. AS <- MS (CFW REPORT terminate)

```
  CFW 1632eead7e3b REPORT
  Seq: 2
  Status: terminate
  Timeout: 25
  Content-Type: application/msc-ivr+xml
  Content-Length: 137

  <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
     <response status="200" reason="Dialog started" \
               dialogid="5f5cb45"/>
  </mscivr>
```

C6. AS -> MS (CFW 200, ACK to 'REPORT terminate')

```
  CFW 1632eead7e3b 200
  Seq: 2
```

D1. AS <- MS (CFW CONTROL event)

```
  CFW 502a5fd83db8 CONTROL
  Control-Package: msc-ivr/1.0
```

```
   Content-Type: application/msc-ivr+xml
   Content-Length: 230

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
     <event dialogid="5f5cb45">
       <dialogexit status="1" reason="Dialog successfully completed">
         <promptinfo duration="10366" termmode="completed"/>
       </dialogexit>
     </event>
   </mscivr>


D2. AS -> MS (CFW 200, ACK to 'CONTROL event')
-----------------------------------------------
   CFW 502a5fd83db8 200
```

6.2. Phone Call

TOC

Another scenario that might involve the interaction between an AS and a MS is the classic phone call between two UACs. In fact, even though the most straightforward way to achieve this would be to let the UACs negotiate the session and the media to make use of between themselves, there are cases when the services provided by a MS might come in handy for phone calls as well.

One of these cases is when the two UACs have no common supported codecs: having the two UACs directly negotiate the session would result in a session with no available media. Involving the MS as a transcoder would instead allow the two UACs to communicate anyway. Another interesting case is when the AS (or any other entity the AS is working in behalf of) is interested in manipulating or monitoring the media session between the UACs, e.g. to record the conversation: a similar scenario will be dealt with in Section 6.2.2 (Conference-based Approach).

Before proceeding in looking at how such a scenario might be accomplished by means of the Media Control Channel Framework, it is worth spending a couple of words upon how the SIP signaling involving all the interested parties might look like. In fact in such a scenario a 3PCC approach is absolutely needed. An example is provided in Figure 15 (Phone Call: Example of 3PCC). Again, the presented example is not at all to be considered best common practice when 3PCC is needed in a MediaCtrl-based framework. It is only described in order to let the reader more easily understand what are the requirements on the MS side, and as a consequence which information might be required.

[RFC3725] (Rosenberg, J., Peterson, J., Schulzrinne, H., and G. Camarillo, “Best Current Practices for Third Party Call Control (3pcc)

in the Session Initiation Protocol (SIP)," April 2004.) provides with a much more detailed overview on 3PCC patterns in several use cases. Only an explainatory sequence diagram is provided, without delving into the details of the exchanged SIP messages.

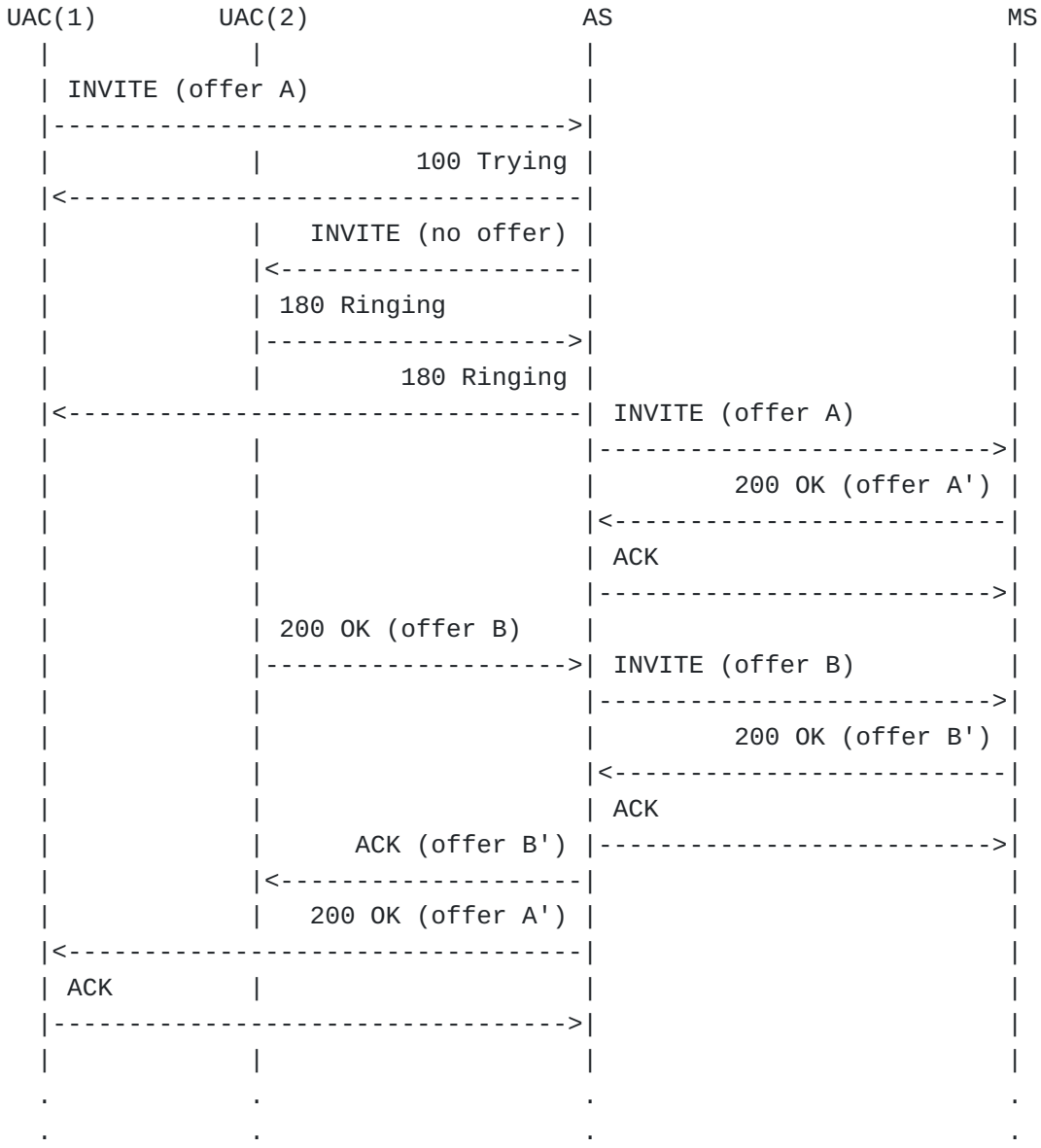


Figure 15: Phone Call: Example of 3PCC

In the example, the UAC1 wants to place a phone call to UAC2. To do so, it sends an INVITE to the AS with its offer A. The AS sends an offerless INVITE to UAC2. When the UAC2 responds with a 180, the same

message is forwarded by the AS to the UAC1 to notify it the callee is ringing. In the meanwhile, the AS also adds a leg to the MS for UAC1, as explained in the previous sections: to do so it of course makes use of the offer A the UAC1 made. Once the UAC2 accepts the call, by providing its own offer B in the 200, the AS adds a leg for it too to the MS. At this point, the negotiation can be completed by providing the two UACs with the SDP answer negotiated by the MS with them (A' and B' respectively).
This is only one way to deal with the signaling, and shall not absolutely be considered as a mandatory approach of course.
Once the negotiation is over, the two UACs are not in communication yet. In fact, it's up to the AS now to actively trigger the MS into attaching their media streams to each other someway, by referring to the connection identifiers associated with the UACs as explained previously. This document presents two different approaches that might be followed, according to what needs to be accomplished. A generic media perspective of the phone call scenario is depicted in Figure 16 (Phone Call: Media Perspective): the MS is basically in the media path between the two UACs.

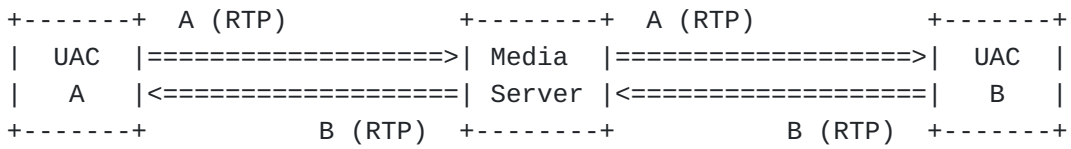


Figure 16: Phone Call: Media Perspective

From the framework point of view, when the UACs' leg are not attached to anything yet, what appears is described in Figure 17 (Phone Call: UAC Media Leg not attached): since there are no connections involving the UACs yet, the frames they might be sending are discarded, and nothing is sent to them (except for silence, if it is requested to be transmitted).

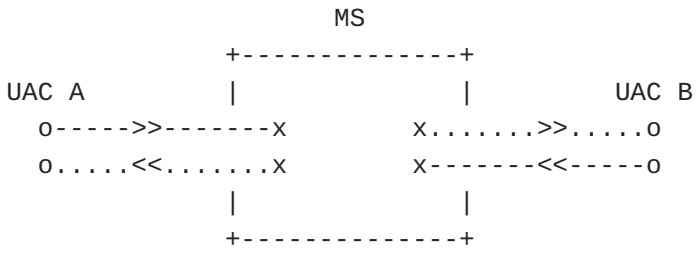


Figure 17: Phone Call: UAC Media Leg not attached

6.2.1. Direct Connection

TOC

The Direct Connection is the easiest and more straightforward approach to get the phone call between the two UACs to work. The idea is basically the same as the one in the Direct Echo approach: a <join> directive is used to directly attach one UAC to the other, by leaving the MS to only deal with the transcoding/adaption of the flowing frames, if needed.
This approach is depicted in Figure 18 (Phone Call: Direct Connection).

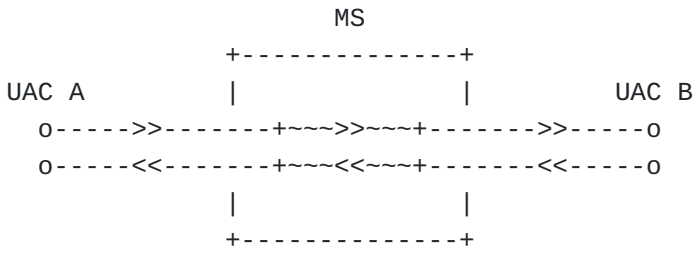


Figure 18: Phone Call: Direct Connection

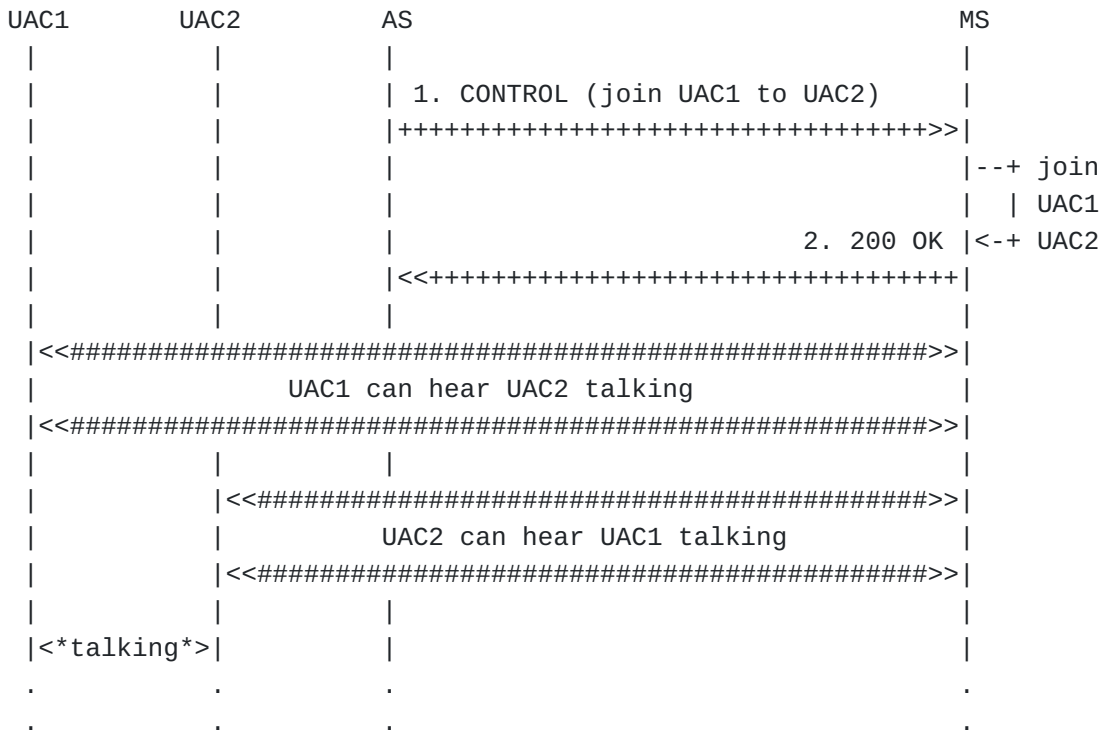


Figure 19: Direct Connection: Framework Transactions

The framework transactions needed to accomplish this scenario are very trivial and easy to understand. They basically are the same as the one presented in the Direct Echo Test scenario, with the only difference being in the provided identifiers. In fact, this time the MS is not supposed to attach the UAC's media connections to themselves, but has to join the media connections of two different UACs, i.e. UAC1 and UAC2. This means that, in this transaction, id1 and i2 will have to address the media connections of UAC1 and UAC2. In case of a successful transaction, the MS takes care of forwarding all media coming from UAC1 to UAC2 and vice versa, transparently taking care of any required transcoding steps, if necessary.

```
1. AS -> MS (CFW CONTROL)
-------------------------
   CFW 0600855d24c8 CONTROL
   Control-Package: msc-mixer/1.0
   Content-Type: application/msc-mixer+xml
   Content-Length: 90

   <mscmixer version="1.0">
      <join id1="10514b7f~6a900179" \
            id2="e1e1427c~1c998d22"/>
   </mscmixer>


2. AS <- MS (CFW 200 OK)
------------------------
   CFW 0600855d24c8 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" \
                reason="Join successful"/>
   </mscmixer>
```

Such a simple approach has its drawbacks. For instance, with such an approach recording a conversation between two users might be tricky to accomplish. In fact, since no mixing would be involved, only the single connections (UAC1<->MS and UAC2<->MS) could be recorded. If the AS wants a conversation recording service to be provided anyway, it needs additional business logic on its side. An example of such a use case is provided in Section 6.2.3 (Recording a conversation).

6.2.2. Conference-based Approach

TOC

The approach described in Section 6.2.1 (Direct Connection) surely works for a basic phone call, but as already explained might have some drawbacks whenever more advanced features were needed. For intance, you can't record the whole conversation, only the single connections, since no mixing is involved. Besides, even the single task of playing an announcement over the conversation could be complex, especially if the MS does not support implicit mixing over media connections. For this reason, in more advanced cases a different approach might be taken, like the conference-based approach described in this section.

The idea is to make use of a mixing entity in the MS that acts as a bridge between the two UACs: the presence of this entity allows for more customization on what needs to be done on the conversation, like the recording of the conversation that has been provided as example. The approach is depicted in Figure 20 (Phone Call: Conference-based Approach). The mixing functionality in the MS will be described in more detail in the following section (which deals with many conference-related scenarios), so only some hints will be provided here for a basic comprehension of the approach.

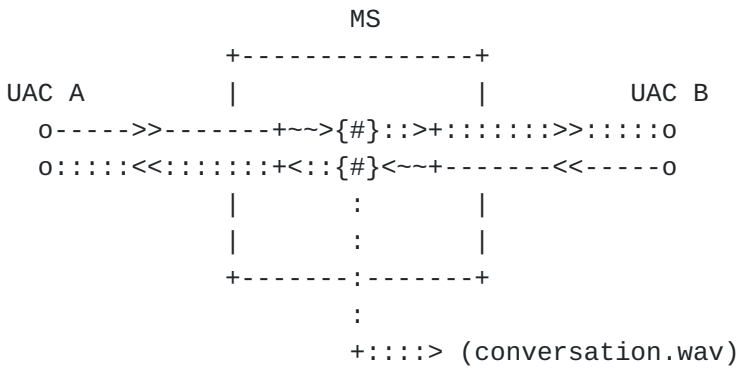


Figure 20: Phone Call: Conference-based Approach

To identify a single sample scenario, let's consider a phone call the AS wants to be recorded.
Figure 21 (Conference-based Approach: Framework Transactions) shows how this could be accomplished in the Media Control Channel Framework: the example, as usual, hides the previous interaction between the UACs and the AS, and instead focuses on the control channel operations and what follows.

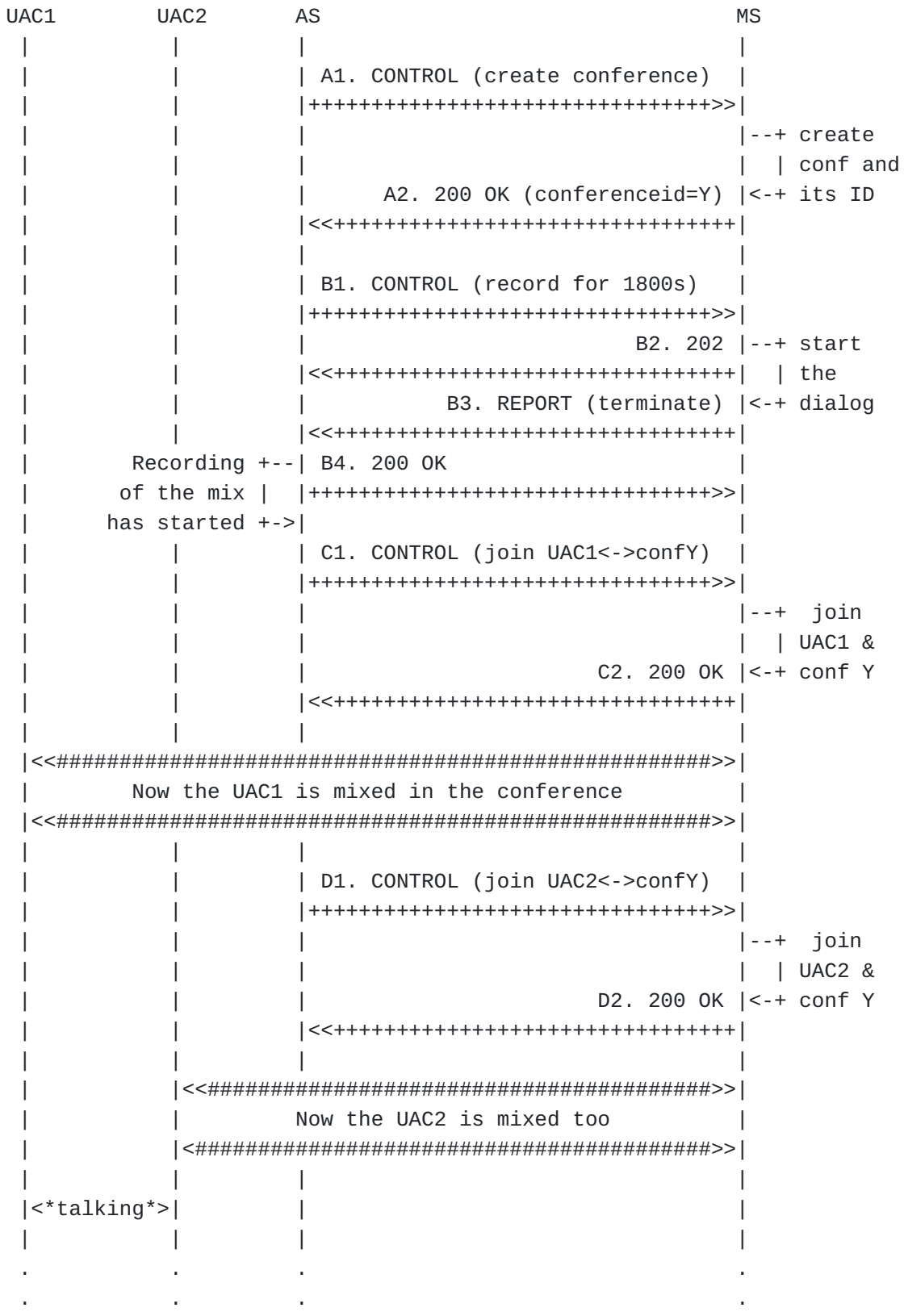

UAC1
UAC2
AS
MS
A1. CONTROL (create conference)
--+ create
conf and
A2. 200 OK (conferenceid=Y)
<-+ its ID
B1. CONTROL (record for 1800s)
B2. 202
--+ start
the
B3. REPORT (terminate)
<-+ dialog
Recording
of the mix
has started
B4. 200 OK
C1. CONTROL (join UAC1<->confY)
--+ join
UAC1 &
C2. 200 OK
<-+ conf Y
Now the UAC1 is mixed in the conference
D1. CONTROL (join UAC2<->confY)
--+ join
UAC2 &
D2. 200 OK
<-+ conf Y
Now the UAC2 is mixed too
<*talking*>

Figure 21: Conference-based Approach: Framework Transactions

The AS makes use of two different packages to accomplish this scenario: the mixer package (to create the mixing entity and join the UACs) and the IVR package (to record what happens in the conference). The framework transaction steps can be described as follows:

*First of all, the AS creates a new hidden conference by means of a 'createconference' request (A1); this conference is properly configured according to the use it is assigned to; in fact, since only two participants will be joined to it, both 'reserved-talkers' and 'reserved-listeners' are set to 2; besides, the video layout as well is set accordingly (single-view/dual-view);

*the MS notifies the successful creation of the new conference in a 200 framework message (A2); the identifier assigned to the conference, which will be used in subsequent request addressed to it, is 6013f1e;

*the AS requests a new recording upon the newly created conference; to do so, it places a proper request to the IVR package (B1); the AS is interested in a video recording (type=video/mpeg), which must not last longer than 3 hours (maxtime=1800s), after which the recording must end; besides, no beep must be played on the conference (beep=false), and the recording must start immediately whether or not any audio activity has been reported (vadinitial=false);

*the transaction is extended by the MS (B3), and when the dialog has been successfully started, a REPORT terminate is issued to the AS (B4); the message contains the dialogid associated with the dialog (00b29fb), which the AS must refer to for later notifications;

*at this point, the AS attaches both the UACs to the conference with two separate 'join' directives (C1/D1); when the MS confirms the success of both the operations (C2/D2), the two UACs are actually in contact with each other (even though indirectly, since a hidden conference they're unaware of is on their path) and their media contribution is recorded.

A1. AS -> MS (CFW CONTROL, createconference)

```
   CFW 238e1f2946e8 CONTROL
   Control-Package: msc-mixer
   Content-Type: application/msc-mixer+xml
   Content-Length: 357

   <mscmixer version="1.0">
      <createconference reserved-talkers="2" reserved-listeners="2">
         <audio-mixing mix-type="nbest"/>
         <video-switch type="controller"/>
         <video-layouts>
           <video-layout min-participants='1'>single-view</video-layout>
           <video-layout min-participants='2'>dual-view</video-layout>
         </video-layouts>
      </createconference>
   </mscmixer>
```

A2. AS <- MS (CFW 200 OK)

```
   CFW 238e1f2946e8 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 151

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Conference created" \
                conferenceid="6013f1e"/>
   </mscmixer>
```

B1. AS -> MS (CFW CONTROL, record)

```
   CFW 515f007c5bd0 CONTROL
   Control-Package: msc-ivr
   Content-Type: application/msc-ivr+xml
   Content-Length: 188

   <mscivr version="1.0">
      <dialogstart conferenceid="6013f1e">
         <dialog>
            <record beep="false" vadinitial="false" maxtime="1800s" \
                    type="video/mpeg"/>
         </dialog>
      </dialogstart>
   </mscivr>
```

B2. AS <- MS (CFW 202)

```
   CFW 515f007c5bd0 202
```

B3. AS <- MS (CFW REPORT terminate)

```
   CFW 515f007c5bd0 REPORT
   Seq: 1
   Status: terminate
   Timeout: 25
   Content-Type: application/msc-ivr+xml
   Content-Length: 137

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
     <response status="200" reason="Dialog started" dialogid="00b29fb"/>
   </mscivr>
```

B4. AS -> MS (CFW 200, ACK to 'REPORT terminate')

```
   CFW 515f007c5bd0 200
   Seq: 1
```

C1. AS -> MS (CFW CONTROL, join)

```
   CFW 0216231b1f16 CONTROL
   Control-Package: msc-mixer
   Content-Type: application/msc-mixer+xml
   Content-Length: 83

   <mscmixer version="1.0">
      <join id1="10514b7f~6a900179" id2="6013f1e"/>
   </mscmixer>
```

C2. AS <- MS (CFW 200 OK)

```
   CFW 0216231b1f16 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
   </mscmixer>
```

```
D1. AS -> MS (CFW CONTROL, join)
--------------------------------
   CFW 140e0f763352 CONTROL
   Control-Package: msc-mixer
   Content-Type: application/msc-mixer+xml
   Content-Length: 84

   <mscmixer version="1.0">
      <join id1="219782951~0b9d3347" id2="6013f1e"/>
   </mscmixer>


D2. AS <- MS (CFW 200 OK)
-------------------------
   CFW 140e0f763352 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
   </mscmixer>
```

The recording of the conversation can subsequently be accessed by the AS by waiting for an event notification from the MS: this event, which will be associated with the previously started recording dialog, will contain the URI to the recorded file. Such an event may be triggered either by a natural completion of the dialog (e.g. the dialog has reached its programmed 3 hours) or by any interruption of the dialog itself (e.g. the AS actively requested the recording to be interrupted since the call between the UACs ended).

6.2.3. Recording a conversation

TOC

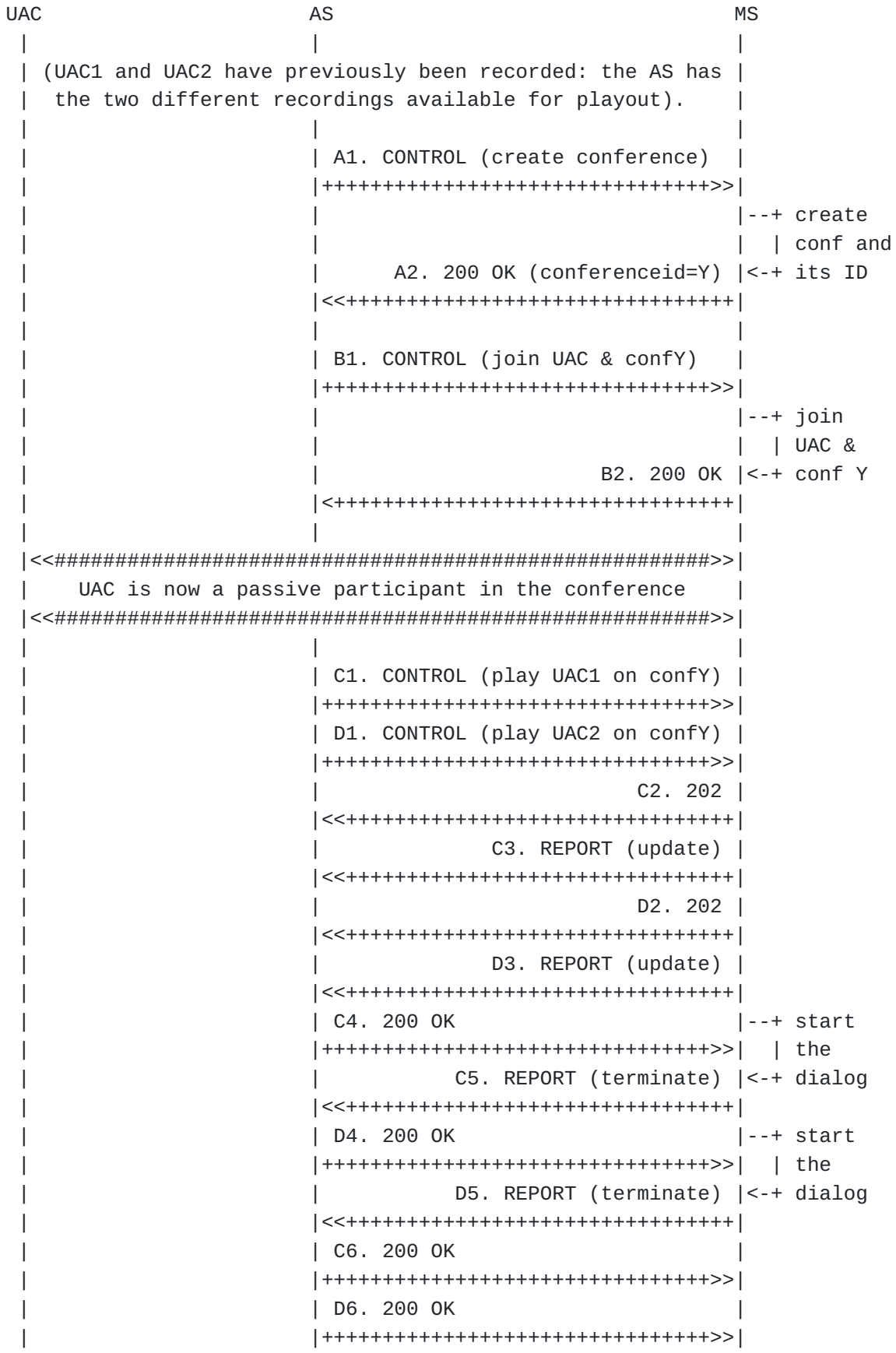
The previous section described how to take advantage of the conferencing functionality of the mixer package in order to allow the recording of phone calls in a simple way. However, making use of a dedicated mixer just for a phone call might be considered overkill. This section shows how recording a conversation and playing it out subsequently can be accomplished without a mixing entity involved in the call, that is by using the direct connection approach as described in Section 6.2.1 (Direct Connection).
As already explained previously, in case the AS wants to record a phone call between two UACs, the use of just the <join> directive without a mixer forces the AS to just rely on separate recording commands. That is, the AS can only instruct the MS to separately record the media

flowing on each media leg: a recording for all the media coming from UAC1, and a different recording for all the media coming from UAC2. In case someone wants to access the whole conversation subsequently, the AS may take at least two different approaches:

1. it may mix the two recordings itself (e.g. by delegating it to an offline mixing entity) in order to obtain a single file containing the combination of thw two recordings; this way, a simple playout as described in section Section 6.1.2 (Echo Test based on Recording) would suffice;

2. alternatively, it may take advantage of the mixing functionality provided by the MS itself; a way to do so is to create a hidden conference on the MS, attach the UAC as a passive participant to it, and play the separate recordings on the conference as announcements; this way, the UAC accessing the recording would experience both the recordings at the same time.

It is of course option 2 that is considered in this section. The framework transaction as described in Figure 22 (Phone Call: Playout of a Recorded Conversation) assumes that a recording has already been requested for both UAC1 and UAC2, that the phone call has ended and that the AS has successfully received the URIs to both the recordings from the MS. Such steps are not described again since they would be quite similar to the ones described in Section 6.1.2 (Echo Test based on Recording). As anticipated, the idea is to make use of a properly constructed hidden conference to mix the two separate recordings on the fly and present them to the UAC. It is of course up to the AS to subsequently unjoin the user from the conference and destroy the conference itself once the playout of the recordings ends for any reason.


UAC
AS
MS
(UAC1 and UAC2 have previously been recorded: the AS has
the two different recordings available for playout).
A1. CONTROL (create conference)
create
conf and
A2. 200 OK (conferenceid=Y)
its ID
B1. CONTROL (join UAC & confY)
join
UAC &
B2. 200 OK
conf Y
UAC is now a passive participant in the conference
C1. CONTROL (play UAC1 on confY)
D1. CONTROL (play UAC2 on confY)
C2. 202
C3. REPORT (update)
D2. 202
D3. REPORT (update)
C4. 200 OK
start
the
C5. REPORT (terminate)
dialog
D4. 200 OK
start
the
D5. REPORT (terminate)
dialog
C6. 200 OK
D6. 200 OK

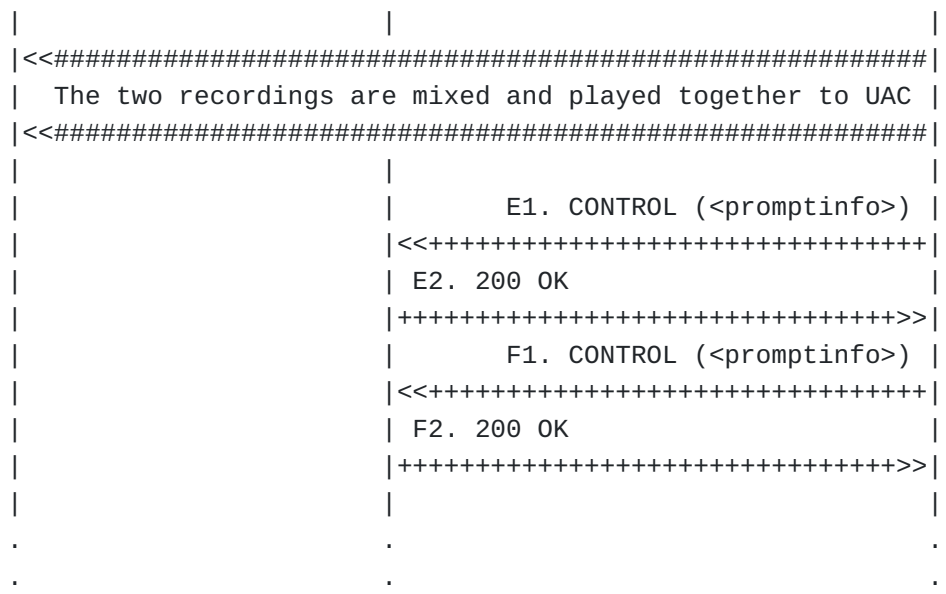


Figure 22: Phone Call: Playout of a Recorded Conversation

The diagram above assumes a recording of both the channels has already taken place. It may have been requested by the AS either shortly before joining UAC1 and UAC2, or shortly after that transaction. Whenever that happened, a recording is assumed to have taken place, and so the AS is supposed to have both the recordings available for playback. Once a new user, UAC, wants to access the recorded conversation, the AS takes care of the presented transactions. The framework transaction steps are only apparently more complicated than the ones presented so far. The only difference, in fact, is that transactions C and D are concurrent, since the recordings must be played together.

*First of all, the AS creates a new conference to act as a mixing entity (A1); the settings for the conference are chosen according to the use case, e.g. the video layout which is fixed to 'dual-view' and the switching type to 'controller'; when the conference has been successfully created (A2) the AS takes note of the conference identifier;

*At this point, the UAC is attached to the conference as a passive user (B1); there would be no point in letting the user contribute to the conference mix, since he will only need to watch a recording; in order to specify his passive status, both the audio and video streams for the user are set to 'recvonly'; in case the transaction succeeds, the MS notifies it to the MS (B2);

*Once the conference has been created and UAC has been attached to it, the AS can request the playout of the recordings; in order to

do so, it requests two concurrent <prompt&gT; directives (C1 and D1), addressing respectively the recording of UAC1 and UAC2; both the prompts must be played on the previously created conference and not to UAC directly, as can be evinced from the 'conferenceid' attribute of the &l;dialog> element;

*The transactions live their life exactly as explained for previous <prompt> examples; the originating transactions are first prepared and started (C2-6, D2-6), and then, as soon as any of the playout ends, a realted CONTROL message to notify this is triggered by the MS (E1, F1); the notification may contain a <promptinfo> element with information about how the playout proceeded (e.g. whether the playout completed normally, or interrupted by a DTMF tone, etc.).

A1. AS -> MS (CFW CONTROL, createconference)

```
   CFW 506e039f65bd CONTROL
   Control-Package: msc-mixer/1.0
   Content-Type: application/msc-mixer+xml
   Content-Length: 271

   <mscmixer version="1.0">
      <createconference reserved-talkers="0" reserved-listeners="2">
         <audio-mixing mix-type="nbest"/>
         <video-switch type="controller"/>
         <video-layouts>
            <video-layout min-participants='1'>dual-view</video-layout>
         </video-layouts>
      </createconference>
   </mscmixer>
```

A2. AS <- MS (CFW 200 OK)

```
   CFW 506e039f65bd 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 151

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Conference created" \
                conferenceid="2625069"/>
   </mscmixer>
```

B1. AS -> MS (CFW CONTROL, join)

```
   CFW 09202baf0c81 CONTROL
   Control-Package: msc-mixer/1.0
   Content-Type: application/msc-mixer+xml
   Content-Length: 174

   <mscmixer version="1.0">
      <join id1="aafaf62d~0eac5236" id2="2625069">
         <stream media="audio" direction="recvonly"/>
         <stream media="video" direction="recvonly"/>
      </join>
   </mscmixer>
```

B2. AS <- MS (CFW 200 OK)

```
   CFW 09202baf0c81 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
   </mscmixer>
```

C1. AS -> MS (CFW CONTROL, play recording from UAC1)

```
   CFW 3c2a08be4562 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 192

   <mscivr version="1.0">
      <dialogstart conferenceid="2625069">
         <dialog>
            <prompt>
               <media \
  src="http://www.pippozzoserver.org/recordings/recording-4ca9fc2.mpg"/>
            </prompt>
         </dialog>
      </dialogstart>
   </mscivr>
```

D1. AS -> MS (CFW CONTROL, play recording from UAC2)

```
   CFW 1c268d810baa CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 192

   <mscivr version="1.0">
      <dialogstart conferenceid="2625069">
         <dialog>
            <prompt>
               <media \
  src="http://www.pippozzoserver.org/recordings/recording-39dfef4.mpg"/>
            </prompt>
         </dialog>
      </dialogstart>
   </mscivr>
```

C2. AS <- MS (CFW 202)

```
----------------------
```

```
   CFW 3c2a08be4562 202


C3. AS <- MS (CFW REPORT update)
--------------------------------
   CFW 3c2a08be4562 REPORT
   Seq: 1
   Status: update
   Timeout: 10


D2. AS <- MS (CFW 202)
----------------------
   CFW 1c268d810baa 202


D3. AS <- MS (CFW REPORT update)
--------------------------------
   CFW 1c268d810baa REPORT
   Seq: 1
   Status: update
   Timeout: 10


C4. AS -> MS (CFW 200, ACK to 'REPORT update')
----------------------------------------------
   CFW 3c2a08be4562 200
   Seq: 1


D4. AS -> MS (CFW 200, ACK to 'REPORT update')
----------------------------------------------
   CFW 1c268d810baa 200
   Seq: 1


C5. AS <- MS (CFW REPORT terminate)
-----------------------------------
   CFW 1c268d810baa REPORT
   Seq: 2
   Status: terminate
   Timeout: 25
   Content-Type: application/msc-ivr+xml
   Content-Length: 137

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <response status="200" reason="Dialog started" \
                dialogid="7a457cc"/>
   </mscivr>
```

```
D5. AS <- MS (CFW REPORT terminate)
-----------------------------------
   CFW 3c2a08be4562 REPORT
   Seq: 2
   Status: terminate
   Timeout: 25
   Content-Type: application/msc-ivr+xml
   Content-Length: 137

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <response status="200" reason="Dialog started" \
                dialogid="1a0c7cf"/>
   </mscivr>


C6. AS -> MS (CFW 200, ACK to 'REPORT terminate')
-------------------------------------------------
   CFW 1c268d810baa 200
   Seq: 2


D6. AS -> MS (CFW 200, ACK to 'REPORT terminate')
-------------------------------------------------
   CFW 3c2a08be4562 200
   Seq: 2


E1. AS <- MS (CFW CONTROL event, playout of recorded UAC1 ended)
----------------------------------------------------------------
   CFW 77aec0735922 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 230

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <event dialogid="7a457cc">
         <dialogexit status="1" reason="Dialog successfully completed">
            <promptinfo duration="10339" termmode="completed"/>
         </dialogexit>
      </event>
   </mscivr>


E2. AS -> MS (CFW 200, ACK to 'CONTROL event')
----------------------------------------------
   CFW 77aec0735922 200


F1. AS <- MS (CFW CONTROL event, playout of recorded UAC2 ended)
```

```
-----------------------------------------------------------------
   CFW 62726ace1660 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 230

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <event dialogid="1a0c7cf">
         <dialogexit status="1" reason="Dialog successfully completed">
            <promptinfo duration="10342" termmode="completed"/>
         </dialogexit>
      </event>
   </mscivr>


F2. AS -> MS (CFW 200, ACK to 'CONTROL event')
-----------------------------------------------
   CFW 62726ace1660 200
```

6.3. Voice Mail

TOC

Another application that typically makes use of the services a MS can provide is Voice Mail. In fact, while it is clear that many of its features are part of the application logic (e.g. the mapping of a URI with a specific user's voice mailbox, the list of messages and their properties, and so on), the actual media work is accomplished through the MS. Features needed by a VoiceMail application include the ability to record a stream and play it back anytime later, give verbose announcements regarding the status of the application, controlling the playout of recorded messages by means of VCR controls and so on, all features which are supported by the MS through the IVR package. Without delving into the details of a full VoiceMail application and all its possible use cases, this section will cover a specific scenario, trying to deal with as many as possible interactions that may happen between the AS and the MS in such a context. The covered scenario, depicted as a sequence diagram in Figure 23 (Voice Mail: Framework Transactions), will be the following:

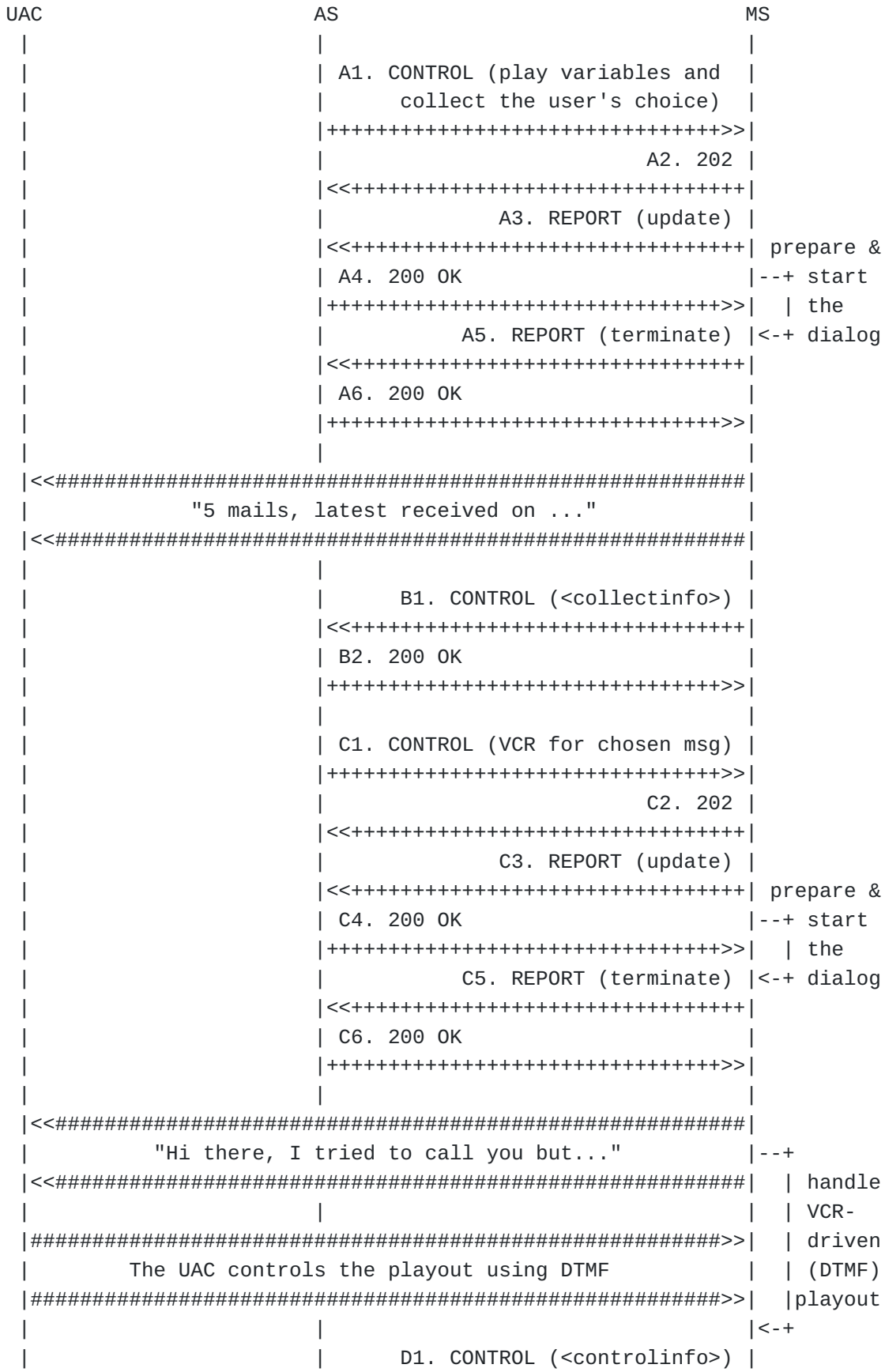
1. The UAC INVITEs a URI associated with his mailbox, and the AS follows the already explained procedure to have the UAC negotiate a new media session with the MS;

2. The UAC is first prompted an announcement giving him a count of the available new messages in the mailbox, and the date and

time the last message has been received; after that, the UAC must choose which message to access by sending a DTMF tone;

3. The UAC is then presented with a VCR controlled announcement, in which the chosen received mail is played back to him; VCR controls allow him to navigate through the prompt.

This is a quite oversimplified scenario, considering it doesn't even allow the UAC to delete old messages, organize them and the like, but just to choose which received message to play. Nevertheless, it gives us the chance to deal with variable announcements and VCR controls, two typical features a Voice Mail application would almost always take advantage of. Besides, other features a Voice Mail application would rely upon (e.g. recording streams, event driven IVR menus and so on) have aready been introduced in previous sections, and so representing them would be redundant.


UAC
AS
MS
A1. CONTROL (play variables and collect the user's choice)
A2. 202
A3. REPORT (update)
prepare & start the dialog
A4. 200 OK
A5. REPORT (terminate)
A6. 200 OK
"5 mails, latest received on ..."
B1. CONTROL (<collectinfo>)
B2. 200 OK
C1. CONTROL (VCR for chosen msg)
C2. 202
C3. REPORT (update)
prepare & start the dialog
C4. 200 OK
C5. REPORT (terminate)
C6. 200 OK
"Hi there, I tried to call you but..."
handle VCR-driven (DTMF) playout
The UAC controls the playout using DTMF
D1. CONTROL (<controlinfo>)

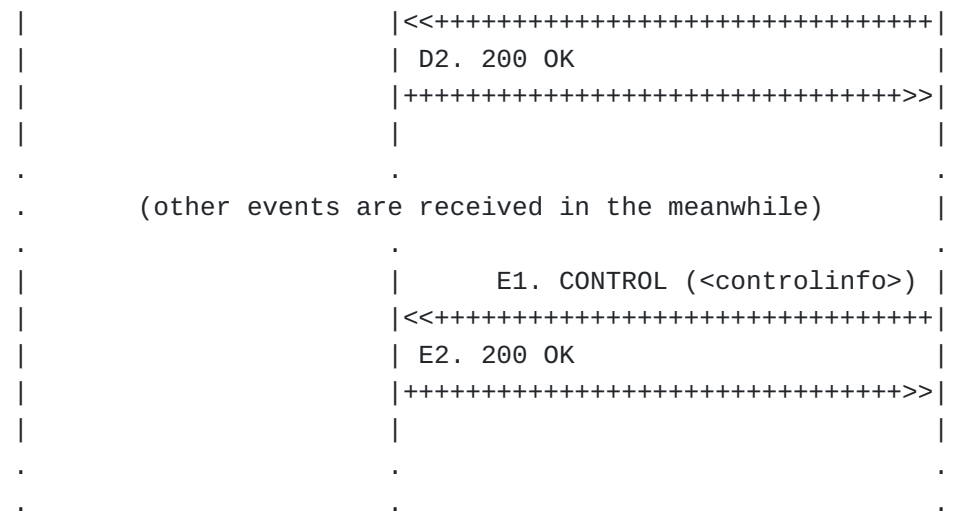


Figure 23: Voice Mail: Framework Transactions

The framework transaction steps are described in the following lines:

*The first transaction (A1) is addressed to the IVR package (msc-ivr); it is basically a 'promptandcollect' dialog, but with a slight difference: some of the prompts to play are actual audio files, for which a URI is provided (media src="xxx"), while others are so-called 'variable' prompts; these 'variable' prompts are actually constructed by the MS itself according to the directives provided by the AS; in this example, this is the sequence of prompts that is requested by the AS:

1. play a wav file ("you have...");

2. play a digit ("five..."), by building it (variable: digit=5);

3. play a wav file ("messages...");

4. play a wav file ("last...");

5. play a wav file ("received...");

6. play a date ("13th of october 2008..."), by building it (variable: date with a ymd=year/month/day format);

7. play a wav file ("at...");

8. play a time ("13:38..."), by building it (variable: time with a t24=24 hour day format);

9. play a wav file ("o' clock...");

a DTMF collection is requested as well (<collect>) to be taken after the prompts have been played; the AS is only interested to a single digit (maxdigits=1);

*the transaction is extended by the MS (A2, A3, A4) and, in case everything went fine (i.e. the MS retrieved all the audio files and successfully built the variable ones), the dialog is started; its start is reported, together with the associated identifier (5db01f4) to the AS in a terminating REPORT message (A5);

*the AS acks the REPORT (A6), and waits for the dialog to end in order to retrieve the results it is interested to (in this case, the DTMF tone the UAC chooses, since it will affect which message will have to be played subsequently);

*the UAC hears the prompts and chooses a message to play; in this example, he wants to listen to the first message, and so digits 1; the MS intercepts this tone, and notifies it to the AS in a newly created CONTROL event message (B1); this CONTROL includes information about how each single requested operations ended (<promptinfo> and <collectinfo>); specifically, the event states that the prompt ended normally (termmode=completed) and that the subsequently collected tone is 1 (dtmf=1); the AS acks the event (B2), since the dialogid provided in the message is the same as the one of the previously started dialog;

*at this point, the AS makes use of the value retrieved from the event to proceed in its business logic; it decides to present the UAC with a VCR-controllable playout of the requested message; this is done with a new request to the IVR package (C1), which contains two operations: <prompt> to address the media file to play (an old recording), and <control> to instruct the MS about how the playout of this media file shall be controlled via DTMF tones provided by the UAC (in this example, different DTMF digits are associated with different actions, e.g. pause/resume, fast forward, rewind and so on); besides, the AS also subscribes to DTMF events related to this control operation (matchmode=control), which means that the MS is to trigger an event anytime a DTMF associated with a control operation (e.g. 7=pause) is intercepted;

*the MS prepares the dialog, notifying about the transaction being extended (C2, C3, C4) and, when the playout starts, notifies it in a terminating REPORT (C5), which is acked by the AS (C6); at this point, the UAC is presented with the prompt, and can make use of DTMF digits to control the playback;

*as explained previously, any DTMF associated with a VCR operation is then reported to the AS, together with a timestamp stating when the event happened; an example is provided (D1) in which the UAC pressed the fast forward key (6) at a specific time; of course, as for any other MS-generated event, the AS acks it (D2);

*when the playback ends (whether because the media reached its termination, or because any other interruption occurred), the MS triggers a concluding event with information about the whole dialog (E1); this event, besides including information about the prompt itself (<promptinfo>), also includes information related to the VCR operations (<controlinfo>), that is, all the VCR controls the UAC made use of (in the example fastforward/rewind/pause/resume) and when it happened; the final ack by the AS (E2) concludes the scenario.

```
A1. AS -> MS (CFW CONTROL, play and collect)
-------------------------------------------
   CFW 2f931de22820 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 830

   <mscivr version="1.0">
      <dialogstart connectionid="10514b7f~6a900179">
         <dialog>
            <prompt>
               <media \
           src="http://www.pippozzoserver.org/prompts/vm-youhave.wav" \
           type="audio/x-wav"/>
               <variable value="5" type="digits"/>
               <media \
           src="http://www.pippozzoserver.org/prompts/vm-messages.wav" \
           type="audio/x-wav"/>
               <media \
           src="http://www.pippozzoserver.org/prompts/vm-last.wav" \
           type="audio/x-wav"/>
               <media \
           src="http://www.pippozzoserver.org/prompts/vm-received.wav" \
           type="audio/x-wav"/>
               <variable value="2008-10-13" type="date" format="ymd"/>
               <media \
           src="http://www.pippozzoserver.org/prompts/at.wav" \
           type="audio/x-wav"/>
               <variable value="13:38" type="time" format="t24"/>
               <media \
           src="http://www.pippozzoserver.org/prompts/oclock.wav" \
           type="audio/x-wav"/>
            </prompt>
            <collect maxdigits="1" escapekey="*" \
                     cleardigitbuffer="true"/>
         </dialog>
      </dialogstart>
   </mscivr>


A2. AS <- MS (CFW 202)
----------------------
   CFW 2f931de22820 202


A3. AS <- MS (CFW REPORT update)
--------------------------------
```

```
   CFW 2f931de22820 REPORT
   Seq: 1
   Status: update
   Timeout: 10


A4. AS -> MS (CFW 200, ACK to 'REPORT update')
----------------------------------------------
   CFW 2f931de22820 200
   Seq: 1


A5. AS <- MS (CFW REPORT terminate)
-----------------------------------
   CFW 2f931de22820 REPORT
   Seq: 2
   Status: terminate
   Timeout: 15
   Content-Type: application/msc-ivr+xml
   Content-Length: 137

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
     <response status="200" reason="Dialog started" dialogid="5db01f4"/>
   </mscivr>


A6. AS -> MS (CFW 200, ACK to 'REPORT terminate')
-------------------------------------------------
   CFW 2f931de22820 200
   Seq: 2


B1. AS <- MS (CFW CONTROL event)
--------------------------------
   CFW 7c97adc41b3e CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 270

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <event dialogid="5db01f4">
         <dialogexit status="1" reason="Dialog successfully completed">
            <promptinfo duration="11713" termmode="completed"/>
            <collectinfo dtmf="1" termmode="match"/>
         </dialogexit>
      </event>
   </mscivr>


B2. AS -> MS (CFW 200, ACK to 'CONTROL event')
```

```
-----------------------------------------------
   CFW 7c97adc41b3e 200


C1. AS -> MS (CFW CONTROL, VCR)
-------------------------------
   CFW 3140c24614bb CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 386

   <mscivr version="1.0">
      <dialogstart connectionid="10514b7f~6a900179">
         <dialog>
            <prompt bargein="false">
               <media \
  src="http://www.pippozzoserver.org/recordings/recording-4ca9fc2.mpg"/>
            </prompt>
            <control gotostartkey="1" gotoendkey="3" \
                     ffkey="6" rwkey="4" pausekey="7" resumekey="9" \
                     volupkey="#" voldnkey="*"/>
            </dialog>
         <subscribe>
            <dtmfsub matchmode="control"/>
         </subscribe>
      </dialogstart>
   </mscivr>


C2. AS <- MS (CFW 202)
----------------------
   CFW 3140c24614bb 202


C3. AS <- MS (CFW REPORT update)
--------------------------------
   CFW 3140c24614bb REPORT
   Seq: 1
   Status: update
   Timeout: 10


C4. AS -> MS (CFW 200, ACK to 'REPORT update')
----------------------------------------------
   CFW 3140c24614bb 200
   Seq: 1


C5. AS <- MS (CFW REPORT terminate)
-----------------------------------
```

```
   CFW 3140c24614bb REPORT
   Seq: 2
   Status: terminate
   Timeout: 25
   Content-Type: application/msc-ivr+xml
   Content-Length: 137

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
     <response status="200" reason="Dialog started" dialogid="3e936e0"/>
   </mscivr>


C6. AS -> MS (CFW 200, ACK to 'REPORT terminate')
-------------------------------------------------
   CFW 3140c24614bb 200
   Seq: 2


D1. AS <- MS (CFW CONTROL event, dtmfnotify)
--------------------------------------------
   CFW 361840da0581 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 179

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <event dialogid="3e936e0">
         <dtmfnotify matchmode="control" dtmf="6" \
                     timestamp="2008-10-15T15:50:36Z"/>
      </event>
   </mscivr>


D2. AS -> MS (CFW 200, ACK to 'CONTROL event')
----------------------------------------------
   CFW 361840da0581 200


      [..] The other VCR DTMF notifications are skipped for brevity [..]


E1. AS <- MS (CFW CONTROL event, dialogexit)
--------------------------------------------
   CFW 3ffab81c21e9 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 485

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <event dialogid="3e936e0">
```

```
            <dialogexit status="1" reason="Dialog successfully completed">
               <promptinfo duration="10270" termmode="completed"/>
               <controlinfo>
                  <controlmatch dtmf="6" timestamp="2008-10-15T15:50:36Z"/>
                  <controlmatch dtmf="4" timestamp="2008-10-15T15:50:37Z"/>
                  <controlmatch dtmf="7" timestamp="2008-10-15T15:50:38Z"/>
                  <controlmatch dtmf="9" timestamp="2008-10-15T15:50:40Z"/>
               </controlinfo>
            </dialogexit>
         </event>
      </mscivr>


   E2. AS -> MS (CFW 200, ACK to 'CONTROL event')
   ----------------------------------------------
      CFW 3ffab81c21e9 200
```

6.4. Conferencing

TOC

One of the most important services the MS must be able to provide is mixing. This involves mixing media streams from different sources, and delivering the resulting mix(es) to each interested party, often according to per-user policies, settings and encoding. A typical scenario involving mixing is of course media conferencing. In such a scenario, the media sent by each participant is mixed, and each participant typically receives the overall mix excluding its own contribtion and encoded in the format it negotiated. This example points out in a quite clear way how mixing must take care of the profile of each involved entity.
A media perspective of such a scenario is depicted in Figure 24 (Conference: Media Perspective).

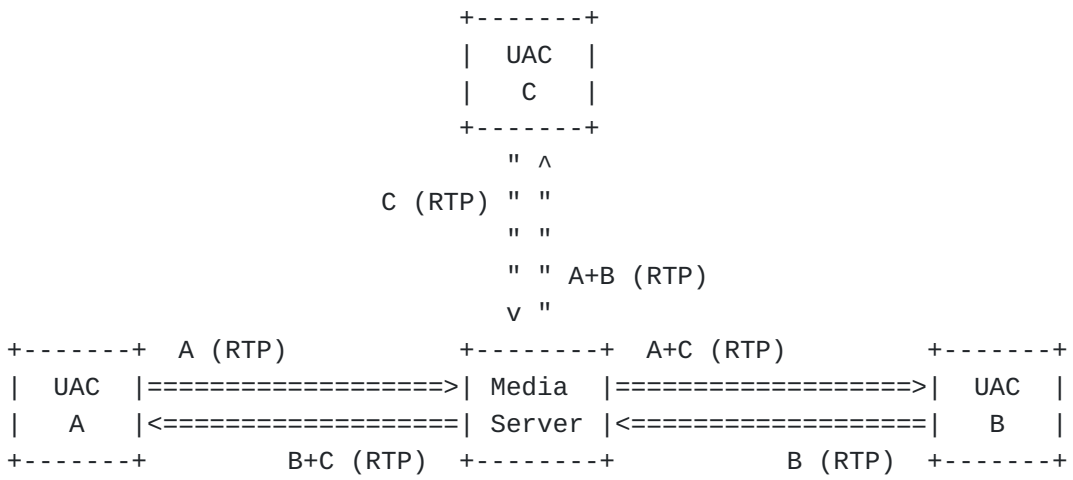


Figure 24: Conference: Media Perspective

From the framework point of view, when the UACs' legs are not attached to anything yet, what appears is described in Figure 25 (Conference: UAC Legs not attached): since there are no connections involving the UACs yet, the frames they might be sending are discarded, and nothing is sent back to them either (except for silence, if it is requested to be transmitted).

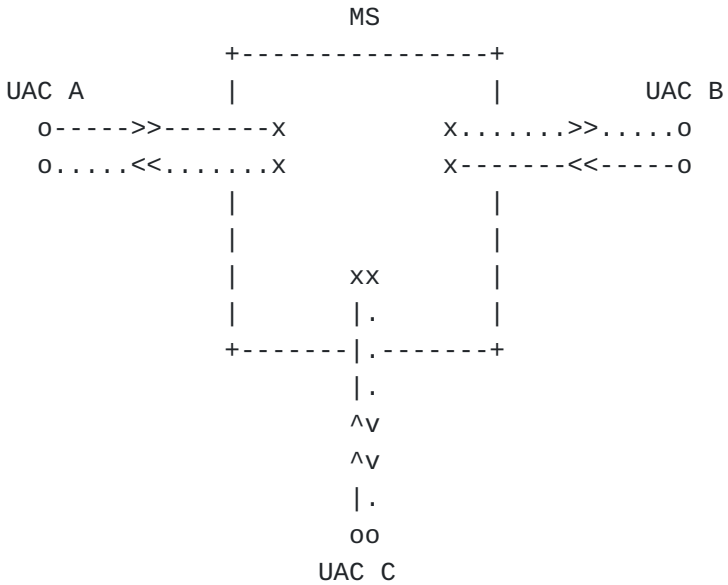


Figure 25: Conference: UAC Legs not attached

The next subsections will cover several typical scenarios involving mixing and conferencing as a whole, specifically:

1. Simple Bridging, where the scenario will be a very basic (i.e. no "special effects", just mixing involved) conference between two and more participants;

2. Rich Conference Scenario, which enriches the Simple Bridging scenario by adding additional features typically found in conferencing systems (e.g. DTMF collection for PIN-based conference access, private and global announcements, recordings and so on);

3. Coaching Scenario, a more complex scenario which involves per-user mixing (cusomers, agents and coaches don't get all the same mixes);

4. Sidebars Scenario, which adds more complexity to the previous conferencing scenarios by involving sidebars (i.e. separate conference instances that only exist within the context of a parent conference instance) and the custom media delivery that follows.

All the above mentioned scenarios depend on the availability of a mixing entity. Such an entity is provided in the Media Control Channel Framework by the conferencing package. This package in fact, besides allowing for the joining of media sources between each other as seen in the Direct Echo Test section, enables the creation of abstract connections that can be joined to multiple connections: these abstract connections, called conferences, mix the contribution of each attached connection and feed them accordingly (e.g. a connection with 'sendrecv' property would be able to contribute to the mix and to listen to it, while a connection with a 'recvonly' property would only be able to listen to the overall mix but not to actively contribute to it).
That said, each of the above mentioned scenarios will start more or less in the same way: by the creation of a conference connection (or more than one, as needed in some cases) to be subsequently referred to when it comes to mixing. A typical framework transaction to crete a new conference instance in the Media Control Channel Framework is depicted in Figure 26 (Conference: Framework Transactions):

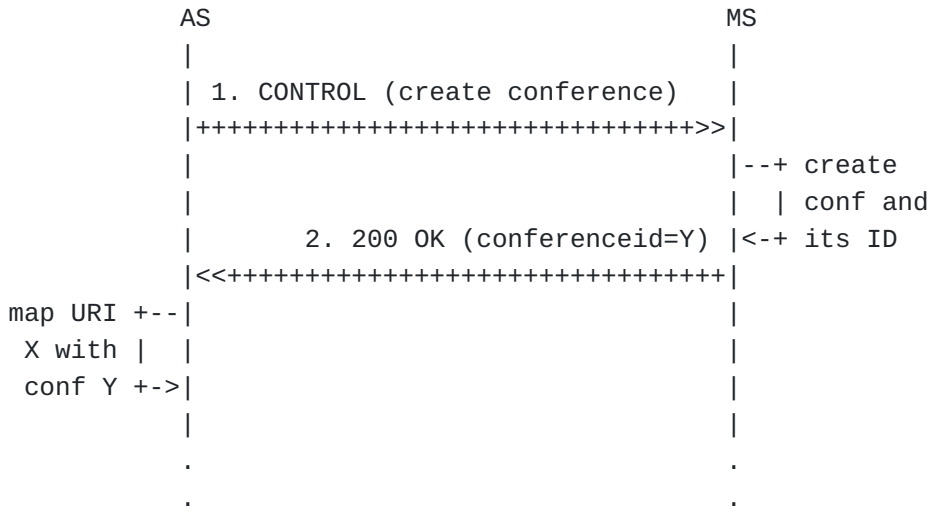


Figure 26: Conference: Framework Transactions

The call flow is quite straightforward, and can typically be summarized in the following steps:

- The AS invokes the creation of a new conference instance by means of a CONTROL request (1); this request is addressed to the conferencing package (msc-mixer/1.0) and contains in the body the directive (createconference) with all the desired settings for it; in the example, the mixing policy is to mix the five (reserved-talkers) loudest speakers (nbest), while ten listeners at max are allowed; video settings are configured, including the mechanism used to select active video sources (controller, meaning the AS will explicitly instruct the MS about it) and details about the video layouts to make available; in this example, the AS is instructing the MS to use a single-view layout when only one video source is active, to pass to a quad-view layout when at least two video sources are active, and to use a 5x1 layout whenever the number of sources is at least five; finally, the AS also subscribes to the "active-talkers" event, which means it wants to be informed (at a rate of 4 seconds) whenever an active participant is speaking;

- The MS creates the conference instance assigning a unique identifier to it (6146dd5), and completes the transaction with a 200 response (2);

- At this point, the requested conference instance is active and ready to be used by the AS; it is then up to the AS to integrate the use of this identifier in its application logic.

```
1. AS -> MS (CFW CONTROL)
-------------------------
   CFW 3032e5fb79a1 CONTROL
   Control-Package: msc-mixer/1.0
   Content-Type: application/msc-mixer+xml
   Content-Length: 452

   <mscmixer version="1.0">
     <createconference reserved-talkers="5" reserved-listeners="10">
        <audio-mixing mix-type="nbest"/>
        <video-switch type="controller"/>
        <video-layouts>
          <video-layout min-participants='1'>single-view</video-layout>
          <video-layout min-participants='2'>quad-view</video-layout>
          <video-layout min-participants='5'>multiple-5x1</video-layout>
        </video-layouts>
        <subscribe>
           <active-talkers-sub interval="4"/>
        </subscribe>
     </createconference>
   </mscmixer>


2. AS <- MS (CFW 200)
---------------------
   CFW 3032e5fb79a1 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 151

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Conference created" \
                conferenceid="6146dd5"/>
   </mscmixer>
```

6.4.1. Simple Bridging

TOC

As already introduced before, the simplest use an AS can make of a conference instance is simple bridging. In this scenario, the conference instance just acts as a bridge for all the participants that are attached to it. The bridge takes care of transcoding, if needed (in general, different participants may make use of different codecs for their streams), echo cancellation (each participant will receive the

overall mix excluding their own contribution) and per-participant mixing (each participant may receive different mixed streams, according to what it needs/is allowed to send/receive). This assumes of course that each interested participant must be joined somehow to the bridge in order to indirectly communicate with the other paricipants. From the media perspective, the scenario can be seen as depicted in Figure 27 (Conference: Simple Bridging).

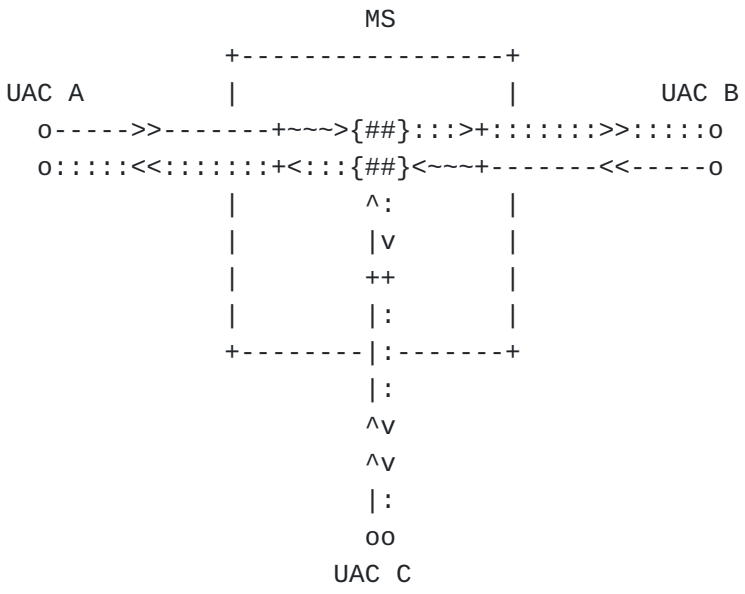


Figure 27: Conference: Simple Bridging

In the framework, the first step is obviously to create a new conference instance as seen in the introductory section (Figure 26 (Conference: Framework Transactions)). Assuming a conference instance has already been created, bridging participants to it is quite straightforward, and can be accomplished as already seen in the Direct Echo Test Scenario: the only difference here is that each participant is not directly connected to itself (Direct Echo) or another UAC (Direct Connection) but to the bridge instead. Figure 28 (Simple Bridging: Framework Transactions (1)) shows the example of two different UACs joining the same conference: the example, as usual, hides the previous interaction between each of the two UACs and the AS, and instead focuses on what the AS does to actually join the participants to the bridge so that they can interact in a conference.

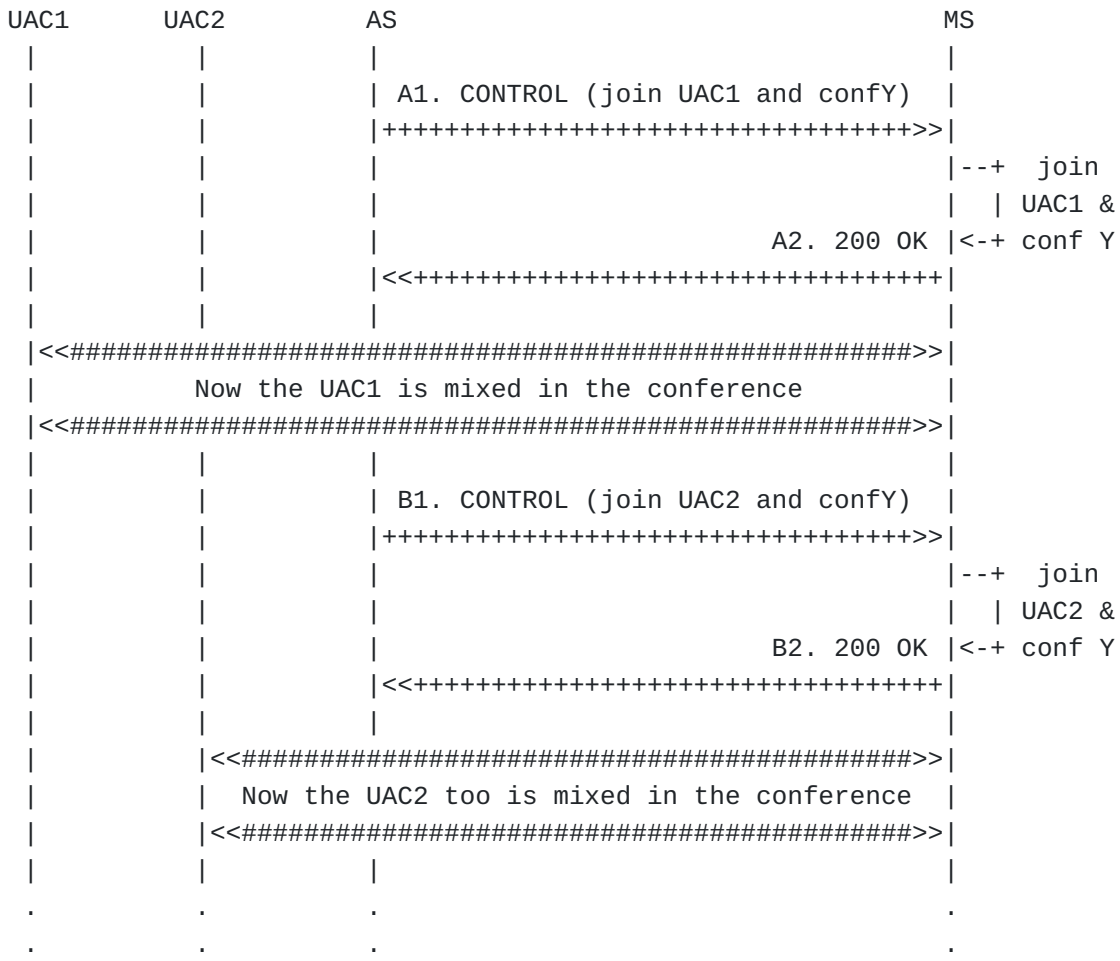


Figure 28: Simple Bridging: Framework Transactions (1)

The framework transaction steps are actually quite trivial to understand, since they're very similar to some previously described scenarios. What the AS does is just joining both UAC1 (id1 in A1) and UAC2 (id1 in B1) to the conference (id2 in both transactions). As a result of these two operations, both the UACs are mixed in the conference. Since no <stream> is explicitly provided in any of the transactions, all the media from the UACs (audio/video) are attached to the conference (as long as the conference has been properly configured to support both, of course).

```
A1. AS -> MS (CFW CONTROL)
-------------------------
   CFW 434a95786df8 CONTROL
   Control-Package: msc-mixer/1.0
   Content-Type: application/msc-mixer+xml
   Content-Length: 80

   <mscmixer version="1.0">
      <join id1="e1e1427c~1c998d22"
            id2="6146dd5"/>
   </mscmixer>


A2. AS <- MS (CFW 200 OK)
-------------------------
   CFW 434a95786df8 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
   </mscmixer>


B1. AS -> MS (CFW CONTROL)
-------------------------
   CFW 5c0cbd372046 CONTROL
   Control-Package: msc-mixer/1.0
   Content-Type: application/msc-mixer+xml
   Content-Length: 80

   <mscmixer version="1.0">
      <join id1="10514b7f~6a900179"
            id2="6146dd5"/>
   </mscmixer>


B2. AS <- MS (CFW 200 OK)
-------------------------
   CFW 5c0cbd372046 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
```

```
        </mscmixer>
```

Once one or more participants have been attached to the bridge, their connections and how their media are handled by the bridge can be dynamically manipulated by means of another directive, called <modifyjoin>: a typical use case for this directive is the change of direction of an existing media (e.g. a previously speaking participant is muted, which means its media direction changes from 'sendrecv' to 'recvonly'). Figure 29 (Simple Bridging: Framework Transactions (2)) shows how a framework transaction requesting such a directive might appear.

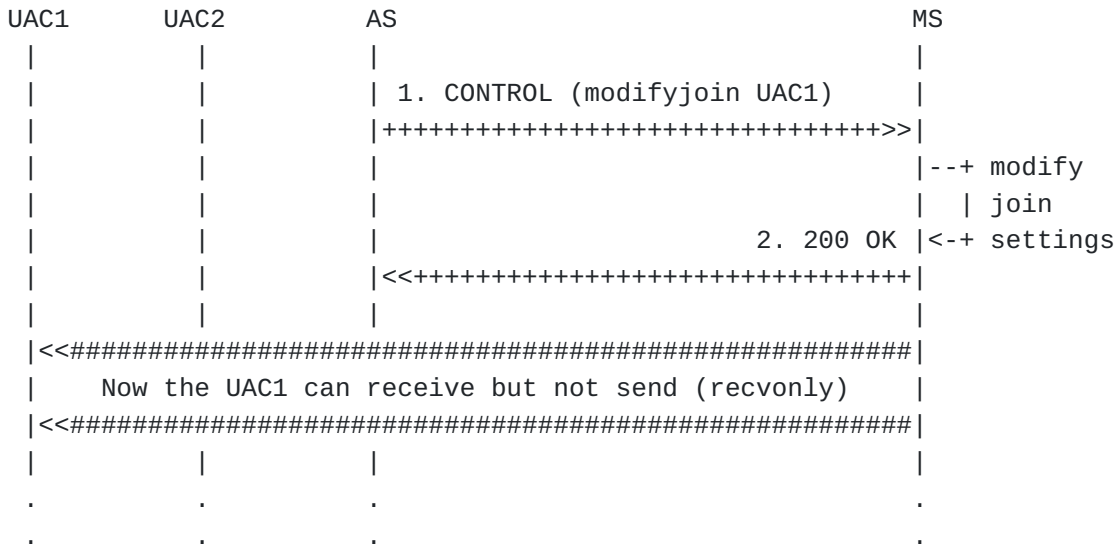


Figure 29: Simple Bridging: Framework Transactions (2)

The directive used to modify an existing join configuration is <modifyjoin>, and its syntax is exactly the same as the one required in <join> instructions. In fact, the same syntax is used for identifiers (id1/id2). Whenever a modifyjoin is requested and id1 and id2 address one or more joined connections, the AS is requesting a change of the join configuration. In this case, the AS instructs the MS to mute (stream=audio, direction=recvonly) UAC1 (id1=UAC1) in the conference (id2) it has been attached to previously. Any other connection existing between them is left untouched.

```
1. AS -> MS (CFW CONTROL)
-------------------------
   CFW 57f2195875c9 CONTROL
   Control-Package: msc-mixer/1.0
   Content-Type: application/msc-mixer+xml
   Content-Length: 142

   <mscmixer version="1.0">
      <modifyjoin id1="e1e1427c~1c998d22" id2="6146dd5">
         <stream media="audio" direction="recvonly"/>
      </modifyjoin>
   </mscmixer>


2. AS <- MS (CFW 200 OK)
------------------------
   CFW 57f2195875c9 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 123

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join modified"/>
   </mscmixer>
```

6.4.2. Rich Conference Scenario

TOC

The previous scenario can be enriched with additional features often found in existing conferencing systems. Typical examples include IVR-based menus (e.g. the DTMF collection for PIN-based conference access), partial and complete recordings in the conference (e.g. for the "state your name" functionality and recording of the whole conference), private and global announcements and so on. All of this can be achieved by means of the functionality provided by the MS. In fact, even if the conferencing and IVR features come from different packages, the AS can interact with both of them and achieve complex results by correlating the results of different transactions in its application logic.
From the media and framework perspective, a typical rich conferencing scenario can be seen as it is depicted in Figure 30 (Conference: Rich Conference Scenario).

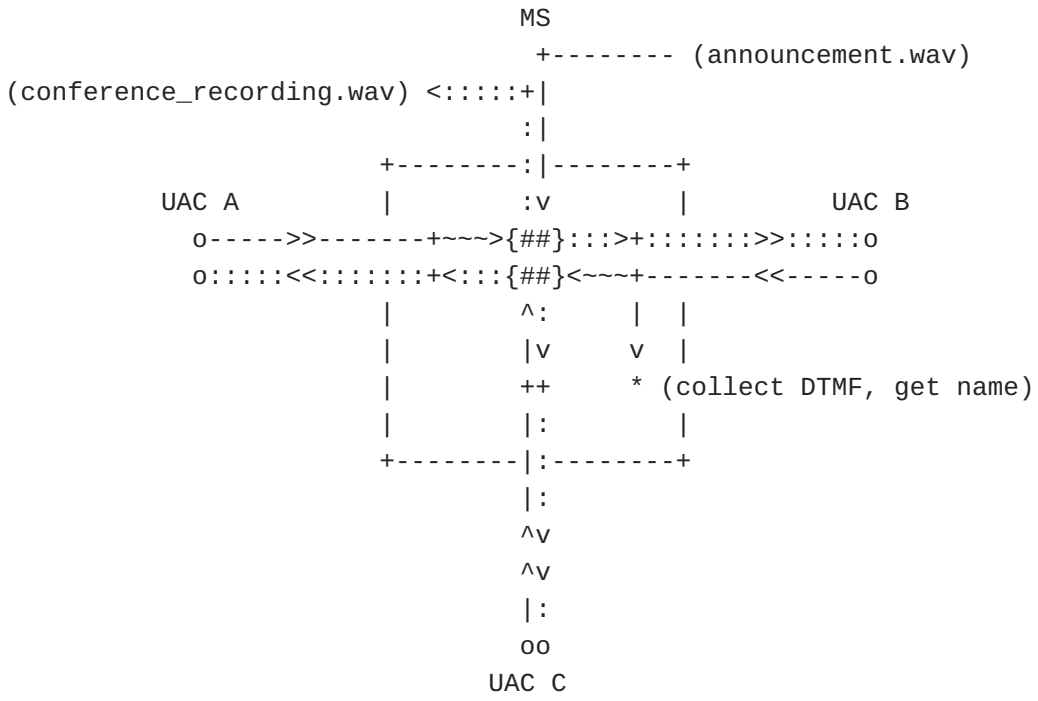


Figure 30: Conference: Rich Conference Scenario

To identify a single sample scenario, let's consider this sequence for a participant joining a conference (which again we assume has already been created):

1. The UAC as usual INVITEs a URI associated with a conference, and the AS follows the already explained procedure to have the UAC negotiate a new media session with the MS;

2. The UAC is presented with an IVR menu, in which it is requested to digit a PIN code to access the conference;

3. If the PIN is correct, the UAC is asked to state its name so that it can be recorded;

4. The UAC is attached to the conference, and the previously recorded name is announced globally to the conference to announce its arrival.

Figure 31 (Rich Conference Scenario: Framework Transactions) shows a single UAC joining a conference: the example, as usual, hides the previous interaction between the UAC and the AS, and instead focuses on what the AS does to actually interact with the participant and join it to the conference bridge.

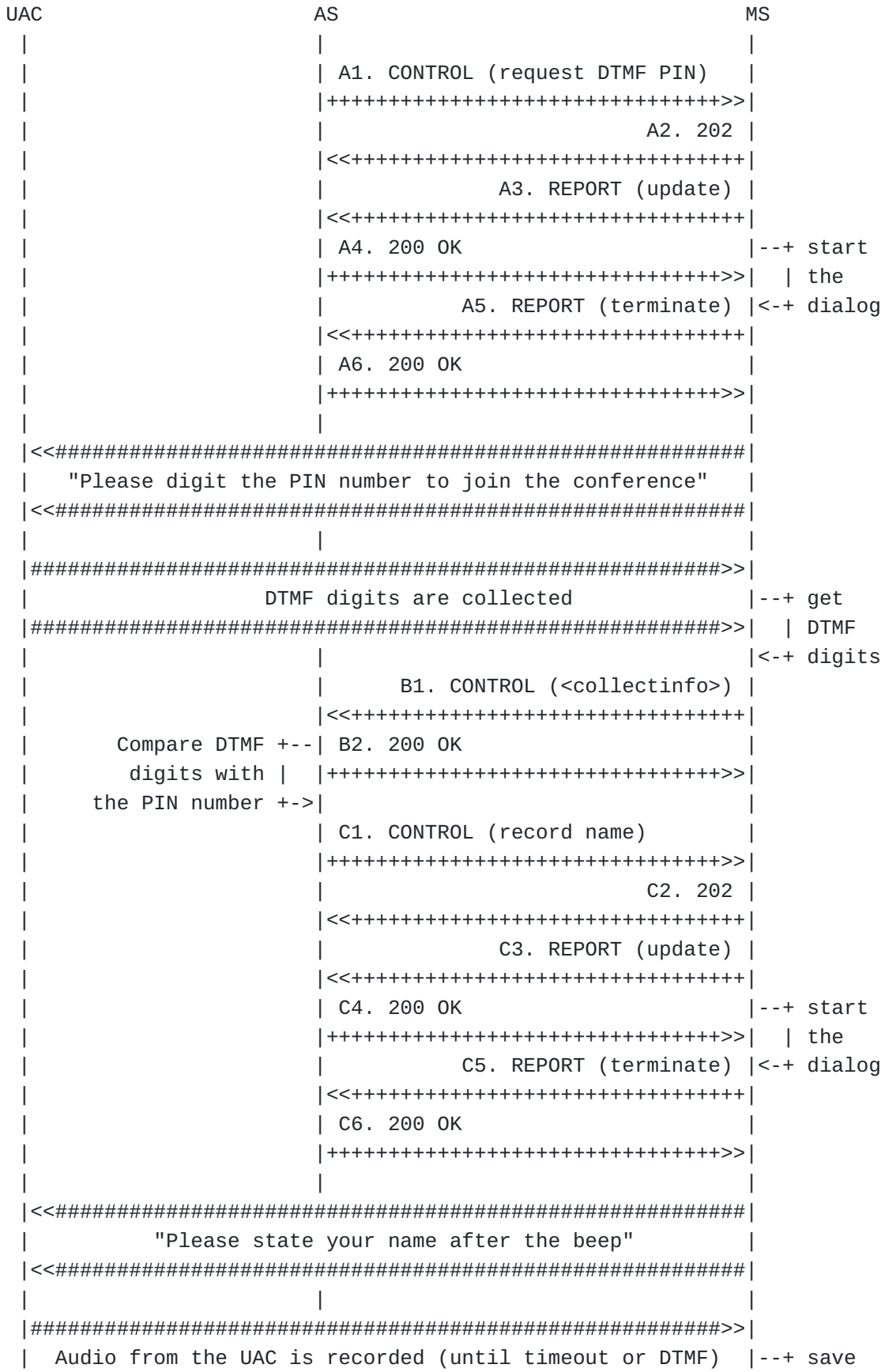

UAC
AS
MS
A1. CONTROL (request DTMF PIN)
A2. 202
A3. REPORT (update)
A4. 200 OK
--+ start
A5. REPORT (terminate)
<-+ dialog
A6. 200 OK
"Please digit the PIN number to join the conference"
DTMF digits are collected
--+ get
<-+ DTMF
<-+ digits
B1. CONTROL (<collectinfo>)
B2. 200 OK
Compare DTMF
digits with
the PIN number
C1. CONTROL (record name)
C2. 202
C3. REPORT (update)
C4. 200 OK
--+ start
C5. REPORT (terminate)
<-+ dialog
C6. 200 OK
"Please state your name after the beep"
Audio from the UAC is recorded (until timeout or DTMF)
--+ save

Figure 31: Rich Conference Scenario: Framework Transactions

As it can be evinced from the sequence diagram above, the AS, in its business logic, correlates the results of different transactions, addressed to different packages, to implement a more complex conferencing scenario than the Simple Bridging previously described. The framework transaction steps are the following:

Since this is a private conference, the UAC is to be presented with a request for a password, in this case a PIN number; to do so, the AS instructs the MS (A1) to collect a series of DTMF digits from the specified UAC (connectionid=UAC); the request includes both a voice message (<prompt>) and the described digit collection context (<collect>); the PIN is assumed to be a 4-digit number, and so the MS has to collect at max 4 digits (maxdigits=4); the DTMF digit buffer must be cleared before collecting (cleardigitbuffer=true) and the UAC can make use of the star key to restart the collection (escapekey=), e.g. in case it is aware he miswrote any of the digits and wants to start again;

*the transaction goes on as usual (A2, A3, A4, A5, A6), with the transaction being extended, and the dialog start being notified in a REPORT terminate; after that, the UAC is actually presented with the voice message, and is subsequently requested to insert the required PIN number;

*we assume UAC wrote the correct PIN number (1234), which is reported by the MS to the AS by means of the usual MS-generated CONTROL event (B1); the AS correlates this event to the previously started dialog by checking the referenced dialogid (06d1bac) and acks the event (B2); it then extracts the information it needs from the event (in this case, the digits provided by the MS) from the <controlinfo> container (dtmf=1234) and verifies it is correct;

*since the PIN is correct, the AS can proceed towards the next step, that is asking the UAC to state his name, in order to play the recording subsequently on the conference to report the new participant; again, this is done with a request to the IVR package (C1); the AS instructs the MS to play a voice message ("say your name after the beep"), to be followed by a recording of only the audio from the UAC (in stream, media=audio/sendonly, while media=video/inactive); a beep must be played right before the recording starts (beep=true), and the recording must only last 3 seconds (maxtime=3s) since it is only needed as a brief announcement;

*without delving again into the details of a recording-related transaction (C2/C3/C4/C5/C6), the AS finally gets an URI to the requested recording (D1, acked in D2);

*at this point, the AS attaches the UAC (id1) to the conference (id2) just as explained for Simple Bridging (E1/E2);

*finally, to notify the other participants that a new participant has arrived, the AS requests a global announcement on the conference; this is a simple <prompt> request to the IVR package (F1) just as the ones explained in previous sections, but with a slight difference: the target of the prompt is not a connectionid (a media connection) but the conference itself (conferenceid=6146dd5); as a result of this transaction, the announcement would be played on all the media connections attached to the conference which are allowed to receive media from it; the AS specifically requests two media files to be played:

1. the media file containing the recorded name of the new user as retrieved in D1 ("Simon...");

2. a pre-recorded media file explaining what happened ("... has joined the conference");

the transaction then takes its usual flow (F2/F3/F4/F5/F6), and the event notifying the end of the announcement (G1, acked in G2) concludes the scenario.

A1. AS -> MS (CFW CONTROL, collect)

```
   CFW 50e56b8d65f9 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 274

   <mscivr version="1.0">
     <dialogstart connectionid="10514b7f~6a900179">
        <dialog>
          <prompt>
              <media \
           src="http://www.pippozzoserver.org/prompts/conf-getpin.wav" \
           type="audio/x-wav"/>
          </prompt>
          <collect maxdigits="4" escapekey="*" cleardigitbuffer="true"/>
        </dialog>
     </dialogstart>
   </mscivr>
```

A2. AS <- MS (CFW 202)

```
   CFW 50e56b8d65f9 202
```

A3. AS <- MS (CFW REPORT update)

```
   CFW 50e56b8d65f9 REPORT
   Seq: 1
   Status: update
   Timeout: 10
```

A4. AS -> MS (CFW 200, ACK to 'REPORT update')

```
   CFW 50e56b8d65f9 200
   Seq: 1
```

A5. AS <- MS (CFW REPORT terminate)

```
   CFW 50e56b8d65f9 REPORT
   Seq: 2
   Status: terminate
   Timeout: 25
   Content-Type: application/msc-ivr+xml
   Content-Length: 137
```

```
   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
     <response status="200" reason="Dialog started" dialogid="06d1bac"/>
   </mscivr>


A6. AS -> MS (CFW 200, ACK to 'REPORT terminate')
-------------------------------------------------
   CFW 50e56b8d65f9 200
   Seq: 2


B1. AS <- MS (CFW CONTROL event)
--------------------------------
   CFW 166d68a76659 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 272

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <event dialogid="06d1bac">
         <dialogexit status="1" reason="Dialog successfully completed">
            <promptinfo duration="2312" termmode="completed"/>
            <collectinfo dtmf="1234" termmode="match"/>
         </dialogexit>
      </event>
   </mscivr>


B2. AS -> MS (CFW 200, ACK to 'CONTROL event')
----------------------------------------------
   CFW 166d68a76659 200


C1. AS -> MS (CFW CONTROL, record)
----------------------------------
   CFW 61fd484f196e CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 355

   <mscivr version="1.0">
      <dialogstart connectionid="10514b7f~6a900179">
         <dialog>
            <prompt>
               <media \
         src="http://www.pippozzoserver.org/prompts/conf-rec-name.wav" \
         type="audio/x-wav"/>
            </prompt>
            <record beep="true" maxtime="3s" vadinitial="false"/>
```

```
        </dialog>
        <stream media="audio" direction="sendonly"/>
        <stream media="video" direction="inactive"/>
      </dialogstart>
   </mscivr>


C2. AS <- MS (CFW 202)
----------------------
   CFW 61fd484f196e 202


C3. AS <- MS (CFW REPORT update)
--------------------------------
   CFW 61fd484f196e REPORT
   Seq: 1
   Status: update
   Timeout: 10


C4. AS -> MS (CFW 200, ACK to 'REPORT update')
----------------------------------------------
   CFW 61fd484f196e 200
   Seq: 1


C5. AS <- MS (CFW REPORT terminate)
-----------------------------------
   CFW 61fd484f196e REPORT
   Seq: 2
   Status: terminate
   Timeout: 25
   Content-Type: application/msc-ivr+xml
   Content-Length: 137

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
     <response status="200" reason="Dialog started" dialogid="1cf0549"/>
   </mscivr>


C6. AS -> MS (CFW 200, ACK to 'REPORT terminate')
-------------------------------------------------
   CFW 61fd484f196e 200
   Seq: 2


D1. AS <- MS (CFW CONTROL event)
--------------------------------
   CFW 3ec13ab96224 CONTROL
   Control-Package: msc-ivr/1.0
```

```
   Content-Type: application/msc-ivr+xml
   Content-Length: 384

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <event dialogid="1cf0549">
         <dialogexit status="1" reason="Dialog successfully completed">
            <promptinfo duration="3757" termmode="completed"/>
            <recordinfo \
recording="http://www.pippozzoserver.org/recordings/recording-1cf0549.wav" \
type="audio/x-wav" duration="3000" size="48044" termmode="maxtime"/>
         </dialogexit>
      </event>
   </mscivr>


D2. AS -> MS (CFW 200, ACK to 'CONTROL event')
----------------------------------------------
   CFW 3ec13ab96224 200


E1. AS -> MS (CFW CONTROL, join)
--------------------------------
   CFW 261d188b63b7 CONTROL
   Control-Package: msc-mixer/1.0
   Content-Type: application/msc-mixer+xml
   Content-Length: 80

   <mscmixer version="1.0">
      <join id1="10514b7f~6a900179" id2="6146dd5"/>
   </mscmixer>


E2. AS <- MS (CFW 200 OK)
-------------------------
   CFW 261d188b63b7 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
   </mscmixer>


F1. AS -> MS (CFW CONTROL, play)
--------------------------------
   CFW 718c30836f38 CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 299
```

```
   <mscivr version="1.0">
      <dialogstart conferenceid="6146dd5">
         <dialog>
            <prompt>
               <media \
  src="http://www.pippozzoserver.org/recordings/recording-1cf0549.wav" \
  type="audio/x-wav"/>
               <media \
  src="http://www.pippozzoserver.org/prompts/conf-hasjoin.wav" \
  type="audio/x-wav"/>
            </prompt>
         </dialog>
      </dialogstart>
   </mscivr>


F2. AS <- MS (CFW 202)
----------------------
   CFW 718c30836f38 202


F3. AS <- MS (CFW REPORT update)
--------------------------------
   CFW 718c30836f38 REPORT
   Seq: 1
   Status: update
   Timeout: 10


F4. AS -> MS (CFW 200, ACK to 'REPORT update')
----------------------------------------------
   CFW 718c30836f38 200
   Seq: 1


F5. AS <- MS (CFW REPORT terminate)
-----------------------------------
   CFW 718c30836f38 REPORT
   Seq: 2
   Status: terminate
   Timeout: 25
   Content-Type: application/msc-ivr+xml
   Content-Length: 137

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
     <response status="200" reason="Dialog started" dialogid="5f4bc7e"/>
   </mscivr>
```

```
F6. AS -> MS (CFW 200, ACK to 'REPORT terminate')
-------------------------------------------------
   CFW 718c30836f38 200
   Seq: 2


G1. AS <- MS (CFW CONTROL event)
--------------------------------
   CFW 6485194f622f CONTROL
   Control-Package: msc-ivr/1.0
   Content-Type: application/msc-ivr+xml
   Content-Length: 229

   <mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
      <event dialogid="5f4bc7e">
         <dialogexit status="1" reason="Dialog successfully completed">
            <promptinfo duration="1838" termmode="completed"/>
         </dialogexit>
      </event>
   </mscivr>


G2. AS -> MS (CFW 200, ACK to 'CONTROL event')
----------------------------------------------
   CFW 6485194f622f 200
```

6.4.3. Conferencing with Floor Control

TOC

TBD. (Add sequence diagrams and signaling issues; reference draft [I-D.miniero-bfcp-control-package] (Miniero, L., Romano, S., Even, R., and S. McGlashan, “A Binary Floor Control Protocol (BFCP) Control Package for the Media Control Channel Framework,” July 2008.))

(Figure not available yet.)

Figure 32: Floor Control: Media Perspective

(Figure not available yet.)

Figure 33: Floor Control: UAC Legs not attached

(Figure not available yet.)

Figure 34: Floor Control: UAC Legs mixed and attached

(Figure not available yet.)

Figure 35: Floor Control: Framework Transactions

6.4.4. Coaching Scenario

TOC

Another typical conference-based use case is the so called Coaching Scenario. In such a scenario, a customer (called A in the following example) places a call to a business call center. An agent (B) is assigned to the customer. Besides, a coach (C), unheard from the customer, provides the agent with whispered suggestions about what to say. This scenario is also described in RFC4579 [RFC4579] (Johnston, A. and O. Levin, "Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents," August 2006.).
As it can be evinced from the scenario description, per-user policies for media mixing and delivery, i.e who can hear what, are very important. The MS must make sure that only the agent can hear the coach's suggestions. Since this is basically a multiparty call (despite

what the customer may be thinking), a mixing entity is needed in order to accomplish the scenario requirements. To summarize:

- *the customer (A) must only hear what the agent (B) says;
- *the agent (B) must be able to hear both the customer (A) and the coach (C);
- *the coach (C) must be able to hear both the customer (A), in order to give the right suggestions, and the agent (B), in order to be aware of the whole conversation.

From the media and framework perspective, such a scenario can be seen as it is depicted in Figure 36 (Coaching Scenario: Media Perspective).

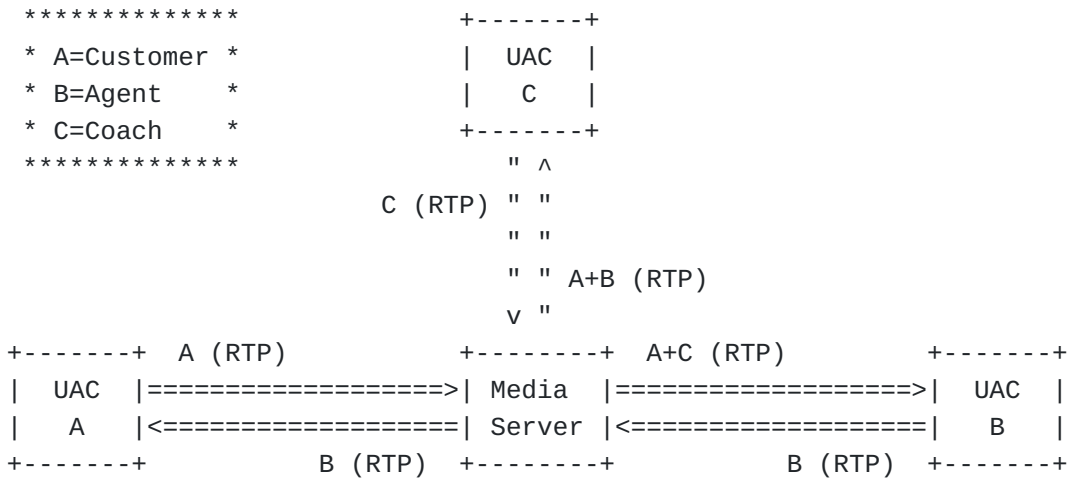


Figure 36: Coaching Scenario: Media Perspective

From the framework point of view, when the mentioned legs are not attached to anything yet, what appears is described in Figure 37 (Coaching Scenario: UAC Legs not attached).

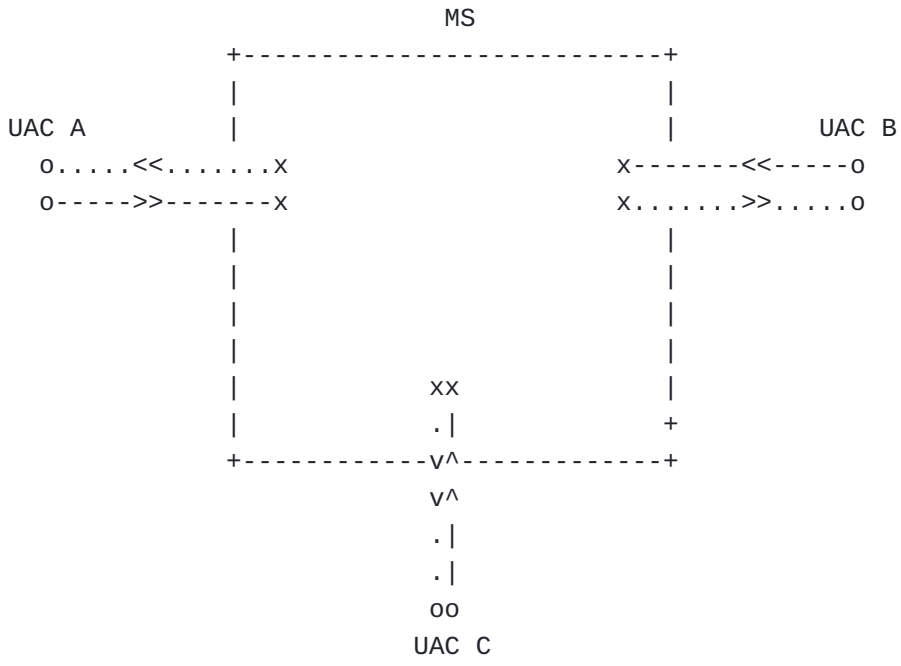


Figure 37: Coaching Scenario: UAC Legs not attached

What the scenario should look like from the framework point of view is instead depicted in Figure 38 (Coaching Scenario: UAC Legs mixed and attached). The customer receives media directly from the agent (recvonly), while all the three involved participants contribute to a hidden conference: of course the customer is not allowed to receive the mixed flows from the conference (sendonly), unlike the agent and the coach which must both be aware of the whole conversation (sendrecv).

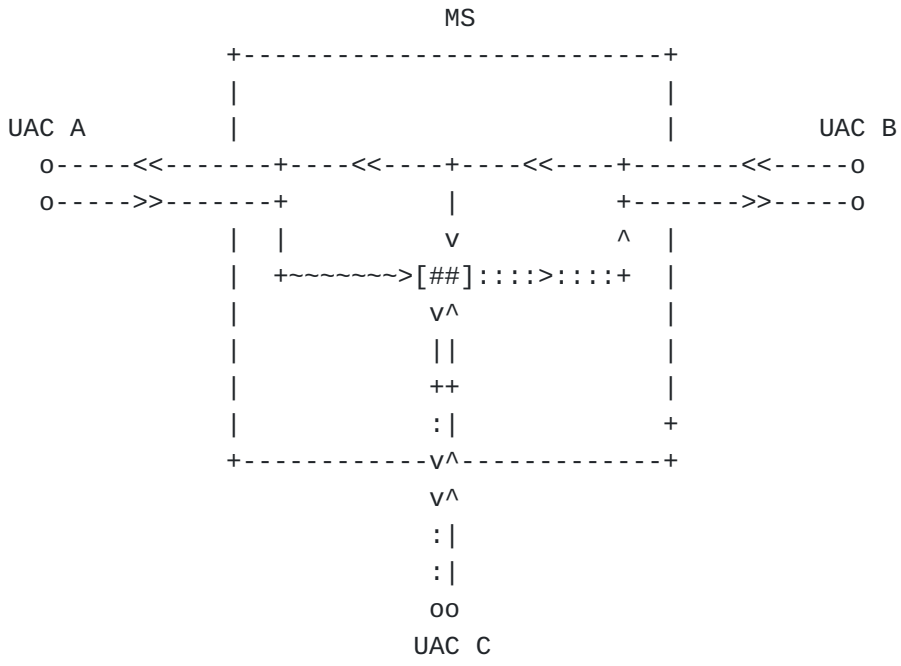


Figure 38: Coaching Scenario: UAC Legs mixed and attached

In the framework this can be achieved by means of the mixer control package, which, as already explained in previous sections, can be exploited whenever mixing and joining entities is needed. The needed steps can be summarized in the following steps:

1. first of all, a hidden conference is created;

2. then, all the three participants are attached to it, each with a custom mixing policy, specifically:

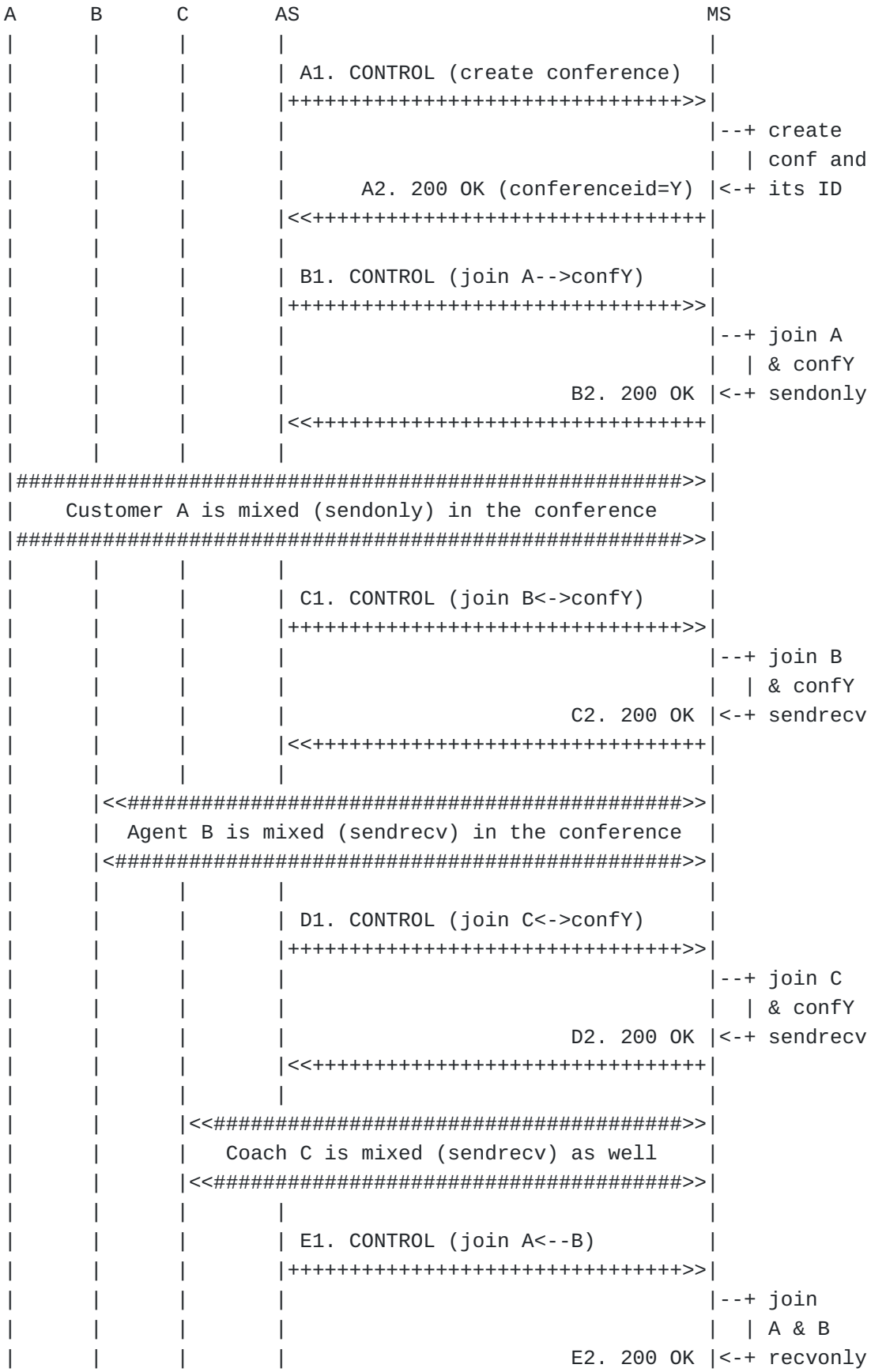
   *the customer (A) as 'sendonly';

   *the agent (B) as 'sendrecv';

   *the coach (C) as 'sendrecv' and with a -3dB gain to halve the volume of its own contribution (so that the agent actually hears the customer louder, and the coach whispering);

3. finally, the customer is joined to the agent as a passive receiver (recvonly).

A sequence diagram of such a sequence of transactions is depicted in Figure 39 (Coaching Scenario: Framework Transactions):


A
B
C
AS
MS
A1. CONTROL (create conference)
--+ create
conf and
A2. 200 OK (conferenceid=Y)
<-+ its ID
B1. CONTROL (join A-->confY)
--+ join A
& confY
B2. 200 OK
<-+ sendonly
Customer A is mixed (sendonly) in the conference
C1. CONTROL (join B<->confY)
--+ join B
& confY
C2. 200 OK
<-+ sendrecv
Agent B is mixed (sendrecv) in the conference
D1. CONTROL (join C<->confY)
--+ join C
& confY
D2. 200 OK
<-+ sendrecv
Coach C is mixed (sendrecv) as well
E1. CONTROL (join A<--B)
--+ join
A & B
E2. 200 OK
<-+ recvonly

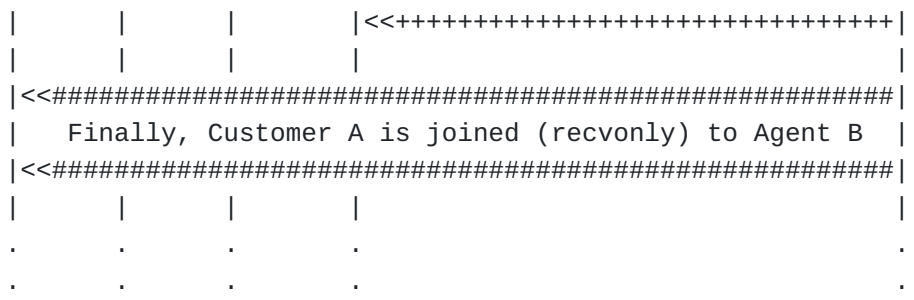


Figure 39: Coaching Scenario: Framework Transactions

TBD. Describe the framework transaction steps.

A1. AS -> MS (CFW CONTROL, createconference)

```
   CFW 238e1f2946e8 CONTROL
   Control-Package: msc-mixer
   Content-Type: application/msc-mixer+xml
   Content-Length: 293

   <mscmixer version="1.0">
      <createconference reserved-talkers="3" reserved-listeners="2">
         <audio-mixing mix-type="nbest"/>
         <video-switch type="controller"/>
         <video-layouts>
            <video-layout min-participants='1'>dual-view</video-layout>
         </video-layouts>
      </createconference>
   </mscmixer>
```

A2. AS <- MS (CFW 200 OK)

```
   CFW 238e1f2946e8 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 151

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Conference created" \
                conferenceid="1df080e"/>
   </mscmixer>
```

B1. AS -> MS (CFW CONTROL, join)

```
   CFW 2eb141f241b7 CONTROL
   Control-Package: msc-mixer
   Content-Type: application/msc-mixer+xml
   Content-Length: 186

   <mscmixer version="1.0">
      <join id1="10514b7f~6a900179" id2="1df080e">
         <stream media="audio" direction="sendonly"/>
         <stream media="video" direction="sendonly"/>
      </join>
   </mscmixer>
```

B2. AS <- MS (CFW 200 OK)

```
   CFW 2eb141f241b7 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
   </mscmixer>
```

C1. AS -> MS (CFW CONTROL, join)

```
   CFW 515f007c5bd0 CONTROL
   Control-Package: msc-mixer
   Content-Type: application/msc-mixer+xml
   Content-Length: 85

   <mscmixer version="1.0">
      <join id1="756471213~c52ebf1b" id2="1df080e"/>
   </mscmixer>
```

C2. AS <- MS (CFW 200 OK)

```
   CFW 515f007c5bd0 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
   </mscmixer>
```

D1. AS -> MS (CFW CONTROL, join)

```
   CFW 0216231b1f16 CONTROL
   Control-Package: msc-mixer
   Content-Type: application/msc-mixer+xml
   Content-Length: 182

   <mscmixer version="1.0">
      <join id1="z9hG4bK19461552~1353807a" id2="1df080e">
         <stream media="audio">
            <volume controltype="setgain" value="-3"/>
         </stream>
      </join>
   </mscmixer>
```

```
D2. AS <- MS (CFW 200 OK)
-------------------------
   CFW 0216231b1f16 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
   </mscmixer>


E1. AS -> MS (CFW CONTROL, join)
--------------------------------
   CFW 140e0f763352 CONTROL
   Control-Package: msc-mixer
   Content-Type: application/msc-mixer+xml
   Content-Length: 197

   <mscmixer version="1.0">
      <join id1="10514b7f~6a900179" id2="756471213~c52ebf1b">
         <stream media="audio" direction="recvonly"/>
         <stream media="video" direction="recvonly"/>
      </join>
   </mscmixer>


E2. AS <- MS (CFW 200 OK)
-------------------------
   CFW 140e0f763352 200
   Timeout: 10
   Content-Type: application/msc-mixer+xml
   Content-Length: 125

   <mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
      <response status="200" reason="Join successful"/>
   </mscmixer>
```

6.4.5. Sidebars

TOC

TBD. (Even more issues than in coaching scenario; of greater interest for conferencing, expecially XCON; as before, focus on per-user and per-conference settings; potential issues and how to deal with them; etc...).

(Figure not available yet.)

Figure 40: Sidebars: Media Perspective

(Figure not available yet.)

Figure 41: Sidebars: UAC Legs not attached

(Figure not available yet.)

Figure 42: Sidebars: UAC Legs mixed and attached

(Figure not available yet).

Figure 43: Sidebars: Framework Transactions

TOC

7. Security Considerations

TBD. (None, since this is informational? Reference the security sections from the core and packages drafts?)

---

8. Change Summary

TOC

The following are the major changes between the 02 and the 03 versions of the draft:

*updated the flows according to the latest drafts;

*updated the State Diagrams;

*recaptured almost all flows with the new prototype;

*captured and explained most of the missing scenarios (e.g. coaching, conferencing, voicemail, etc);

*added a new scenario (record and then replay a phone call);

*clarified that the provided 3PCC signalings are just simplified examples and not the mandatory approach to the issue;

*added new explainatory text in several parts of the document.

The following are the major changes between the 01 and the 02 versions of the draft:

*updated the flows according to the new core draft (COMEDIA, new dialogid, SYNCH->SYNC, etc.);

*updated the flows involving the updated IVR draft;

*changed the token (ESCS -> SCFW -> CFW);

*references updated (RFC5167 [RFC5167] (Dolly, M. and R. Even, “Media Server Control Protocol Requirements,” March 2008.), and IVR draft as WG item [I-D.ietf-mediactrl-ivr-control-package] (McGlashan, S., Melanchuk, T., and C. Boulton, “An Interactive Voice Response (IVR) Control Package for the Media Control Channel Framework,” February 2010.).

The following are the major changes between the 00 and the 01 versions of the draft:

*changed the title of the draft to reflect the current specification of the framework;

*added some definitions to the Terminology section;

*added State Diagrams from both the AS and MS perspective;

*added text to the Control Channel Establishment section;

*added sequence diagrams and text to the Phone Call section;

*added sequence diagrams and text to the Simple Bridging section;

*added sequence diagrams and text to the Rich Conference Scenario section;

*added documented section for Voice Mail;

*added placeholder section for BFCP-moderated Conferencing;

*references updated (RFC3264 [RFC3264] (Rosenberg, J. and H. Schulzrinne, “An Offer/Answer Model with Session Description Protocol (SDP),” June 2002.), RFC4145 [RFC4145] (Yon, D. and G. Camarillo, “TCP-Based Media Transport in the Session Description Protocol (SDP),” September 2005.) and RFC4579 [RFC4579] (Johnston, A. and O. Levin, “Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents,” August 2006.)).

---

9. Acknowledgements

TOC

TBD.

---

10. References

TOC

[RFC2234]	Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," RFC 2234, November 1997 (TXT, HTML, XML).
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC2434]	Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," BCP 26, RFC 2434, October 1998 (TXT, HTML, XML).
[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 (TXT).
[RFC3264]	Rosenberg, J. and H. Schulzrinne, "An Offer/ Answer Model with Session Description Protocol (SDP)," RFC 3264, June 2002 (TXT).
[RFC3725]	Rosenberg, J., Peterson, J., Schulzrinne, H., and G. Camarillo, "Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)," BCP 85, RFC 3725, April 2004 (TXT).
[RFC3550]	Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," STD 64, RFC 3550, July 2003 (TXT, PS, PDF).
[RFC4574]	Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute," RFC 4574, August 2006 (TXT).
[RFC4145]	Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)," RFC 4145, September 2005 (TXT).
[RFC4579]	Johnston, A. and O. Levin, "Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents," BCP 119, RFC 4579, August 2006 (TXT).
[RFC5167]	Dolly, M. and R. Even, "Media Server Control Protocol Requirements," RFC 5167, March 2008 (TXT).
[I-D.ietf-mediactrl-architecture]	Melanchuk, T., "An Architectural Framework for Media Server Control," draft-ietf-mediactrl-architecture-04 (work in progress), November 2008 (TXT).
[I-D.ietf-mediactrl-sip-control-framework]	Boulton, C., Melanchuk, T., and S. McGlashan, "Media Control Channel Framework," draft-ietf-

	mediactrl-sip-control-framework-11 (work in progress), October 2009 (TXT).
[I-D.boulton-mmusic-sdp-control-package-attribute]	Boulton, C., "A Session Description Protocol (SDP) Control Package Attribute," draft-boulton-mmusic-sdp-control-package-attribute-04 (work in progress), March 2009 (TXT).
[I-D.ietf-mediactrl-ivr-control-package]	McGlashan, S., Melanchuk, T., and C. Boulton, "An Interactive Voice Response (IVR) Control Package for the Media Control Channel Framework," draft-ietf-mediactrl-ivr-control-package-08 (work in progress), February 2010 (TXT).
[I-D.ietf-mediactrl-mixer-control-package]	McGlashan, S., Melanchuk, T., and C. Boulton, "A Mixer Control Package for the Media Control Channel Framework," draft-ietf-mediactrl-mixer-control-package-11 (work in progress), February 2010 (TXT).
[I-D.boulton-ivr-vxml-control-package]	Boulton, C., Melanchuk, T., and S. McGlashan, "A VoiceXML Control Package for the Media Control Channel Framework," draft-boulton-ivr-vxml-control-package-04 (work in progress), February 2008 (TXT).
[I-D.miniero-bfcp-control-package]	Miniero, L., Romano, S., Even, R., and S. McGlashan, "A Binary Floor Control Protocol (BFCP) Control Package for the Media Control Channel Framework," draft-miniero-bfcp-control-package-01 (work in progress), July 2008 (TXT).

Authors' Addresses

TOC

	Alessandro Amirante
	University of Napoli
	Via Claudio 21
	Napoli 80125
	Italy
Email:	alessandro.amirante@unina.it
	Tobia Castaldi
	University of Napoli
	Via Claudio 21
	Napoli 80125
	Italy
Email:	tobia.castaldi@unina.it
	Lorenzo Miniero
	University of Napoli

	Via Claudio 21
	Napoli 80125
	Italy
Email:	lorenzo.miniero@unina.it
	Simon Pietro Romano
	University of Napoli
	Via Claudio 21
	Napoli 80125
	Italy
Email:	spromano@unina.it

Full Copyright Statement

TOC

Intellectual Property