

MEDIACTRL
Internet-Draft
Intended status: Informational
Expires: November 24, 2013

A. Amirante
University of Napoli
T. Castaldi
L. Miniero
Meetecho
S P. Romano
University of Napoli
May 23, 2013

**Media Control Channel Framework (CFW) Call Flow Examples
draft-ietf-mediactrl-call-flows-13**

Abstract

This document provides a list of typical Media Control Channel Framework call flows. It aims at being a simple guide to the use of the interface between Application Servers and MEDIACTRL-based Media Servers, as well as a base reference documentation for both implementors and protocol researchers.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 24, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	4
2.	Conventions	5
3.	Terminology	5
4.	A Practical Approach	6
4.1.	State Diagrams	6
5.	Control Channel Establishment	10
5.1.	COMEDIA Negotiation	11
5.2.	SYNC	14
5.3.	K-ALIVE	15
5.4.	Wrong behaviour	17
6.	Use-case scenarios and examples	20
6.1.	Echo Test	27
6.1.1.	Direct Echo Test	28
6.1.2.	Echo Test based on Recording	30
6.2.	Phone Call	38
6.2.1.	Direct Connection	41
6.2.2.	Conference-based Approach	43
6.2.3.	Recording a conversation	49
6.3.	Conferencing	55
6.3.1.	Simple Bridging	60
6.3.2.	Rich Conference Scenario	64
6.3.3.	Coaching Scenario	73
6.3.4.	Sidebars	80
6.3.5.	Floor Control	89
6.4.	Additional Scenarios	95
6.4.1.	Voice Mail	95
6.4.2.	Current Time	102
6.4.3.	DTMF-driven Conference Manipulation	106
7.	Media Resource Brokering	119
7.1.	Publishing Interface	119
7.2.	Consumer Interface	127
7.2.1.	Query Mode	128
7.2.2.	Inline-aware Mode	132
7.2.3.	Inline-unaware Mode	146
7.3.	Handling media dialogs	148
7.3.1.	Query and Inline-aware mode	148
7.3.2.	Inline-unaware mode	151
7.3.3.	CFW Protocol Behaviour	158
8.	Security Considerations	161
9.	IANA Considerations	170

10.	Change Summary	170
11.	Acknowledgements	173
12.	References	173
12.1.	Normative References	173
12.2.	Informative References	174
Authors'	Addresses	175

1. Introduction

This document provides a list of typical MEDIACTRL Media Control Channel Framework [[RFC6230](#)] call flows. The motivation for this comes from our implementation experience with the framework and its protocol. This drove us to writing a simple guide to the use of the several interfaces between Application Servers and MEDIACTRL-based Media Servers and a 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). In [Section 7](#) some flows involving more entities by means of a Media Resource Broker compliant with [[RFC6917](#)] are presented. To ease 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, 'as.example.com' and 'ms.example.net', following [[RFC2606](#)]. Besides, the flows will often focus more on the CFW [[RFC6230](#)] interaction rather than on the other involved protocols, e.g., SIP [[RFC3261](#)], SDP [[RFC3264](#)] or RTP [[RFC3550](#)].

In the next paragraphs a brief overview of our implementation approach is described, with particular focus on 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 the understanding of how the framework works from a practical point of view.

Once done with these 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 use case scenarios, involving the most typical multimedia applications, are depicted and described.

It is worth pointing out that this document is not meant in any way to be a self-contained guide to implementing a MEDIACTRL-compliant framework. The specifications are a mandatory read for all implementors, especially considering that this document by itself follows their guidelines but does not delve into the details of every aspect of the protocol.

2. Conventions

Note that due to RFC formatting conventions, SIP/SDP and CFW lines whose content exceeds 72 characters are split across lines. This line folding is marked by a backslash at the end of the first line. This backslash, the preceding whitespace, the following CRLF, and the whitespace beginning the next line would not appear in the actual protocol contents. Besides, also note that the indentation of the XML content is only provided for readability: actual messages will follow strict XML syntax, which allows for, but does not require, indentation. Due to the same 72 characters limitation, this document also sometimes splits the content of XML elements across lines: please beware that, when this happens, no whitespace is actually meant to be neither at the beginning nor at the end of the element content.

Besides, also note that a few diagrams present arrows that go from a network entity to itself. It's worth pointing out that such arrows do not represent any transaction message, but are rather meant as an indication to the reader that the involved network entity took a decision, within its application logic, according to the input it previously received.

3. Terminology

This document makes use of the same terminology as the referenced documents [[RFC6230](#)] [[RFC6231](#)] [[RFC6505](#)] [[RFC6917](#)]. 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:

COMEDIA: connection-oriented media (i.e., TCP and TLS). Also used to signify the support in SDP for connection-oriented media, and the RFCs that define that support ([[RFC4145](#)] and [[RFC4572](#)]).

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 functions 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.

VCR controls: runtime control of aspects of an audio playback like speed and volume, via DTMF signals sent by the user, in a manner that resembles the functions of a VCR (video cassette recorder) controller.

4. A Practical Approach

In this document we embrace an engineering approach to the description of a number of interesting scenarios that can be realized through the careful orchestration of the Media Control Channel Framework entities, namely the Application Server and the Media Server. We will demonstrate, through detailed call flows, how a variegated bouquet of services (ranging from very simple scenarios to much more complicated ones) can be implemented with the functionality currently offered, within the main MEDIACTRL framework, by the control packages that have been made available to date. The document aims at representing a useful guide for those interested in investigating the inter-operation among MEDIACTRL components, as well as for application developers willing to build advanced services on top of the base infrastructure made available by the framework.

4.1. State Diagrams

In this section we present an "informal" view of the main MEDIACTRL protocol interactions, in the form of state diagrams. Each diagram is indeed a classical representation of a Mealy automaton, comprising a number of possible protocol states, indicated with rectangular boxes. Transitions between states are indicated through edges, with each edge labeled with a slash-separated pair representing a specific input together with the associated output (a dash in the output position means that, for that particular input, no output is generated from the automaton). Some of the inputs are associated with MEDIACTRL protocol messages arriving at a MEDIACTRL component while it is in a certain state: this is the case of 'CONTROL', 'REPORT' (in its various "flavors" -- pending, terminate, etc.), '200', '202', as well as 'Error' (error messages correspond to specific numeric codes). Further inputs represent triggers arriving at the MEDIACTRL automaton from the upper layer, namely the Application Programming Interface used by programmers while implementing MEDIACTRL-enabled services: such inputs have been indicated with the term 'API' followed by the message that the API itself is triggering (as an example, 'API terminate' is a request to send a 'REPORT' message with a status of 'terminate' to the peering component).

Four diagrams are provided. Two of them (Figure 1 and Figure 2) describe normal operation of the framework. Figure 3 contains two diagrams describing asynchronous event notifications. Figure 1 embraces the MS perspective, whereas Figure 2 is on the AS side. The upper part of Figure 3 shows how events are generated, on the MS side, by issuing a CONTROL message addressed to the AS; events are acknowledged by the AS through standard 200 responses. Hence, the behavior of the AS, which mirrors that of the MS, is depicted in the lower part of the figure.

Coming back to Figure 1, the diagram shows that the MS activates upon reception of CONTROL messages coming from the AS, which typically instruct it about the execution of a specific command, belonging to one of the available control packages. The execution of the received command can either be quick, or require some time. In the former case, right after completing its operation, the MS sends back to the AS a 200 message, which basically acknowledges correct termination of the invoked task. In the latter case, the MS first sends back an interlocutory 202 message, which lets it enter a different state ('202' sent). While in the new state, the MS keeps on performing the invoked task: if such task does not complete in a predefined timeout, the server will update the AS on the other side of the control channel by periodically issuing 'REPORT/update' messages; each such message has to be acknowledged by the AS (through a '200' response). Eventually, when the MS is done with the required service, it sends to the AS a 'REPORT/terminate' message, whose acknowledgment receipt concludes a transaction. It is worth pointing out that the MS may send a 202 response after it determines that the request contains no errors that cannot be reported in a later REPORT terminate request. After the MS sends a 202 response, any error that it (or the API) finds in the request is reported in the final REPORT terminate request. Again, the AS behavior, depicted in Figure 2, mirrors the above described actions undertaken at the MS side. Figures also show the cases in which transactions cannot be successfully completed due to abnormal conditions, which always trigger the creation and expedition of a specific 'Error' message which, as anticipated, is reported as a numeric error code.

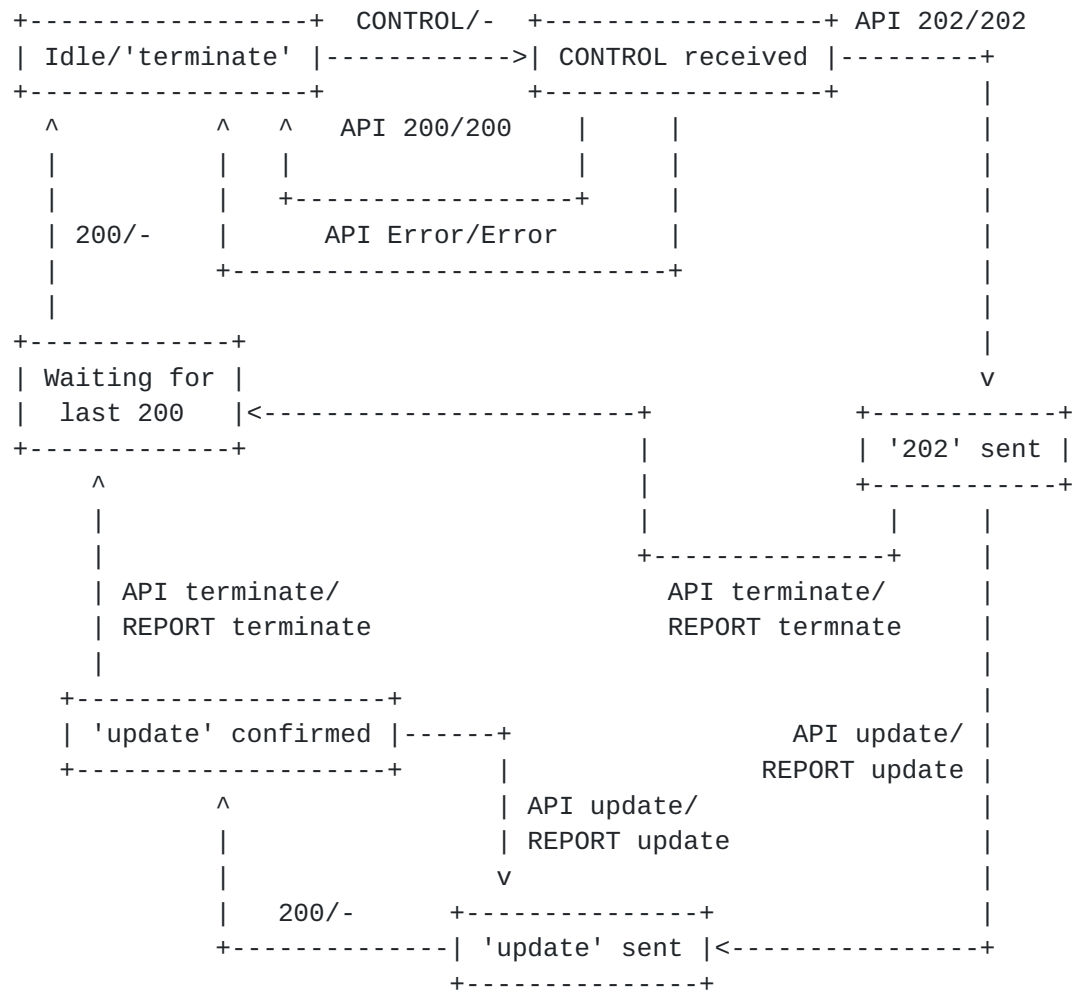


Figure 1: Media Server CFW State Diagram

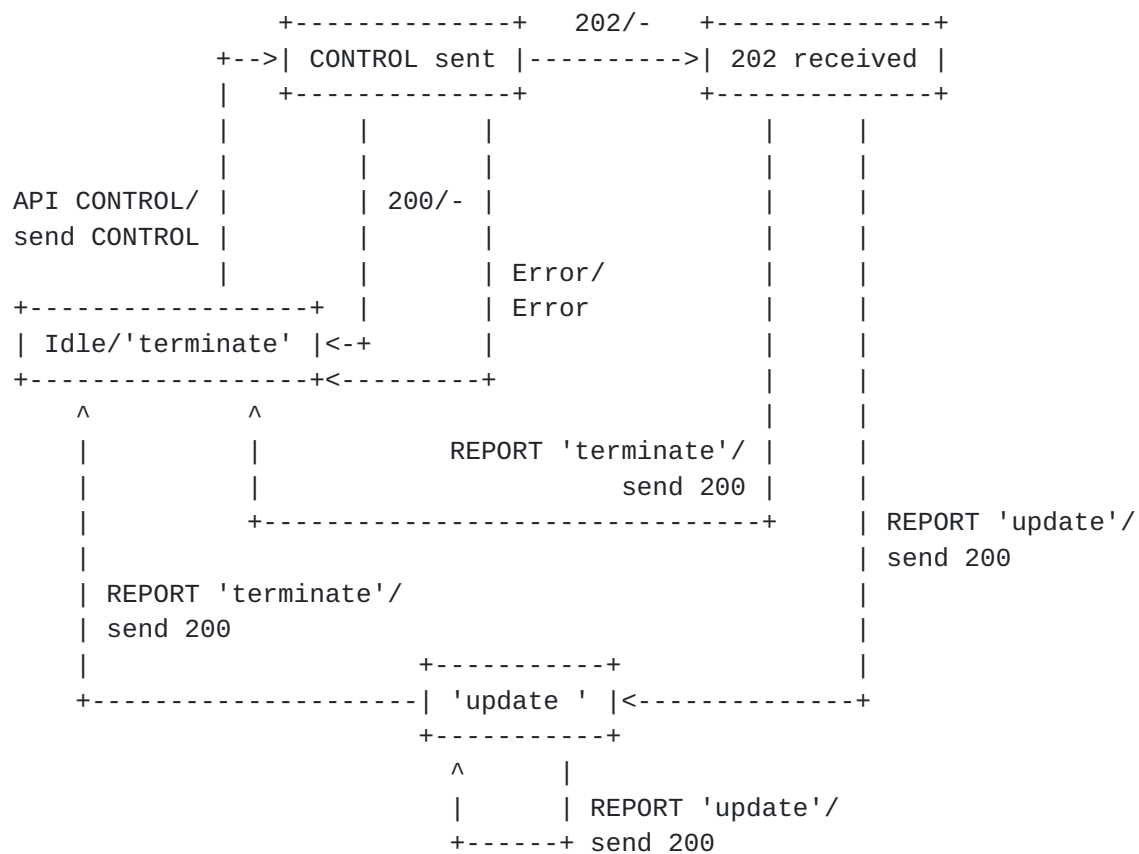
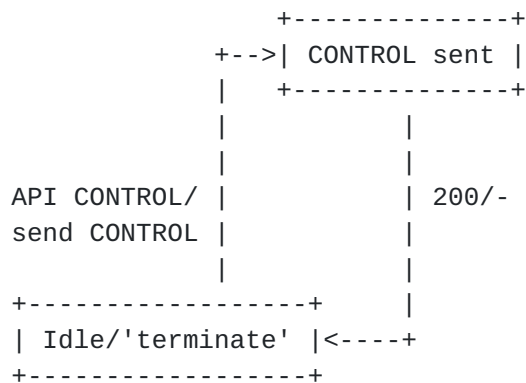
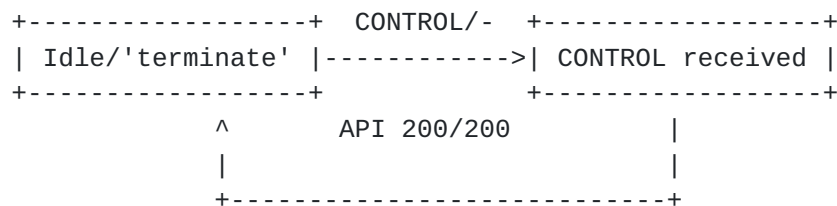


Figure 2: Application Server CFW State Diagram



(Media Server perspective)



(Application Server perspective)

Figure 3: Event Notifications

5. Control Channel Establishment

As specified in [\[RFC6230\]](#), 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 connection-oriented media (COMEDIA) [\[RFC4145\]](#) [\[RFC4572\]](#) negotiation. This negotiation allows for a reliable connection to be created between the AS and the MS: it is here that the AS and the MS agree on the transport level protocol to use (TCP/SCTP) and whether any application level security is needed or not (e.g., TLS). For the sake of simplicity, we assume that an unencrypted TCP connection is negotiated between the two involved entities. Once they have connected, a SYNC message sent by the AS to the MS consolidates the control channel. An example of how a keep-alive message is triggered is also presented in the following paragraphs. For the sake of completeness, this section also includes a couple of common mistakes that can occur when dealing with the Control Channel establishment.


```

AS                                     MS
|                                     |
| INVITE (COMEDIA)                   |
|----->|                             |
|           100 (Trying)              |
|<-----|                             |
|           200 OK (COMEDIA)          |
|<-----|                             |
| ACK                                  |
|----->|                             |
|                                     |
|=====>|                             |
| TCP CONNECT (CTRL CHANNEL)         |
|=====>|                             |
|                                     |
| SYNC (Dialog-ID, etc.)             |
|++++++>|                             |
|                                     | --+
|                                     | | Check SYNC
|                                     |<--+
|           200 OK                    |
|<<+++++|                             |
|                                     |
.                                     .
.                                     .

```

Figure 4: Control Channel Establishment

5.1. COMEDIA Negotiation

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 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 answer, 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 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.

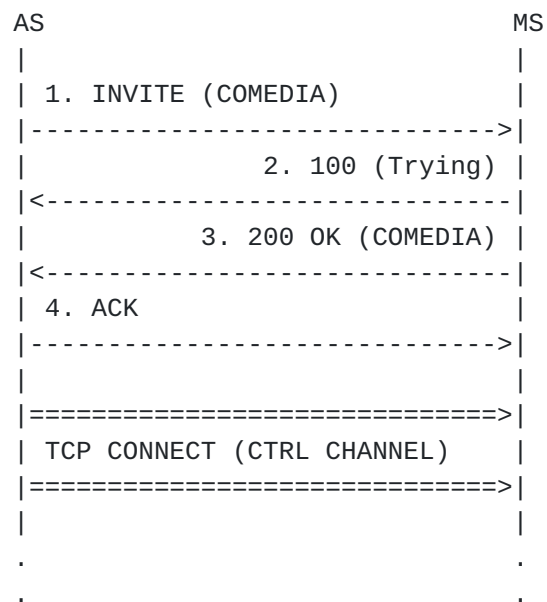


Figure 5: COMEDIA Negotiation: Sequence Diagram

1. AS -> MS (SIP INVITE)

```

-----
INVITE sip:MediaServer@ms.example.net:5060 SIP/2.0
Via: SIP/2.0/UDP 203.0.113.1:5060;\
    branch=z9hG4bK-d8754z-9b07c8201c3aa510-1---d8754z-;rport=5060
Max-Forwards: 70
Contact: <sip:ApplicationServer@203.0.113.1:5060>
To: <sip:MediaServer@ms.example.net:5060>
From: <sip:ApplicationServer@as.example.com:5060>;tag=4354ec63
Call-ID: MDk2YTk1MDU3YmVkJzJgZYTQwYmJlNjE5NTA4ZDQ1OGY.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, REGISTER
Content-Type: application/sdp
Content-Length: 203

```

```

v=0
o=lminiero 2890844526 2890842807 IN IP4 as.example.com
s=MediaCtrl
c=IN IP4 as.example.com
t=0 0
m=application 5757 TCP cfw
a=connection:new
a=setup:active
a=cfw-id:5feb6486792a

```


2. AS <- MS (SIP 100 Trying)

SIP/2.0 100 Trying
Via: SIP/2.0/UDP 203.0.113.1:5060; \
branch=z9hG4bK-d8754z-9b07c8201c3aa510-1---d8754z-;rport=5060
To: <sip:MediaServer@ms.example.net:5060>;tag=499a5b74
From: <sip:ApplicationServer@as.example.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 203.0.113.1:5060; \
branch=z9hG4bK-d8754z-9b07c8201c3aa510-1---d8754z-;rport=5060
Contact: <sip:MediaServer@ms.example.net:5060>
To: <sip:MediaServer@ms.example.net:5060>;tag=499a5b74
From: <sip:ApplicationServer@as.example.com:5060>;tag=4354ec63
Call-ID: MDk2YTk1MDU3YmVkZjgzYTQwYmJlNjE5NTA4ZDQ1OGY.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, REGISTER
Content-Type: application/sdp
Content-Length: 199

v=0
o=lminiero 2890844526 2890842808 IN IP4 ms.example.net
s=MediaCtrl
c=IN IP4 ms.example.net
t=0 0
m=application 7575 TCP cfw
a=connection:new
a=setup:passive
a=cfw-id:5feb6486792a

4. AS -> MS (SIP ACK)

ACK sip:MediaServer@ms.example.net:5060 SIP/2.0
Via: SIP/2.0/UDP 203.0.113.1:5060; \
branch=z9hG4bK-d8754z-22940f5f4589701b-1---d8754z-;rport
Max-Forwards: 70
Contact: <sip:ApplicationServer@203.0.113.1:5060>
To: <sip:MediaServer@ms.example.net:5060>;tag=499a5b74
From: <sip:ApplicationServer@as.example.com:5060>;tag=4354ec63
Call-ID: MDk2YTk1MDU3YmVkZjgzYTQwYmJlNjE5NTA4ZDQ1OGY.
CSeq: 1 ACK

Content-Length: 0

5.2. SYNC

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 entities (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 paragraph, is meant to be globally unique and as such is properly 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 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 in, 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 6 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 Mixer package. 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 just a dummy package providing testing functionality).

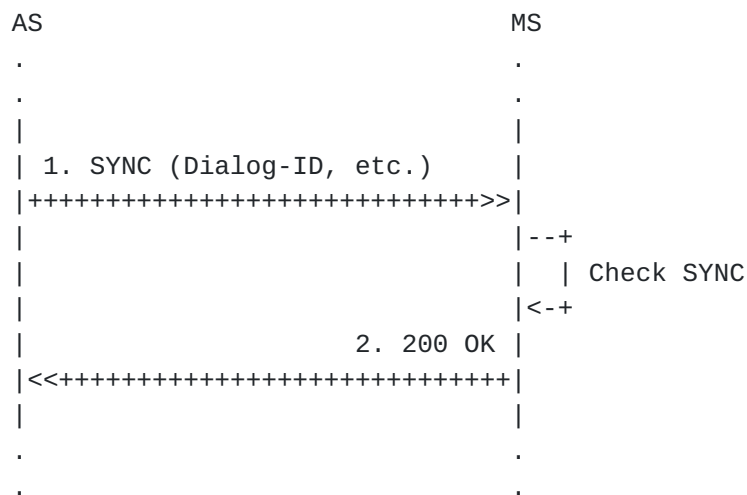


Figure 6: SYNC: Sequence Diagram

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-example-pkg/1.0

The framework level transaction identifier is obviously the same in both the request and the response (6e5e86f95609), since the AS needs to be able to match the response to the original request. At this point, the control channel is finally established, and it can be used by the AS to request services from the MS.

5.3. K-ALIVE

The Control Framework provides a mechanism for implementing a keep-alive functionality. Such a mechanism is especially useful whenever any NAT or firewall sits in the path between an AS and a MS. In fact, NATs and firewalls may have timeout values for the TCP

connections they handle, which means that, if no traffic is detected on these connections within a specific time, they could be shut down. This could be the case of a Control Channel established between an AS and a MS but not used for some time. For this reason, the Control Framework specifies a dedicated framework message (K-ALIVE) that the AS and MS can make use of in order to generate traffic on the TCP connection and keep it alive.

In the previous section it has been described that the timeout value for the keep-alive mechanism is set by the SYNC request. Specifically, in the example the AS specified a value of 100 seconds. In fact, the timeout value is not actually negotiated between the AS and MS, as it is simply specified by whichever endpoint takes the active role. The 100 seconds value is compliant with how NATs and firewalls are usually implemented, since in most cases the timeout value they use before shutting TCP connections down is around 2 minutes. Such value has a strong meaning within the context of this mechanism. In fact, it means that the active role (in this case the AS) has to send a K-ALIVE message before those 100 seconds pass, otherwise the passive role (the MS) will tear down the connection treating it like a timeout. The Control Framework document suggests a more conservative approach towards handling this timeout value, suggesting to trigger the K-ALIVE message before 80% of the negotiated time passes (in this case, 80 seconds). This is exactly the case presented in Figure 7.

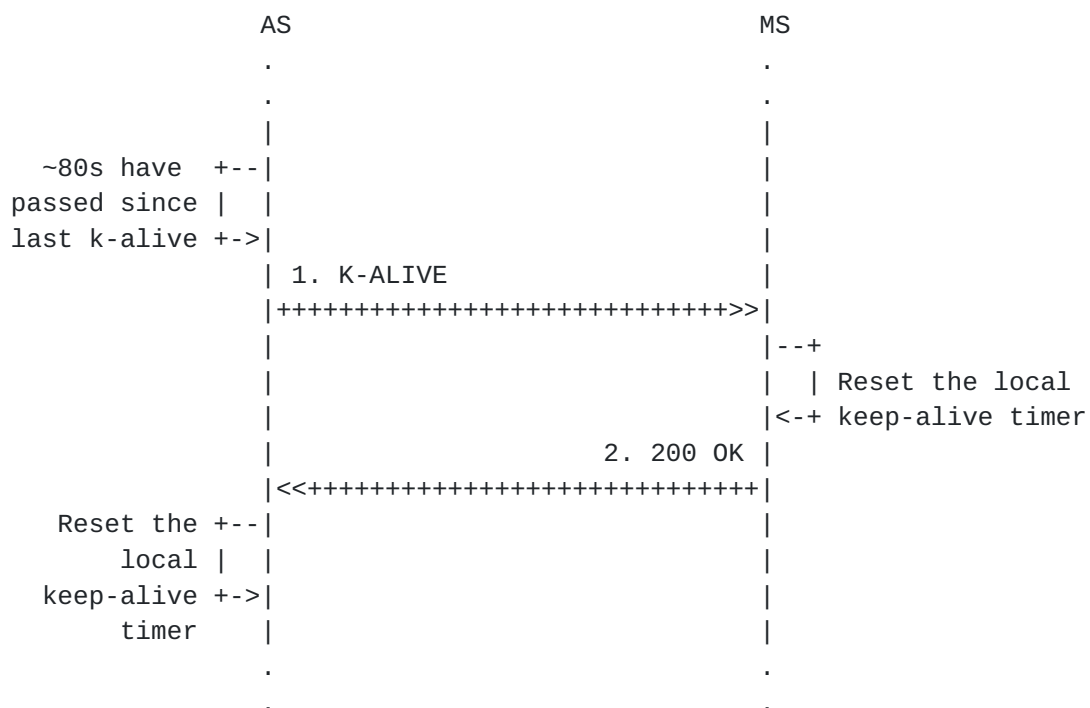


Figure 7: K-ALIVE: Sequence Diagram

After the Control Channel has been established (COMEDIA+SYNC) both the AS and the MS start local keep-alive timers mapped to the negotiated keep-alive timeout value (100 seconds). When about 80 seconds have passed since the start of the timer (80% of 100 seconds), the AS sends the MS a framework level K-ALIVE message. As it can be seen in the protocol message dump, the message is very lightweight, since it only includes a single line with no additional header. When the MS receives the K-ALIVE message, it resets its local keep-alive timer and sends a 200 message back as confirmation. As soon as the AS receives the 200 message, it resets its local keep-alive timer as well and the mechanism starts over again.

The actual transaction steps are presented in the next figure.

1. AS -> MS (K-ALIVE)

CFW 518ba6047880 K-ALIVE

2. AS <- MS (CFW 200)

CFW 518ba6047880 200

In case the timer expired either in the AS or in the MS (i.e., the K-ALIVE or the 200 arrived after the 100 seconds) the connection and the associated SIP Control Dialog would be torn down by the entity detecting the timeout, thus ending the interaction between the AS and the MS.

5.4. Wrong behaviour

This section will briefly address some of those which could represent the most common mistakes when dealing with the establishment of a Control Channel between an AS and a MS. These scenarios are obviously of interest, since they result in the AS and the MS being unable to interact with each other. Specifically, these simple scenarios will be described:

1. an AS providing the MS with a wrong Dialog-ID in the initial SYNC;
2. an AS sending a generic CONTROL message instead of SYNC as a first transaction.

The first scenario is depicted in Figure 8.

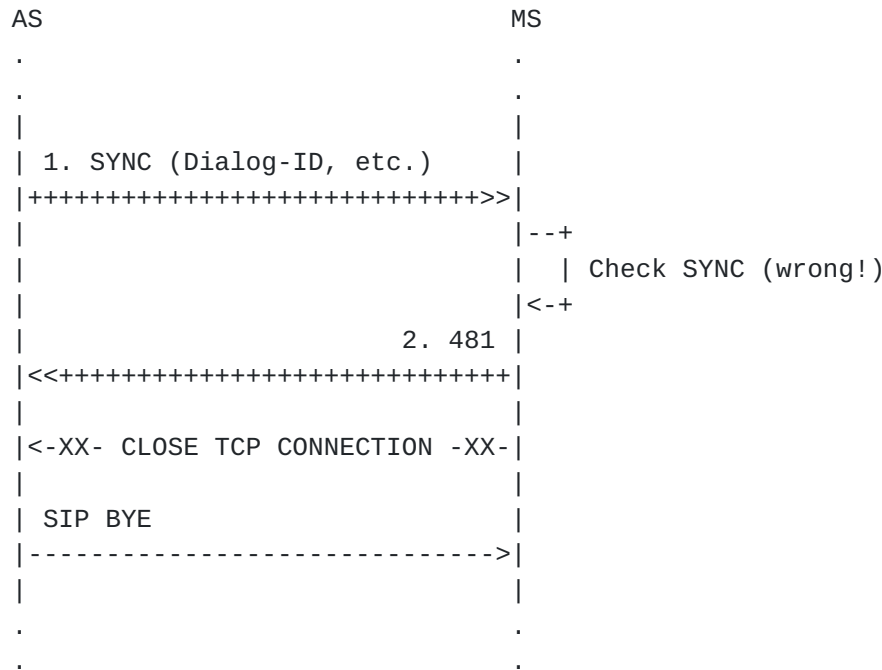


Figure 8: SYNC with wrong Dialog-ID: Sequence Diagram

The scenario is similar to the one presented in [Section 5.2](#) but with a difference: instead of using the correct, expected, Dialog-ID in the SYNC message (5feb6486792a, the one negotiated via COMEDIA), the AS uses a wrong value (4hrn7490012c). This causes the SYNC transaction to fail. First of all, the MS sends a framework level 481 message. This response, when given in reply to a SYNC message, means that the SIP dialog associated with the provided Dialog-ID (the wrong identifier) does not exist. Besides, the Control Channel must be torn down as a consequence, and so the MS also closes the TCP connection it received the SYNC message from. The AS at this point is supposed to tear down its SIP Control Dialog as well, and so sends a SIP BYE to the MS.

The actual transaction is presented in the next picture.

1. AS -> MS (CFW SYNC with wrong Dialog-ID)

```

-----
CFW 2b4dd8724f27 SYNC
Dialog-ID: 4hrn7490012c
Keep-Alive: 100
Packages: msc-ivr/1.0,msc-mixer/1.0

```

2. AS <- MS (CFW 481)

```

-----
CFW 2b4dd8724f27 481

```

The second scenario instead is depicted in Figure 9.

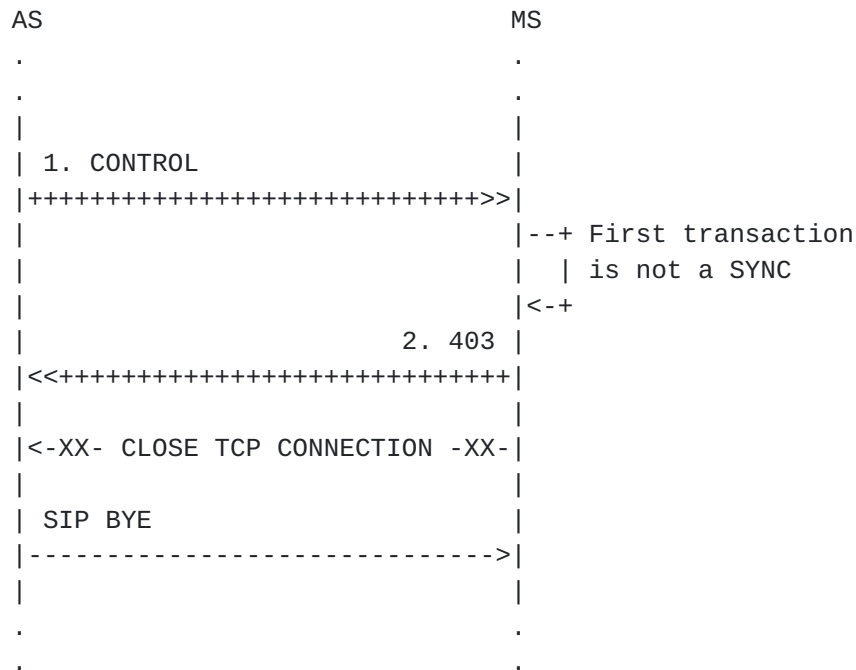


Figure 9: Incorrect first transaction: Sequence Diagram

This scenario is another common mistake that could occur when trying to set up a Control Channel. In fact, the Control Framework mandates that the first transaction after the COMEDIA negotiation be a SYNC to conclude the setup. In case the AS, instead of triggering a SYNC message as expected, sends a different message to the MS (in the example, it tries to send an <audit> message addressed to the IVR Control Package), the MS treats it like an error. As a consequence, the MS replies with a framework level 403 message (Forbidden) and,

just as before, closes the TCP connection and waits for the related SIP Control Dialog to be torn down.

The actual transaction is presented in the next picture.

1. AS -> MS (CFW CONTROL instead of SYNC)

```
-----  
CFW 101fbbd62c35 CONTROL  
Control-Package: msc-ivr/1.0  
Content-Type: application/msc-ivr+xml  
Content-Length: 78  
  
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">  
  <audit/>  
</mscivr>
```

2. AS <- MS (CFW 403 Forbidden)

```
-----  
CFW 101fbbd62c35 403
```

6. Use-case scenarios and examples

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 examples that will be presented in this document. The UAC session is assumed to have been created as the described Figure 10:

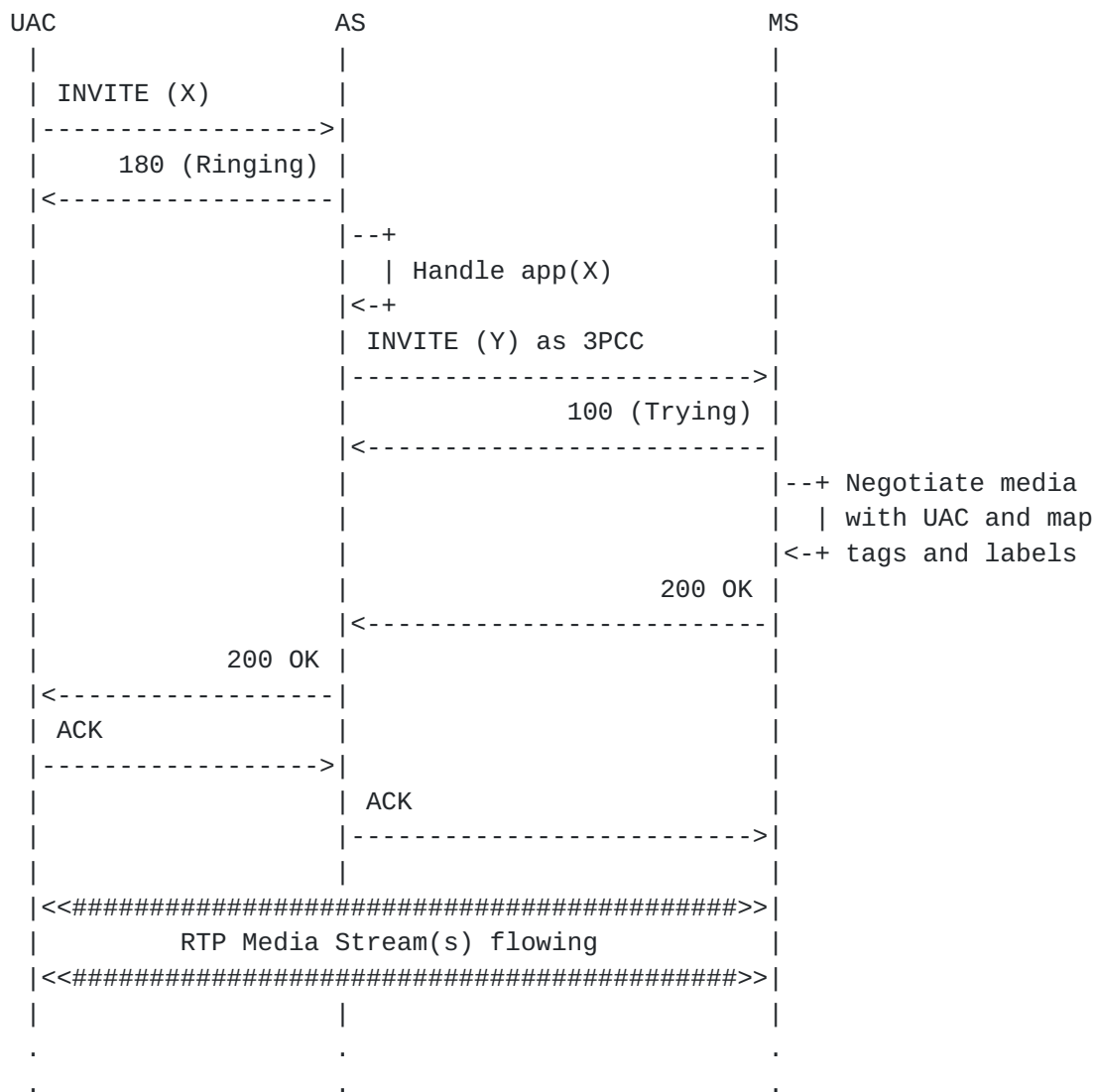


Figure 10: 3PCC Sequence Diagram

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] provides several different solutions and many details about how 3PCC can be realized, with pros and cons. Besides, it is also worth pointing out that the two INVITEs displayed in the figure are different SIP dialogs.

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:mediactrlDemo@as.example.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 [RFC5567]: the INVITE is forwarded by the AS to the MS in a 3PCC fashion, without the SDP provided by the UAC being touched, so 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 in the AS, all the media would start flowing directly 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 [RFC6230]. How the identifiers are created and addressed is explained by making use of the sample signaling provided in the following lines.

1. UAC -> AS (SIP INVITE)

```
-----
INVITE sip:mediactrlDemo@as.example.com SIP/2.0
Via: SIP/2.0/UDP 203.0.113.2:5063;rport;branch=z9hG4bK1396873708
From: <sip:lminiero@users.example.com>;tag=1153573888
To: <sip:mediactrlDemo@as.example.com>
Call-ID: 1355333098
CSeq: 20 INVITE
Contact: <sip:lminiero@203.0.113.2:5063>
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 203.0.113.2
s=A conversation
c=IN IP4 203.0.113.2
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 203.0.113.2:5063;rport=5063; \
                                     branch=z9hG4bK1396873708

Contact: <sip:mediactrlDemo@as.example.com>
To: <sip:mediactrlDemo@as.example.com>;tag=bcd47c32
From: <sip:lminiero@users.example.com>;tag=1153573888
Call-ID: 1355333098
CSeq: 20 INVITE
Content-Length: 0
```

3. AS -> MS (SIP INVITE)

```
-----
INVITE sip:MediaServer@ms.example.net:5060;transport=UDP SIP/2.0
Via: SIP/2.0/UDP 203.0.113.1:5060; \
      branch=z9hG4bK-d8754z-8723e421ebc45f6b-1---d8754z-;rport
Max-Forwards: 70
Contact: <sip:ApplicationServer@203.0.113.1:5060>
To: <sip:MediaServer@ms.example.net:5060>
From: <sip:ApplicationServer@as.example.com:5060>;tag=10514b7f
Call-ID: NzI0ZjQ0ZTB1MTEzMGU1ZjVhMjk5NTliMmJmZjE0NDQ.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, REGISTER
Content-Type: application/sdp
Content-Length: 330
```

```
v=0
o=lminiero 123456 654321 IN IP4 203.0.113.2
s=A conversation
c=IN IP4 203.0.113.2
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 203.0.113.1:5060; \
branch=z9hG4bK-d8754z-8723e421ebc45f6b-1---d8754z-;rport=5060
To: <sip:MediaServer@ms.example.net:5060>;tag=6a900179
From: <sip:ApplicationServer@as.example.com:5060>;tag=10514b7f
Call-ID: NzI0ZjQ0ZTB1MTEzMGU1ZjVhMjk5NTliMmJmZjE0NDQ.
CSeq: 1 INVITE
Content-Length: 0

5. AS <- MS (SIP 200 OK)

SIP/2.0 200 OK
Via: SIP/2.0/UDP 203.0.113.1:5060; \
branch=z9hG4bK-d8754z-8723e421ebc45f6b-1---d8754z-;rport=5060
Contact: <sip:MediaServer@ms.example.net:5060>
To: <sip:MediaServer@ms.example.net:5060>;tag=6a900179
From: <sip:ApplicationServer@as.example.com:5060>;tag=10514b7f
Call-ID: NzI0ZjQ0ZTB1MTEzMGU1ZjVhMjk5NTliMmJmZjE0NDQ.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, REGISTER
Content-Type: application/sdp
Content-Length: 374

v=0
o=lminiero 123456 654322 IN IP4 ms.example.net
s=MediaCtrl
c=IN IP4 ms.example.net
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 203.0.113.2:5063;rport=5063; \
                                     branch=z9hG4bK1396873708
Contact: <sip:mediactrlDemo@as.example.com>
To: <sip:mediactrlDemo@as.example.com>;tag=bcd47c32
From: <sip:lminiero@users.example.com>;tag=1153573888
Call-ID: 1355333098
CSeq: 20 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE, REGISTER
Content-Type: application/sdp
Content-Length: 374

v=0
o=lminiero 123456 654322 IN IP4 ms.example.net
s=MediaCtrl
c=IN IP4 ms.example.net
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@as.example.com SIP/2.0
Via: SIP/2.0/UDP 203.0.113.2:5063;rport;branch=z9hG4bK1113338059
From: <sip:lminiero@users.example.com>;tag=1153573888
To: <sip:mediactrlDemo@as.example.com>;tag=bcd47c32
Call-ID: 1355333098
CSeq: 20 ACK
Contact: <sip:lminiero@203.0.113.2:5063>
Max-Forwards: 70
User-Agent: Linphone/2.1.1 (eXosip2/3.0.3)
Content-Length: 0
```

8. AS -> MS (SIP ACK)

```
-----
ACK sip:MediaServer@ms.example.net:5060;transport=UDP SIP/2.0
Via: SIP/2.0/UDP 203.0.113.1:5060; \
```



```

        branch=z9hG4bK-d8754z-5246003419ccd662-1---d8754z-;rport
Max-Forwards: 70
Contact: <sip:ApplicationServer@203.0.113.1:5060>
To: <sip:MediaServer@ms.example.net:5060;tag=6a900179
From: <sip:ApplicationServer@as.example.com:5060>;tag=10514b7f
Call-ID: NzIOZjQ0ZTB1MTEzMGU1ZjVhMjk5NTliMmJmZjE0NDQ.
CSeq: 1 ACK
Content-Length: 0

```

As a result of the 3PCC negotiation just presented, the following relevant information is retrieved:

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

```

From: <sip:ApplicationServer@as.example.com:5060>;tag=10514b7f
                                             ^^^^^^^^^^
To: <sip:MediaServer@ms.example.net:5060>;tag=6a900179
                                             ^^^^^^^^^^

```

2. the labels [[RFC4574](#)] associated with the negotiated media connections, in this case an audio stream (7eda834) and a video stream (0132ca2).

```

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

```

These four 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 through a colon (':') token, addresses all the media connections between the MS and the UAC;
2. 10514b7f:6a900179 <-> 7eda834, the association of the previous value with the label attribute, addresses only one of the media connections of the UAC session (in this case, the audio stream);

since, as it will be clearer in the scenario examples, the explicit identifiers in requests can only address 'from:tag' connections, additional mechanism will be required to have a finer control upon individual media streams (i.e., by means of the <stream> element in package level requests).

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.

6.1. Echo Test

The echo test is the simplest 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 11.

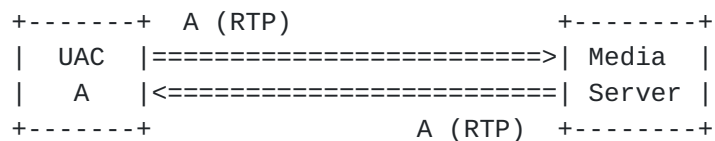


Figure 11: Echo Test: Media Perspective

From the framework point of view, when the UAC's leg is not attached to anything yet, what appears is described in Figure 12: 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 12: 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 paragraphs:

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

In the Direct Echo Test approach, the UAC is directly connected to itself. This means that, as depicted in Figure 13, each frame the MS receives from the UAC is sent back to it in real-time.

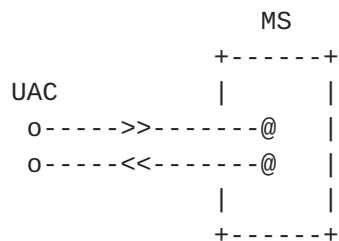


Figure 13: Echo Test: Direct Echo (self connection)

In the framework this can be achieved by means of the mixer control package [[RFC6505](#)], which is in charge of joining connections and conferences.

A sequence diagram of a potential transaction is depicted in Figure 14:

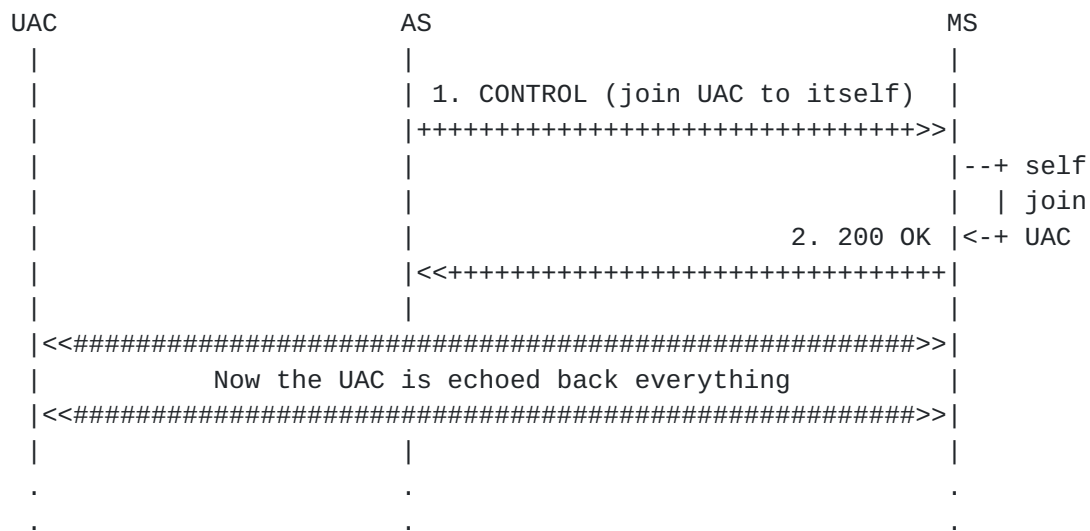


Figure 14: Self Connection: Framework Transaction

All the transaction steps have been numbered to ease the understanding. A reference to the above numbered messages is in fact made in the following explanation lines:

- o 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, both id1 and id2 attributes are set to the same value, i.e., the connectionid;
- o 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 AS, 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: 130

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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

In the Recording-based Echo Test approach, instead, the UAC is NOT directly connected to itself, but rather indirectly. This means that, as depicted in Figure 15, 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.

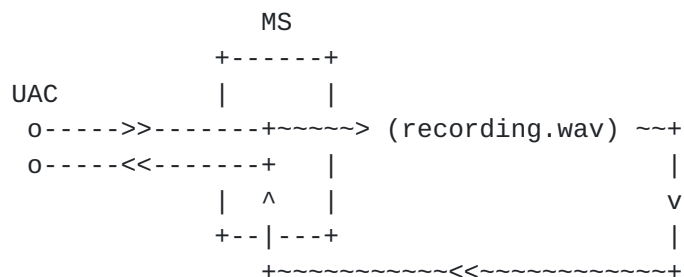


Figure 15: Echo Test: Recording involved

In the framework this can be achieved by means of the IVR control package [[RFC6231](#)], which is in charge of both the recording and the

playout phases. However, the whole scenario cannot be accomplished in a single transaction; at least two steps, in fact, need to be performed:

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 16:

[illegible]

Figure 16: Recording-based Echo: Two Framework Transactions

The first, obvious difference that comes out when looking at the diagram is that, unlike what happened in the Direct Echo example, the MS does not reply with a 200 message to the CONTROL request originated by the AS. Instead, a 202 provisional message is sent first, followed by a REPORT message. The 202+REPORT(s) mechanism is used whenever the MS wants to tell the AS that the requested operation might take more time than the limit specified in the definition of the Control Package. So, while the <join> operation in the Direct Echo scenario was expected to be fulfilled in a very short time, the IVR request was assumed to last longer instead. A 202 message provides a timeout value and tells the AS to wait a bit, since the preparation of the dialog might not be an immediate task. In this example, the preparation ends before the timeout, and so the transaction is concluded with a 'REPORT terminate', which reports the end of the transaction (as did the 200 message in the previous example). In case the preparation took longer than the timeout, an additional 'REPORT update' would have been sent with a new timeout value, and so on until completion by means of a 'REPORT terminate'.

Notice that the REPORT mechanism depicted is only shown to clarify its behaviour. In fact, the 202+REPORT mechanism is assumed to be involved only when the requested transaction is expected to take a long time (e.g., retrieving a large media file for a prompt from an external server). In this scenario the transaction would be prepared in much less time, and as a consequence would very likely be completed within the context of a simple CONTROL+200 request/response. The following scenarios will only involve 202+REPORTs when they are strictly necessary.

About the dialog itself, notice how the AS-originated CONTROL transactions are terminated as soon as the requested dialogs start: as specified in [[RFC6231](#)], 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 the understanding and the following of the subsequent explanation lines. Besides, the two transactions are distinguished by the preceding letter (A,B=recording, C,D=playout).

- o 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 (considering the associated connection includes both audio and video and no restriction is enforced on streams to record), which is to be fed by both the negotiated media streams; a beep has to be played (beep) right before the recording starts to notify the UAC;

- o As seen before, the MS sends a provisional 202 response, to let the AS know the operation might need some time;
- o 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 (A3) with a terminated status; as explained previously, interlocutory REPORT messages with an update status would have been sent in case the preparation took longer than the timeout provided in the 202 message (e.g., if retrieving the resource via HTTP took longer than expected); once the dialog has been prepared and started, the UAC 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;
- o The AS acks the latest REPORT (A4), thus terminating this transaction, waiting for the result to come;
- o 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 contains information about the prompt (<promptinfo>), which is however not relevant to the scenario;
- o 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:

- o 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 again 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);
- o Again, the usual provisional 202 (C2) takes place;
 - o In the meanwhile, the MS prepares the new dialog and starts it, notifying the AS about it with a new REPORT (C3) with a terminated status: the connection is then passed to the IVR package, which plays the file on it;
 - o The AS acks the terminating REPORT (C4), now waiting for the announcement to end;
 - o 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;
 - o 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: 265

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart connectionid="10514b7f:6a900179">
    <dialog>
      <prompt>
        <media
          loc="http://www.example.com/demo/echorecord.mpg"/>
        </prompt>
        <record beep="true" maxtime="10s"/>
      </dialog>
    </dialogstart>
  </mscivr>
```

A2. AS <- MS (CFW 202)

CFW 796d83aa1ce4 202

A3. AS <- MS (CFW REPORT terminate)

CFW 796d83aa1ce4 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="68d6569"/>
</mscivr>
```

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

CFW 796d83aa1ce4 200

Seq: 1

B1. AS <- MS (CFW CONTROL event)

CFW 0eb1678c0bfc CONTROL

Control-Package: msc-ivr/1.0

Content-Type: application/msc-ivr+xml

Content-Length: 403

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <event dialogid="68d6569">
    <dialogexit status="1" reason="Dialog successfully completed">
      <promptinfo duration="9987" termmode="completed"/>
      <recordinfo duration="10017" termmode="maxtime">
        <mediainfo
loc="http://www.example.net/recordings/recording-68d6569.mpg"
type="video/mpeg" size="591872"/>
        </recordinfo>
      </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: 241

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart connectionid="10514b7f:6a900179">
    <dialog>
      <prompt>
        <media
loc="http://www.example.net/recordings/recording-68d6569.mpg"/>
        </prompt>
      </dialog>
    </dialogstart>
  </mscivr>
```

C2. AS <- MS (CFW 202)

CFW 1632eead7e3b 202

C3. AS <- MS (CFW REPORT terminate)

CFW 1632eead7e3b 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="5f5cb45"/>
</mscivr>
```

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

CFW 1632eead7e3b 200

Seq: 1

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')

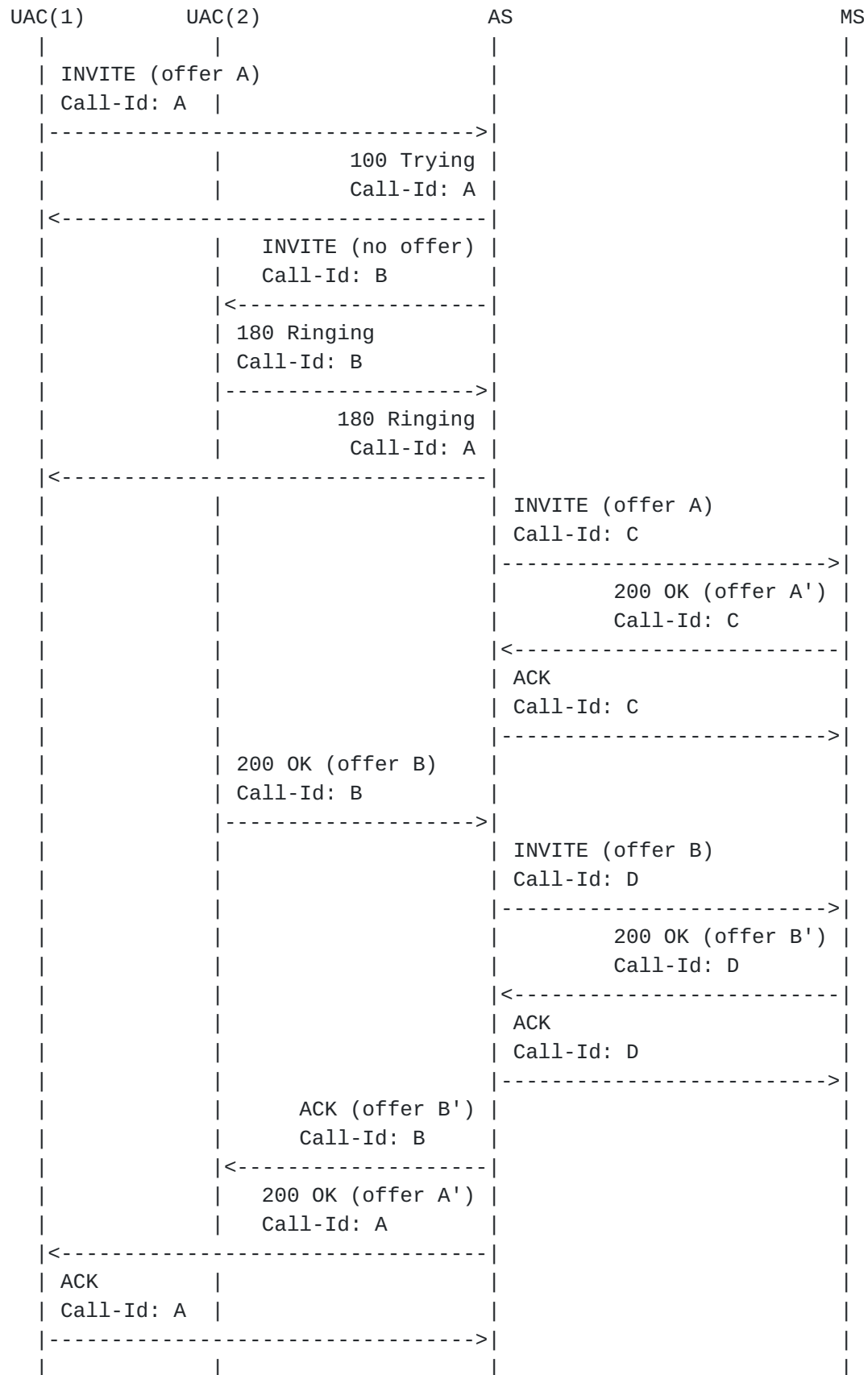
CFW 502a5fd83db8 200

6.2. Phone Call

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 prove useful 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](#).

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 17. 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. [\[RFC3725\]](#) provides a much more detailed overview on 3PCC patterns in several use cases. Only an explanatory sequence diagram is provided, without delving into the details of the exchanged SIP messages.




```

      .
      .
      .
      .

```

Figure 17: 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 somehow, 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 18: the MS is basically in the media path between the two UACs.

```

+-----+ UAC1 (RTP)      +-----+ UAC1 (RTP)      +-----+
|  UAC  |=====>| Media |=====>|  UAC  |
|   1   |<=====| Server |<=====|   2   |
+-----+          +-----+          +-----+
                UAC2 (RTP)      UAC2 (RTP)

```

Figure 18: Phone Call: Media Perspective

From the framework point of view, when the UACs' legs are not attached to anything yet, what appears is described in Figure 19: 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).

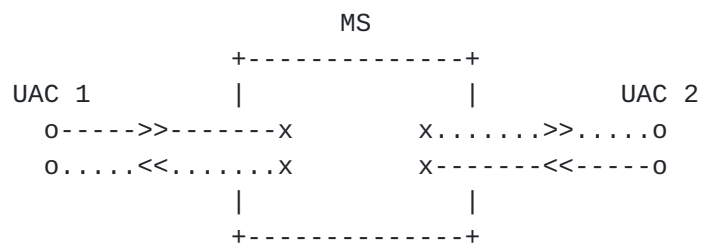


Figure 19: Phone Call: UAC Media Leg not attached

6.2.1. Direct Connection

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 exploiting the MS to only deal with the transcoding/adaption of the flowing frames, if needed.

This approach is depicted in Figure 20.



Figure 20: Phone Call: Direct Connection

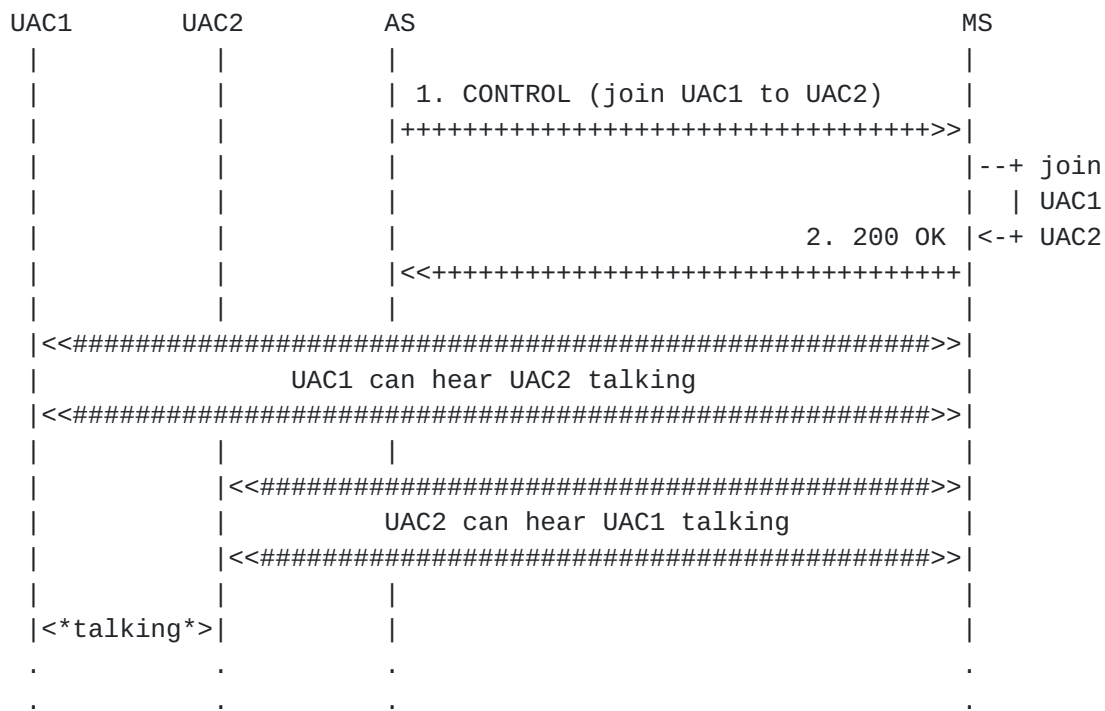


Figure 21: Direct Connection: Framework Transactions

The framework transactions needed to accomplish this scenario are very trivial and easy to understand. They basically are the same as the ones 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 UACs' 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: 130

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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](#).

[6.2.2](#). Conference-based Approach

The approach described in [Section 6.2.1](#) surely works for a basic phone call, but as already explained might have some drawbacks whenever more advanced features are needed. For instance, 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 an

Figure 23 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 10800s)	
		+++++	--+ start
			the
		B2. 200 OK	<--+ dialog
		<<+++++	
Recording +--			
of the mix			
has started +-->			
		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 23: 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:

- o 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, just as the 'n' value for the N-best audio mixing algorithm; besides, the video layout as well is set accordingly (single-view/dual-view);
- o 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 requests addressed to it, is 6013f1e;
- o 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=10800s), 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);
- o the transaction is handled by the MS, and when the dialog has been successfully started, a 200 OK is issued to the AS (B2); the message contains the dialogid associated with the dialog (00b29fb), which the AS must refer to for later notifications;
- o at this point, the AS attaches both UACs to the conference with two separate 'join' directives (C1/D1); when the MS confirms the success of both 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: 395

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <createconference reserved-talkers="2" reserved-listeners="2">
    <audio-mixing type="nbest" n="2"/>
    <video-layouts>
```



```
    <video-layout min-participants='1'>
      <single-view/>
    </video-layout>
    <video-layout min-participants='2'>
      <dual-view/>
    </video-layout>
  </video-layouts>
  <video-switch>
    <controller/>
  </video-switch>
</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: 208

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart conferenceid="6013f1e">
    <dialog>
      <record beep="false" maxtime="10800s"/>
    </dialog>
  </dialogstart>
</mscivr>
```

B2. AS <- MS (CFW 200 OK)

```
CFW 515f007c5bd0 200
Timeout: 10
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>
```

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

CFW 0216231b1f16 CONTROL
Control-Package: msc-mixer
Content-Type: application/msc-mixer+xml
Content-Length: 123

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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: 124

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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 requests the recording to be interrupted since the call between the UACs ended).

6.2.3. Recording a conversation

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](#).

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 data coming from UAC1, and a different recording for all the data 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 the two recordings; this way, a simple playout as described in [Section 6.1.2](#) 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 24 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](#). 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.

UAC3	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 &
	A2. 200 OK (conferenceid=Y)	<-+ its ID
	<+++++	
	B1. CONTROL (join UAC3 & confY)	
	++++++	
		--+ join
		UAC &
	B2. 200 OK	<-+ conf Y
	<+++++	
<<#####>>		
UAC is now a passive participant in the conference		
<<#####>>		
	C1. CONTROL (play REC1 on confY)	
	++++++	
	D1. CONTROL (play REC2 on confY)	
	++++++	
		--+ Start
		both
		the
		dialogs
	C2. 200 OK	<-+
	<+++++	

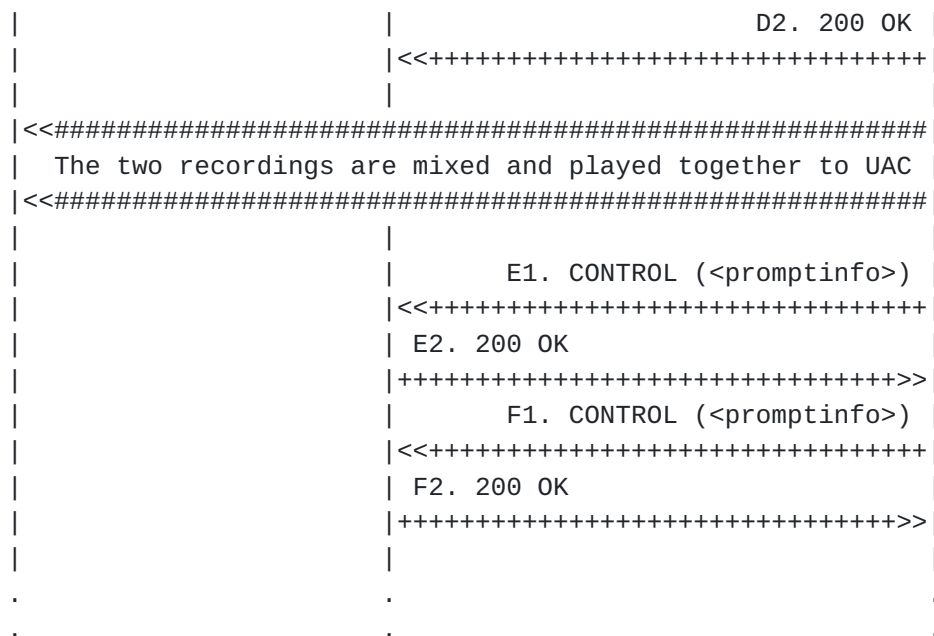


Figure 24: Phone Call: Playout of a Recorded Conversation

The diagram above assumes a recording of both the channels (UAC1 and UAC2) has already taken place. Later, when we desire to play the whole conversation to a new user, UAC3, the AS may take 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.

- o 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;
- o At this point, UAC3 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 AS (B2);
- o Once the conference has been created and UAC3 has been attached to it, the AS can request the playout of the recordings; in order to do so, it requests two concurrent <prompt> directives (C1 and D1), addressing respectively the recording of UAC1 (REC1) and UAC2 (REC2); both the prompts must be played on the previously created conference and not to UAC3 directly, as can be deduced from the

- 'conferenceid' attribute of the <dialog> element;
- o The transactions live their life exactly as explained for previous <prompt> examples; the originating transactions are first prepared and started (C2, D2), and then, as soon as any of the playout ends, a related 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: 312

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
 <createconference reserved-talkers="0" reserved-listeners="1">
 <audio-mixing type="controller"/>
 <video-layouts>
 <video-layout min-participants='1'>
 <dual-view/>
 </video-layout>
 </video-layouts>
 <video-switch>
 <controller/>
 </video-switch>
 </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: 214  
  
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">  
  <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: 229  
  
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">  
  <dialogstart conferenceid="2625069">  
    <dialog>  
      <prompt>  
        <media  
loc="http://www.example.net/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: 229

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart conferenceid="2625069">
    <dialog>
      <prompt>
        <media
loc="http://www.example.net/recordings/recording-39dfef4.mpg"/>
      </prompt>
    </dialog>
  </dialogstart>
</mscivr>
```

C2. AS <- MS (CFW 200 OK)

CFW 1c268d810baa 200
Timeout: 10
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>
```

D2. AS <- MS (CFW 200 OK)

CFW 3c2a08be4562 200
Timeout: 10
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>
```

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. Conferencing

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 contribution 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 25.

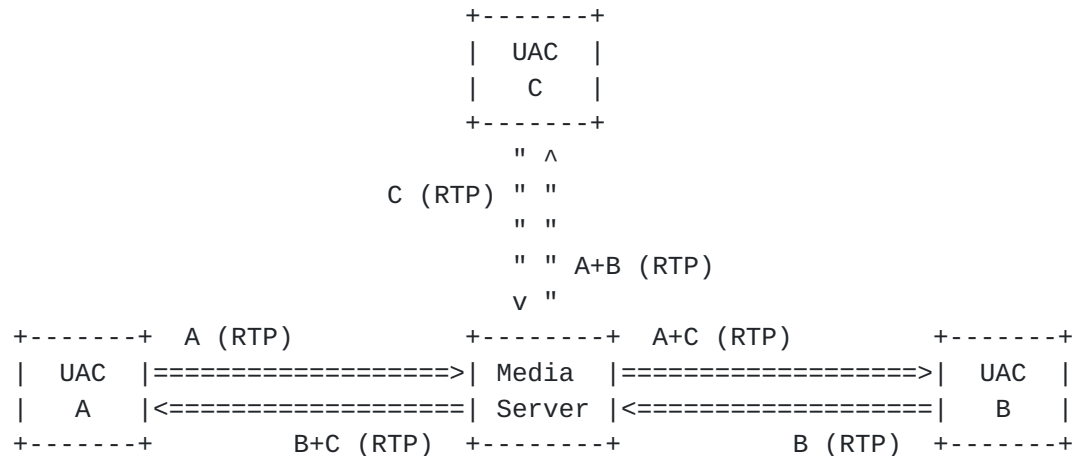


Figure 25: Conference: Media Perspective

From the framework point of view, when the UACs' legs are not attached to anything yet, what appears is described in Figure 26: 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).

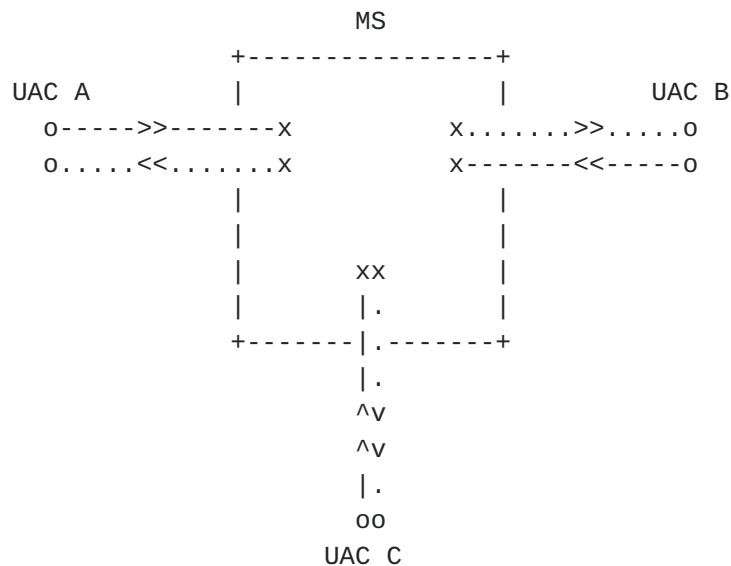


Figure 26: Conference: UAC Legs not attached

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 involving one or 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 (customers, 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;
5. Floor Control Scenario, which provides some guidance on how floor control could be involved in a MEDIACTRL-based media conference.

All of 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 interconnection of media sources 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 create a new conference instance in the Media Control Channel Framework is depicted in Figure 27:

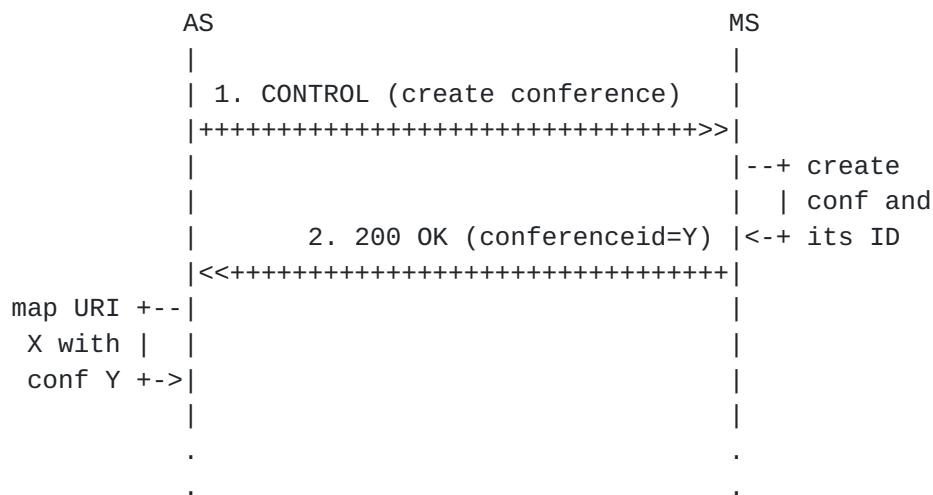


Figure 27: Conference: Framework Transactions

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

- o 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 the new conference instance; 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;
- o The MS creates the conference instance assigning a unique identifier to it (6146dd5), and completes the transaction with a 200 response (2);
- o 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: 489

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <createconference reserved-talkers="5" reserved-listeners="10">
    <audio-mixing type="nbest"/>
    <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>
    <video-switch>
      <controller/>
    </video-switch>
    <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>
```


6.3.1. Simple Bridging

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 its 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 participants. From the media perspective, the scenario can be seen as depicted in Figure 28.

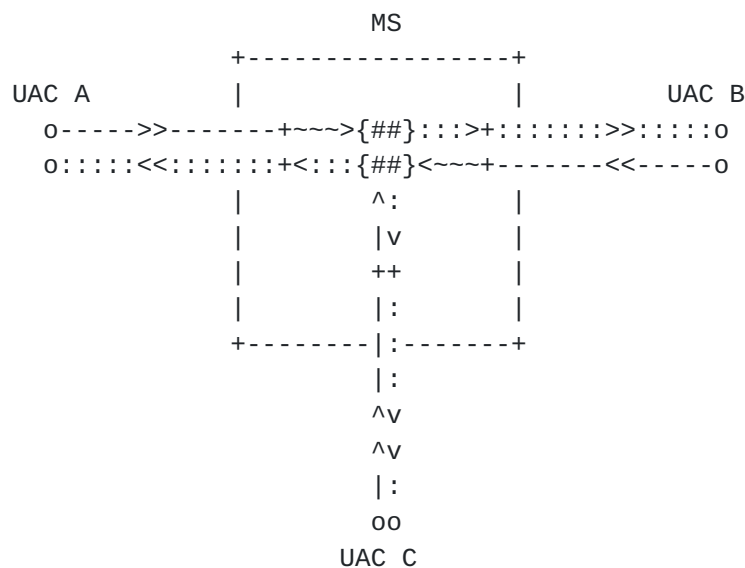


Figure 28: Conference: Simple Bridging

In the framework, the first step is obviously to create a new conference instance as seen in the introductory section (Figure 27). 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 29 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 in order to actually join the participants to the bridge so that they can interact in a conference. Please also note that, to make the diagram more readable, two different identifiers (UAC1 and UAC2) are used in place of the identifiers previously employed to introduce the scenario (UAC A, B, C).

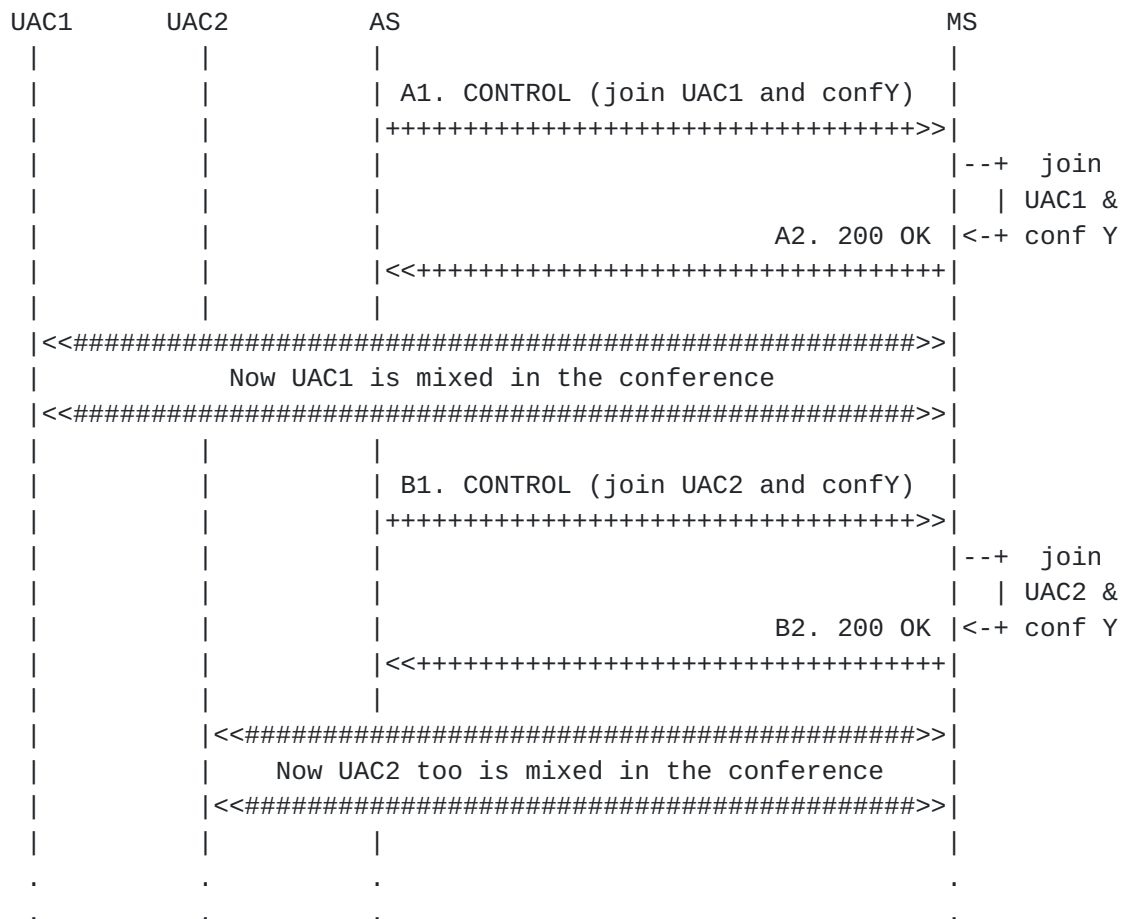


Figure 29: Simple Bridging: Framework Transactions (1)

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 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: 120

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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: 120

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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 30 shows how a framework transaction requesting such a directive might appear.

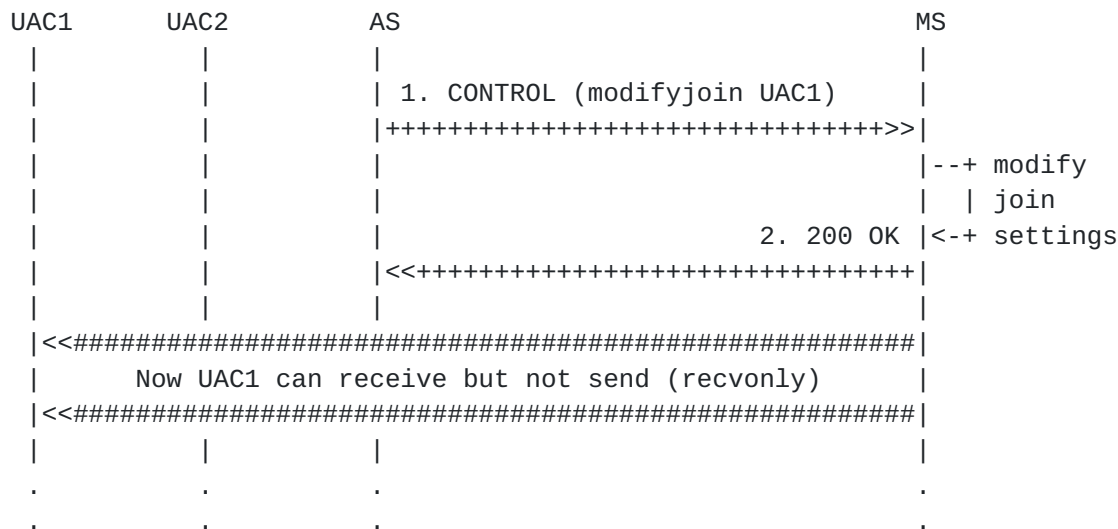


Figure 30: Simple Bridging: Framework Transactions (2)

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.

It is worth noticing that the `<stream>` settings are enforced according to both the provided direction AND the `id1` and `id2` identifiers. For instance, in this example `id1` refers to UAC1, while `id2` to the conference in the MS. This means that the required modifications have to be applied to the stream specified in the `<stream>` element of the message, along the direction which goes from 'id1' to 'id2' (as specified in the `<modifyjoin>` element of the message). In the provided example, the AS wants to mute UAC1 with

respect to the conference. To do so, the direction is set to 'recvonly', meaning that, for what concerns id1, the media stream is only to be received. If id1 referred to the conference and id2 to the UAC1, to achieve the same result the direction would have to be set to 'sendonly', meaning 'id1 (the conference) can only send to id2 (UAC1), and no media stream must be received'. Additional settings upon a <stream> (e.g., audio volume, region assignments and so on) follow the same approach, as it is presented in subsequent sections.

1. AS -> MS (CFW CONTROL)

CFW 57f2195875c9 CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 182

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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>
```

6.3.2. Rich Conference Scenario

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 effects 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 31.

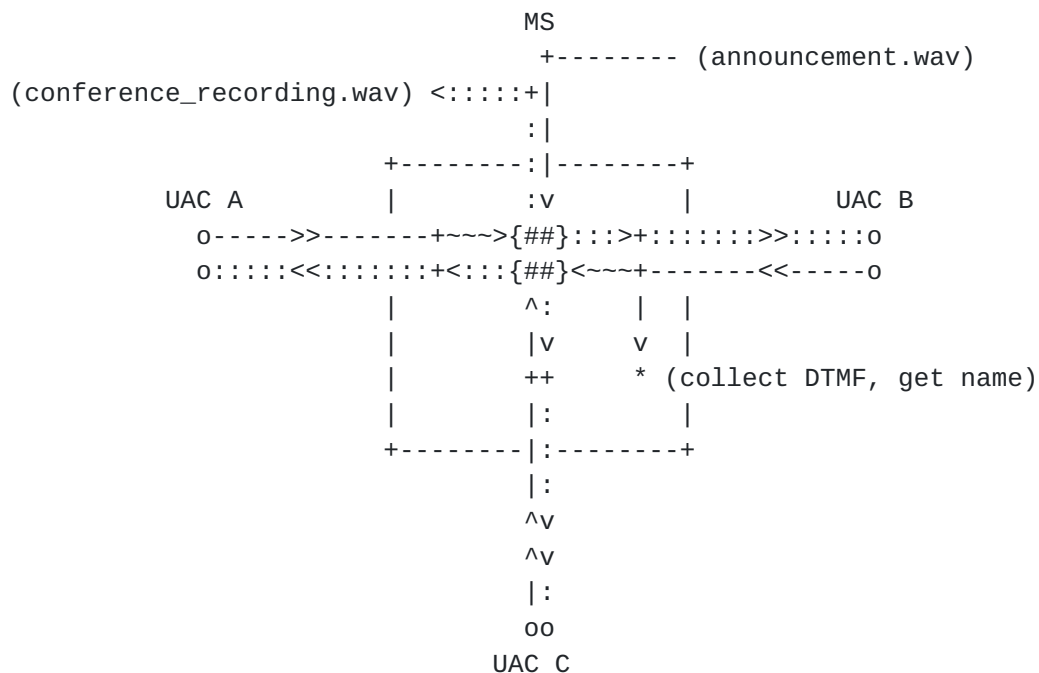


Figure 31: Conference: Rich Conference Scenario

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 advertise its arrival.

Figure 32 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)	
	++++++>>	
		--+ start
		the
	A2. 200 OK	<-+ dialog
	<<+++++	
<<#####		
"Please digit the PIN number to join the conference"		
<<#####		
#####>>		
DTMF digits are collected		--+ get
#####>>		DTMF
		<-+ digits
	B1. CONTROL (<collectinfo>)	
	<<+++++	
Compare DTMF +--	B2. 200 OK	
digits with	++++++>>	
the PIN number +->		
	C1. CONTROL (record name)	
	++++++>>	
		--+ start
		the
	C2. 200 OK	<-+ dialog
	<<+++++	
<<#####		
"Please state your name after the beep"		
<<#####		
#####>>		
Audio from the UAC is recorded (until timeout or DTMF)		--+ save
#####>>		in a
		<-+ file
	D1. CONTROL (<recordinfo>)	
	<<+++++	
Store recorded +--	D2. 200 OK	
file to play	++++++>>	
announcement in +->		
conference later		


```

|                                     | E1. CONTROL (join UAC & confY) |
|                                     | ++++++>> |
|                                     |                                     | --+ join
|                                     |                                     | | UAC &
|                                     |                                     | E2. 200 OK <--+ conf Y
|                                     | <+++++ |
|                                     |                                     |
| <<#####>> |
| UAC is now included in the mix of the conference |
| <<#####>> |
|                                     |
|                                     | F1. CONTROL (play name on confY) |
|                                     | ++++++>> |
|                                     |                                     | --+ start
|                                     |                                     | | the
|                                     |                                     | F2. 200 OK <--+ dialog
|                                     | <<+++++ |
|                                     |                                     |
| <<#####>> |
| Global announcement: "Simon has joined the conference" |
| <<#####>> |
|                                     |
|                                     | G1. CONTROL (<promptinfo>) |
|                                     | <<+++++ |
|                                     | G2. 200 OK |
|                                     | ++++++>> |
|                                     |
| .                                     .
| .                                     .

```

Figure 32: Rich Conference Scenario: Framework Transactions

As it can be deduced 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:

- o 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;
- o the transaction goes on as usual (A2), with the transaction being handled, and the dialog start being notified in a 200 OK; after that, the UAC is actually presented with the voice message, and is subsequently requested to insert the required PIN number;
 - o 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 if it is correct;
 - o 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;
 - o without delving again into the details of a recording-related transaction (C2), the AS finally gets an URI to the requested recording (D1, acked in D2);
 - o at this point, the AS attaches the UAC (id1) to the conference (id2) just as explained for Simple Bridging (E1/E2);
 - o 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), 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: 311

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart connectionid="10514b7f:6a900179">
    <dialog>
      <prompt>
        <media
          loc="http://www.example.net/prompts/conf-getpin.wav"
          type="audio/x-wav"/>
        </prompt>
        <collect maxdigits="4" escapekey="*" cleardigitbuffer="true"/>
      </dialog>
    </dialogstart>
  </mscivr>
```

A2. AS <- MS (CFW 200 OK)

CFW 50e56b8d65f9 200

Timeout: 10

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>
```

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: 373

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart connectionid="10514b7f:6a900179">
    <dialog>
      <prompt>
        <media
          loc="http://www.example.net/prompts/conf-rec-name.wav"
          type="audio/x-wav"/>
        </prompt>
        <record beep="true" maxtime="3s"/>
      </dialog>
      <stream media="audio" direction="sendonly"/>
      <stream media="video" direction="inactive"/>
    </dialogstart>
  </mscivr>
```

C2. AS <- MS (CFW 200 OK)

CFW 61fd484f196e 200

Timeout: 10

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>
```

D1. AS <- MS (CFW CONTROL event)

CFW 3ec13ab96224 CONTROL

Control-Package: msc-ivr/1.0

Content-Type: application/msc-ivr+xml

Content-Length: 402

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



```
<event dialogid="1cf0549">
  <dialogexit status="1" reason="Dialog successfully completed">
    <promptinfo duration="4988" termmode="completed"/>
    <recordinfo duration="3000" termmode="maxtime">
      <mediainfo
loc="http://www.example.net/recordings/recording-1cf0549.wav"
type="audio/x-wav" size="48044"/>
    </recordinfo>
  </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: 120

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
 <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: 334


```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart conferenceid="6146dd5">
    <dialog>
      <prompt>
        <media
loc="http://www.example.net/recordings/recording-1cf0549.wav"
type="audio/x-wav"/>
        <media
loc="http://www.example.net/prompts/conf-hasjoin.wav"
type="audio/x-wav"/>
      </prompt>
    </dialog>
  </dialogstart>
</mscivr>
```

F2. AS <- MS (CFW 200 OK)

```
-----
CFW 718c30836f38 200
Timeout: 10
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>
```

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
```


6.3.3. Coaching Scenario

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 [RFC4597](#) [RFC4597].

As it can be deduced 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 might be thinking), a mixing entity is needed in order to accomplish the scenario requirements. To summarize:

- o the customer (A) must only hear what the agent (B) says;
- o the agent (B) must be able to hear both the customer (A) and the coach (C);
- o 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 33.

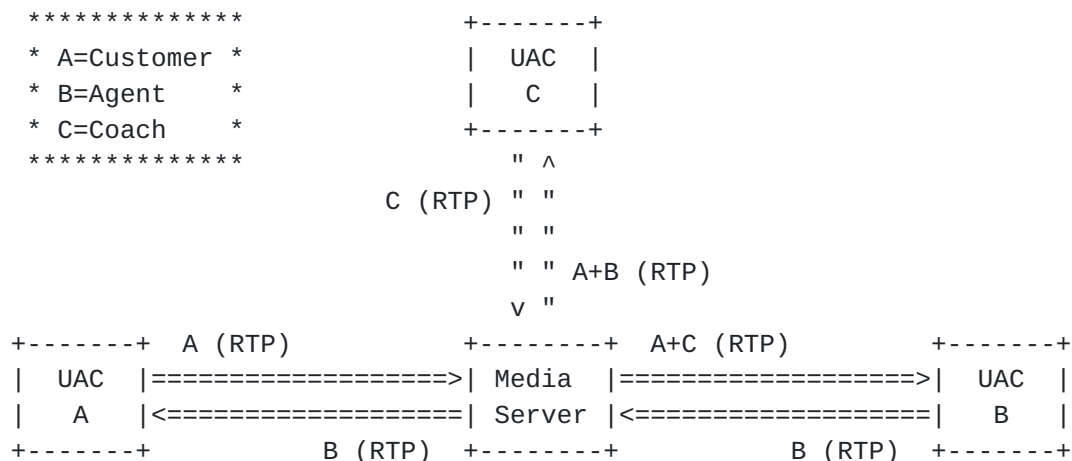


Figure 33: Coaching Scenario: Media Perspective

From the framework point of view, when the mentioned legs are not attached to anything yet, what appears is described in Figure 34.

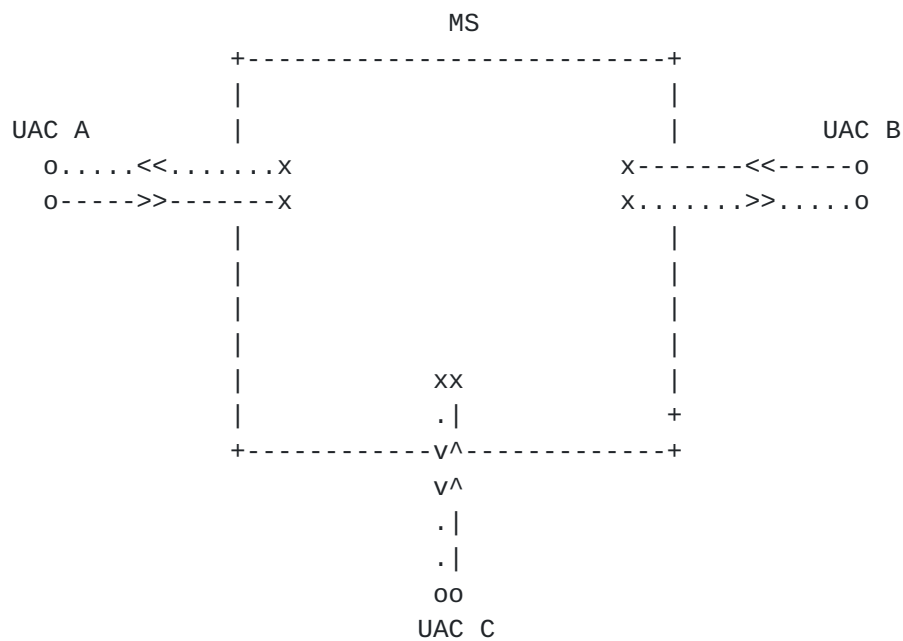


Figure 34: Coaching Scenario: UAC Legs not attached

What the scenario should look like is instead depicted in Figure 35. 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).

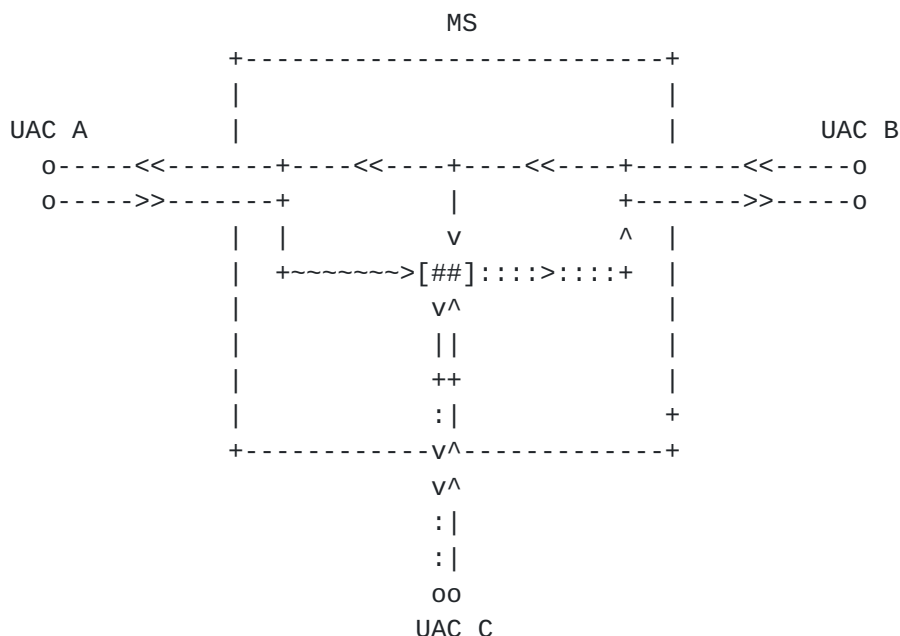
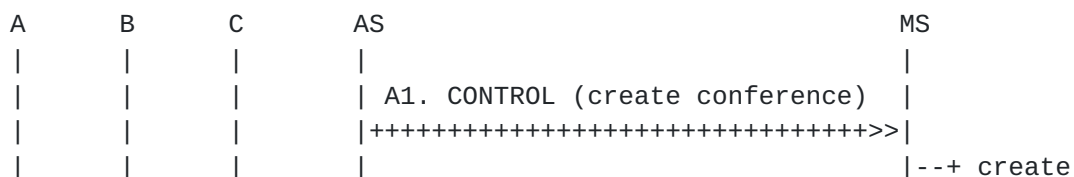


Figure 35: Coaching Scenario: UAC Legs mixed and attached

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 are needed. The needed steps can be summarized in the following list:

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 36:




```
| | 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  
| <<+++++  
| <<#####>  
Finally, Customer A is joined (recvonly) to Agent B  
| <<#####
```



```

.      .      .      .      .
.      .      .      .      .

```

Figure 36: Coaching Scenario: Framework Transactions

- o 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, that is to mix all the involved parties accordingly; since only three participants will be joined to it, 'reserved-talkers' is set to 3; 'reserved-listeners', instead, is set to 2, since only the agent and the coach must receive a mix, while the customer must be unaware of the coach; finally, the video layout is set to dual-view for the same reason, since only the customer and the agent must appear in the mix;
- o 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 requests addressed to it, is 1df080e;
- o now that the conference has been created, the AS joins the three actors to it with different policies, namely: (i) the customer A is joined as sendonly to the conference (B1), (ii) the agent B is joined as sendrecv to the conference (C1) and (iii) the coach is joined as sendrecv (but audio only) to the conference and with a lower volume (D1); the custom policies are enforced by means of properly constructed <stream> elements;
- o the MS takes care of the requests and acks them (B2, C2, D2); at this point, the conference will receive media from all the actors, but only provide the agent and the coach with the resulting mix;
- o to complete the scenario, the AS joins the customer A with the agent B directly as rcvonly (E1); the aim of this request is to provide customer A with media too, namely the media contributed by agent B; this way, customer A is unaware of the fact that its media are accessed by coach C by means of the hidden mixer;
- o the MS takes care of this request too and acks it (E2), concluding the scenario.

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

```

CFW 238e1f2946e8 CONTROL
Control-Package: msc-mixer
Content-Type: application/msc-mixer+xml
Content-Length: 329

```



```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <createconference reserved-talkers="3" reserved-listeners="2">
    <audio-mixing type="nbest"/>
    <video-layouts>
      <video-layout min-participants='1'>
        <dual-view/>
      </video-layout>
    </video-layouts>
    <video-switch>
      <controller/>
    </video-switch>
  </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: 226

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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: 122

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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: 221

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <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: 236  
  
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">  
  <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>
```

6.3.4. Sidebars

Within the context of conferencing, there could be the need for the so-called sidebars, or side conferences. It is the case, for instance, of two or more participants of a conference willing to create a side conference among each others, while still receiving part of the original conference mix in the background. Motivations for such an use case can be found in both [[RFC4597](#)] and [[RFC5239](#)]. It is clear that, in such a case, the side conference is actually a separate conference, which however must be related somehow to the

original conference. While the application level relationship is out of scope to this document (this belongs to XCON), the media stream relationship is, and it is consequently interesting to analyse how the mixes could be constructed in order to allow for such a feature making use of the MEDIACTRL specification.

The scenario presented in this section is a conference hosting four different participants: A, B, C and D. All these participants are attached to the conference as active senders and receivers of the existing media streams. At a certain point in time, two participants (B and D) decide to create a sidebar just for them. The sidebar they want to create is constructed so that only audio is involved. Besides, the audio mix of the sidebar must not be made available to the main conference. The mix of the conference, instead, must be attached to the sidebar, but with a lower volume (30%), being just a background to the actual conversation. This would allow both B and D to talk to each other without A and C listening to them, while B and D could still have an overview of what's happening in the main conference.

From the media and framework perspective, such a scenario can be seen as it is depicted in Figure 37.

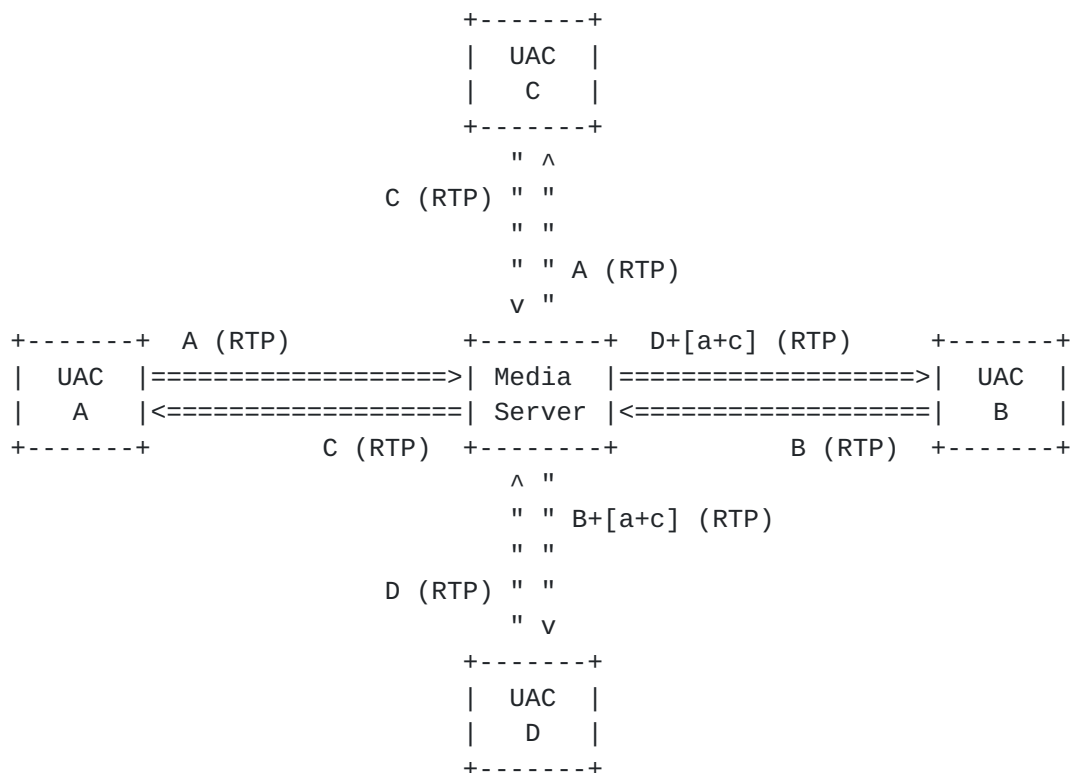


Figure 37: Sidebars: Media Perspective

From the framework point of view, when all the participants are attached to the main conference, what appears is described in Figure 38.

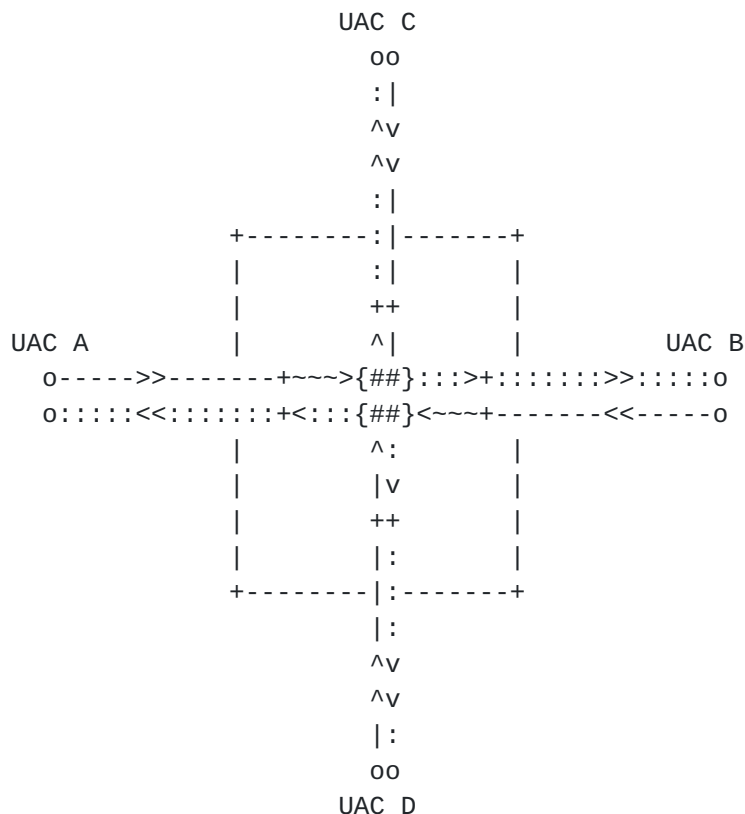


Figure 38: Sidebars: UAC Legs in main conference

What the scenario should look like is instead depicted in Figure 39. A new mixer is created to host the sidebar. The main mix is then attached as 'sendonly' to the sidebar mix at a lower volume (in order to provide the sidebar users with a background of the main conference). The two interested participants (B and D) have their audio leg detached from the main conference and attached to the sidebar. This detach can be achieved either by actually detaching the leg or by just modifying the status of the leg to inactive. Notice that this only affects the audio stream: the video of the two users is still attached to the main conference, and what happens at the application level may or may not have been changed accordingly (e.g., XCON protocol interactions).

Please notice that the main conference is assumed to be already in place, and the involved participants (A, B, C and D) to be already attached (sendrecv) to it.

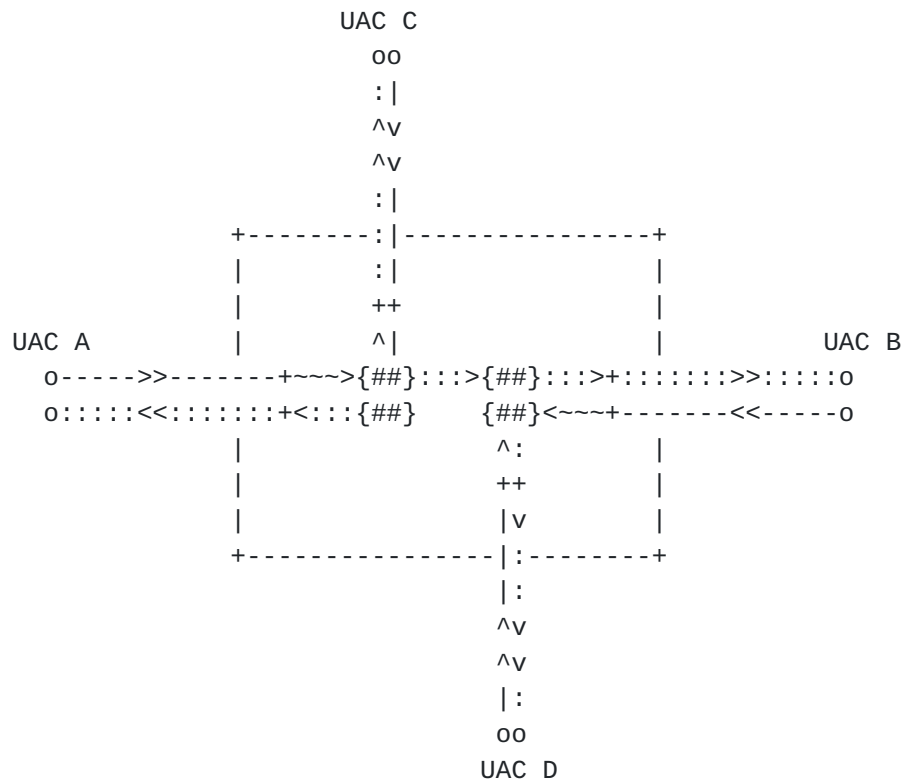


Figure 39: Sidebars: UAC Legs mixed and attached

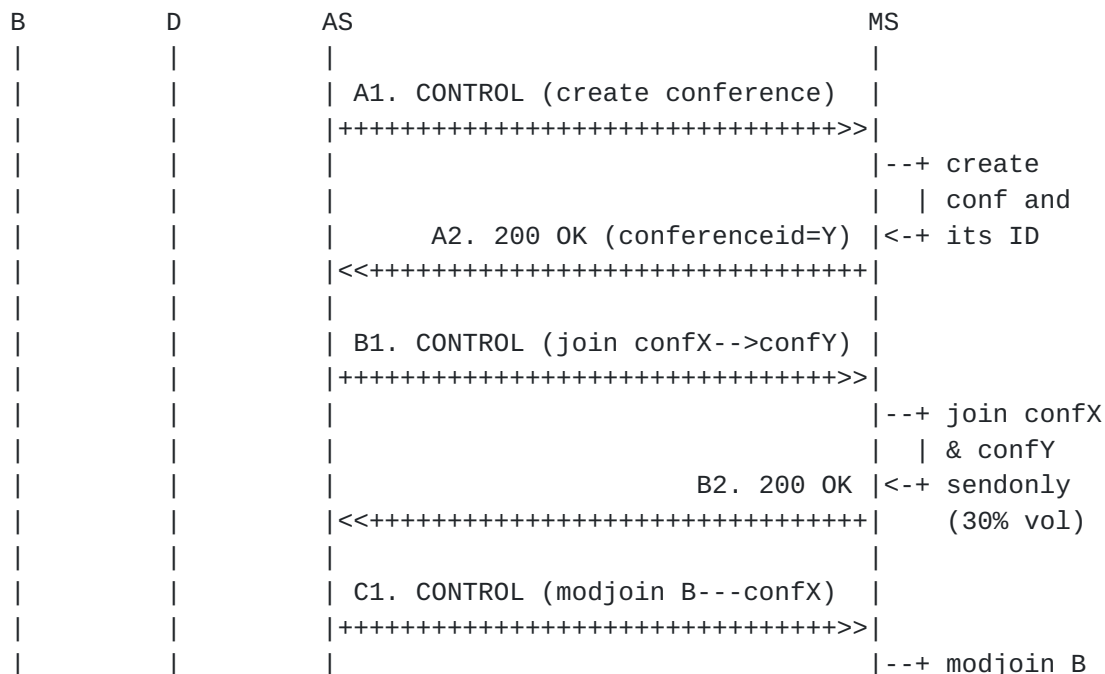
The situation may subsequently be reverted (e.g., destroying the sidebar conference and reattaching B and D to the main conference mix) in the same way. The AS would just need to unjoin B and D from the hidden conference and change their connection with the main conference back to 'sendrecv'. After unjoining the main mix and the sidebar mix, the sidebar conference could then be destroyed. Anyway, these steps are not described for brevity, considering similar transactions have already been presented.

In the framework the presented scenario 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 are needed. The needed steps can be summarized in the following list:

1. first of all, a hidden conference is created (the sidebar mix);
2. then, the main conference mix is attached to it as 'sendonly' and with a -5dB gain to limit the volume of its own contribution to 30% (so that B and D can hear each other louder, while still listening to what's happening in the main conference in background);
3. B and D are detached from the main mix for what concerns audio (modifyjoin with inactive status);
4. B and D are attached to the hidden sidebar mix for what concerns audio.

Note that for detaching B and D from the main mix, a <modifyjoin> with an 'inactive' status is used, instead of an <unjoin>. The motivation for this is related to how a subsequent rejoining of B and D to the main mix could take place. In fact, by using <modifyjoin> the resources created when first joining B and D to the main mix remain in place, even if marked as unused at the moment. An <unjoin>, instead, would actually free those resources, which could be granted to other participants joining the conference in the meanwhile. This means that, when needing to reattach B and D to the main mix, the MS could not have the resources to do so, resulting in an undesired failure.

A sequence diagram of such a sequence of transactions (where confX is the identifier of the pre-existing main conference mix) is depicted in Figure 40:




```

|                                     | | & confX
|                                     | <-+ (inactive)
|                                     | <<+++++
|                                     |
| D1. CONTROL (join B<-->confY)      |
| ++++++>>
|                                     | --+ join B
|                                     | | & confY
|                                     | D2. 200 OK <-+ sendrecv
|                                     | (audio)
|                                     | <<+++++
| <<#####>>
| Participant B is mixed (sendrecv) in the sidebar
| (A, C and D can't listen to her/him anymore)
| <<#####>>
|                                     |
| E1. CONTROL (modjoin D---confX)    |
| ++++++>>
|                                     | --+ modjoin D
|                                     | | & confX
|                                     | E2. 200 OK <-+ (inactive)
|                                     | <<+++++
|                                     |
| F1. CONTROL (join D<-->confY)      |
| ++++++>>
|                                     | --+ join D
|                                     | | & confY
|                                     | F2. 200 OK <-+ sendrecv
|                                     | (audio)
|                                     | <<+++++
| <<#####>>
| D is mixed (sendrecv) in the sidebar too
| (A and C can't listen to her/him anymore)
| <<#####>>
|
| .
| .
| .

```

Figure 40: Sidebars: Framework Transactions

- o First of all, the hidden conference mix is created (A1); the request is basically the same already presented in previous sections, that is a <createconference> request addressed to the Mixer package; the request is very lightweight, and asks the MS to only reserve two listener seats (reserved-listeners, since only B and D have to hear something) and three talker seats (reserved-listeners, considering that besides B and D also the main mix is

- an active contributor to the sidebar); the mixing will be driven by directives from the AS (mix-type=controller); when the mix is successfully created, the MS provides the AS with its identifier (519c1b9);
- o as a first transaction after that, the AS joins the audio from the main conference and the newly created sidebar conference mix (B1); only audio needs to be joined (media=audio), with a sendonly direction (i.e., only flowing from the main conference to the sidebar and not viceversa) and at a 30% volume with respect to the original stream (setgain=-5dB); a successful completion of the transaction is reported to the AS (B2);
 - o at this point, the AS makes the connection of B (2133178233:18294826) and the main conference (2f5ad43) inactive by means of a <modifyjoin> directive (C1); the request is taken care of by the MS (C2) and B is actually excluded from the mix for what concerns both sending and receiving;
 - o after that, the AS (D1) joins B (2133178233:18294826) to the sidebar mix created previously (519c1b9); the MS attaches the requested connections and sends confirmation to the AS (D2);
 - o the same transactions already done for B are placed for D as well (id1=1264755310:2beae5b), that is the <modifyjoin> to make the connection in the main conference inactive (E1-2) and the <join> to attach D to the sidebar mix (F1-2); at the end of these transactions, A and C will only listen to each other, while B and D will listen to each other and to the conference mix as a comfortable background.

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

CFW 7fdcc2331bef CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 198

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <createconference reserved-talkers="3" reserved-listeners="2">
    <audio-mixing type="controller"/>
  </createconference>
</mscmixer>
```

A2. AS <- MS (CFW 200 OK)

CFW 7fdcc2331bef 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="519c1b9"/>
</mscmixer>
```

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

CFW 4e6afb6625e4 CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 226

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <join id1="2f5ad43" id2="519c1b9">
    <stream media="audio" direction="sendonly">
      <volume controltype="setgain" value="-5"/>
    </stream>
  </join>
</mscmixer>
```

B2. AS <- MS (CFW 200 OK)

CFW 4e6afb6625e4 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, modifyjoin with inactive status)

CFW 3f2dba317c83 CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 193

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <modifyjoin id1="2133178233:18294826" id2="2f5ad43">
    <stream media="audio" direction="inactive"/>
  </modifyjoin>
</mscmixer>
```


C2. AS <- MS (CFW 200 OK)

CFW 3f2dba317c83 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>
```

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

CFW 2443a8582d1d CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 181

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <join id1="2133178233:18294826" id2="519c1b9">
    <stream media="audio" direction="sendrecv"/>
  </join>
</mscmixer>
```

D2. AS <- MS (CFW 200 OK)

CFW 2443a8582d1d 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, modifyjoin with inactive status)

CFW 436c6125628c CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 193

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <modifyjoin id1="1264755310:2beeae5b" id2="2f5ad43">
    <stream media="audio" direction="inactive"/>
  </modifyjoin>
</mscmixer>
```



```
</modifyjoin>
</mscmixer>
```

E2. AS <- MS (CFW 200 OK)

```
-----
CFW 436c6125628c 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>
```

F1. AS -> MS (CFW CONTROL, join to sidebar)

```
-----
CFW 7b7ed00665dd CONTROL
Control-Package: msc-mixer/1.0
Content-Type: application/msc-mixer+xml
Content-Length: 181

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <join id1="1264755310:2beae5b" id2="519c1b9">
    <stream media="audio" direction="sendrecv"/>
  </join>
</mscmixer>
```

F2. AS <- MS (CFW 200 OK)

```
-----
CFW 7b7ed00665dd 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>
```

6.3.5. Floor Control

As described in [[RFC4597](#)], floor control is a feature typically needed and employed in most conference scenarios. In fact, while not a mandatory feature to implement when realizing a conferencing application, it provides additional control on the media streams

contributed by participants, thus allowing for moderation of the available resources. The Centralized Conferencing (XCON) framework [[RFC5239](#)] suggests the use of the Binary Floor Control Protocol (BFCP) [[RFC4582](#)] to achieve the aforementioned functionality. That said, a combined use of floor control functionality and the tools made available by the MEDIACTRL specification for conferencing would definitely be interesting to investigate. [[RFC5567](#)] introduces two different approaches to integrating floor control with the MEDIACTRL architecture: (i) a topology where the floor control server is co-located with the AS; (ii) a topology where the floor control server is instead co-located with the MS. The two approaches are obviously different with respect to the amount of information the AS and the MS have to deal with, especially when thinking about the logic behind the floor queues and automated decisions. Nevertheless, considering how the Media Control Channel Framework is conceived, the topology (ii) would need a dedicated package (be it an extension or a totally new package) in order to make the MS aware of floor control, and allow it to interact with the interested UAC accordingly. At the time of writing such a package doesn't exist yet, and as a consequence only the topology (i) will be dealt with in the presented scenario.

The scenario will then assume the Floor Control Server (FCS) to be co-located with the AS. This means that all the BFCP requests will be sent by Floor Control Participants (FCP) to the FCS, which will make the AS directly aware of the floor statuses. The scenario for simplicity assumes the involved participants are already aware of all the identifiers needed in order to make BFCP requests for a specific conference. Such information may have been carried according to the COMEDIA negotiation as specified in [[RFC4583](#)]. It is important to notice that such information must not reach the MS: this means that, within the context of the 3PCC mechanism that may have been used in order to attach a UAC to the MS, all the BFCP-related information negotiated by the AS and the UAC must be removed before making the negotiation available to the MS, which may be unable to understand the specification. A simplified example of how this could be achieved is presented in Figure 41. Please notice that, within the context of this example scenario, different identifiers may be used to address the same entity: specifically, in this case UAC (the endpoint sending and receiving media) is also a FCP, as it negotiates a BFCP channel too.

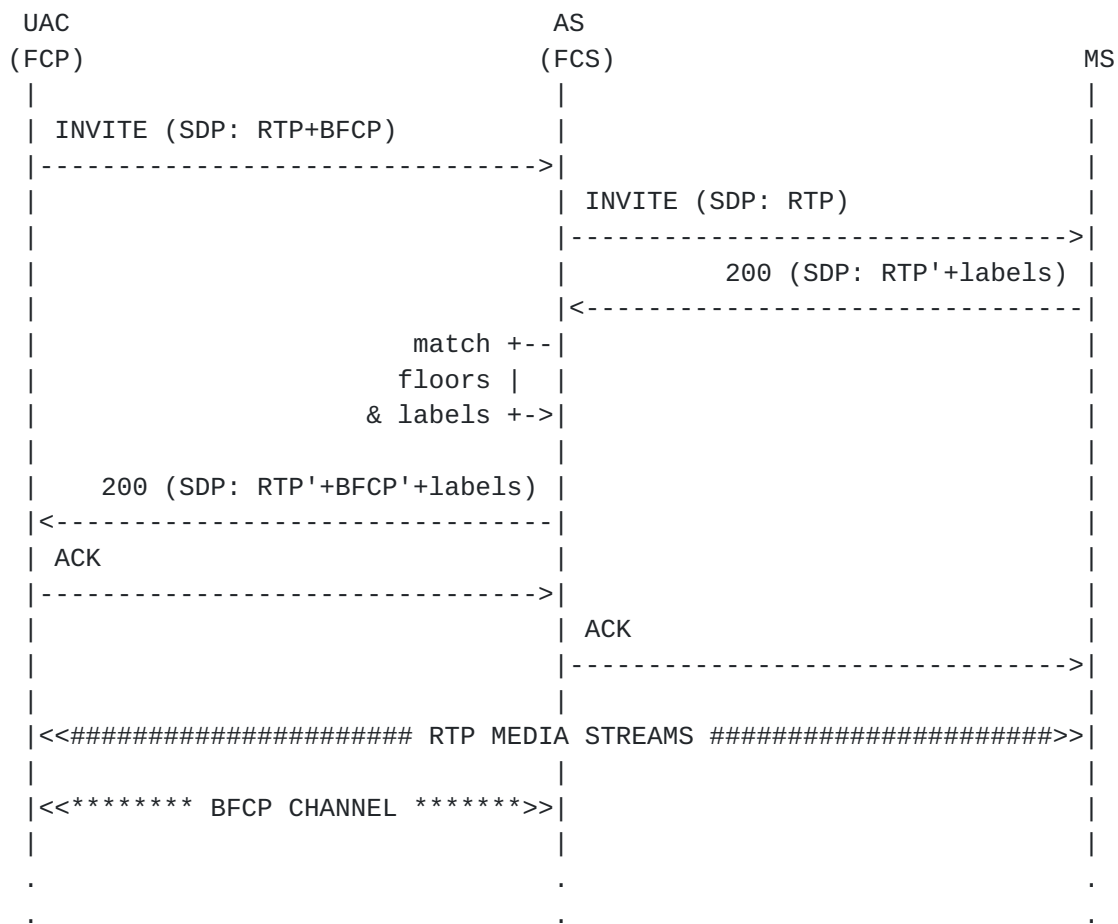


Figure 41: Floor Control: Example of Negotiation

From the media and framework perspective, such a scenario doesn't differ much from the already presented conferencing scenarios. It is more interesting to focus on the chosen topology for the scenario, which can be seen as it is depicted in Figure 42.

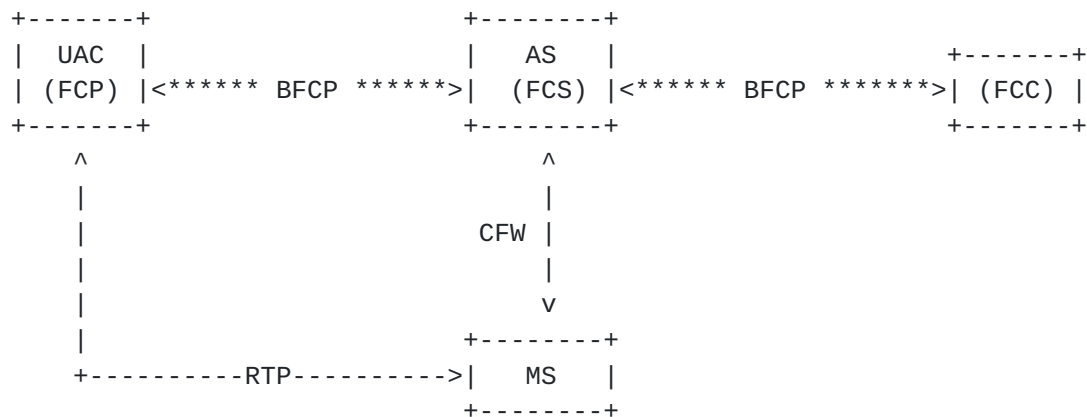


Figure 42: Floor Control: Overall Perspective

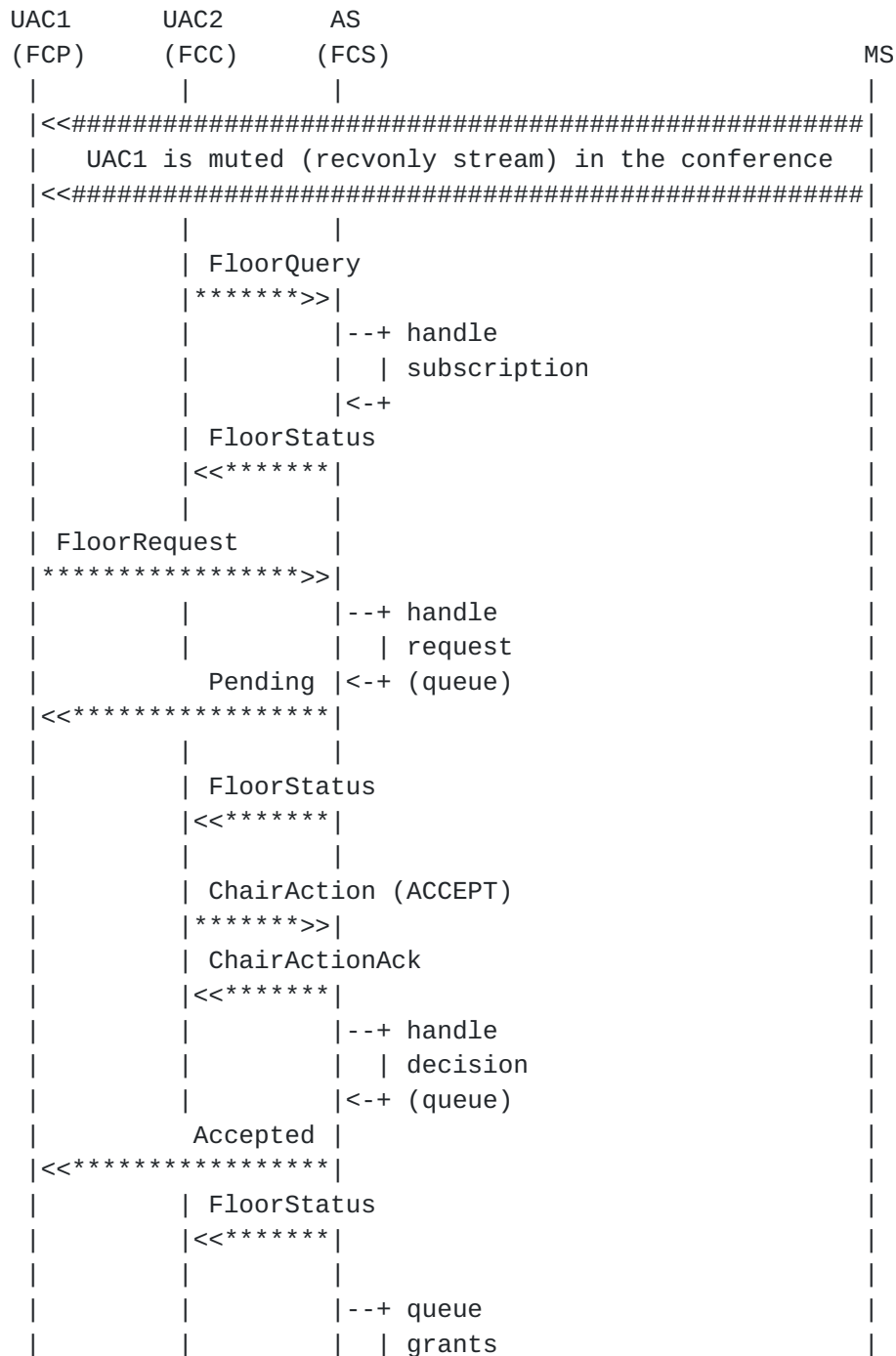
The AS, besides maintaining the already known SIP signaling among the involved parties, also acts as FCS for the participants in the conferences it is responsible of. In the scenario, two Floor Control Participants are involved: a basic Participant (FCP) and a Chair (FCC).

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 are needed. Assuming the conference has already been created, the participant has already been attached (recvonly) to it, and that the participant is aware of the involved BFCP identifiers, the needed steps can be summarized in the following list:

1. the assigned chair, FCC, sends a subscription for events related to the floor it is responsible of (FloorQuery);
2. the FCP sends a BFCP request (FloorRequest) to get access to the audio resource ("I want to speak");
3. the FCS (AS) sends a provisional response to the FCP (FloorRequestStatus PENDING), and handles the request in its queue; since a chair is assigned to this floor, the request is forwarded to the FCC for a decision (FloorStatus);
4. the FCC takes a decision and sends it to the FCS (ChairAction ACCEPTED);
5. the FCS takes note of the decision and updates the queue accordingly; the decision is sent to the FCP (FloorRequestStatus ACCEPTED); anyway, the floor has not been granted yet;
6. as soon as the queue allows it, the floor is actually granted to FCP; the AS, which is co-located with the FCS, understands in its business logic that such an event is associated with the audio resource being granted to FCP; as a consequence, a <modifyjoin> (sendrecv) is sent through the Control Channel to the MS in order

- to unmute the FCP UAC in the conference;
7. the event is notified to FCP (FloorRequestStatus GRANTED), thus ending the scenario.

A sequence diagram of such a sequence of transactions (also involving the BFCP message flow at a higher level) is depicted in Figure 43:



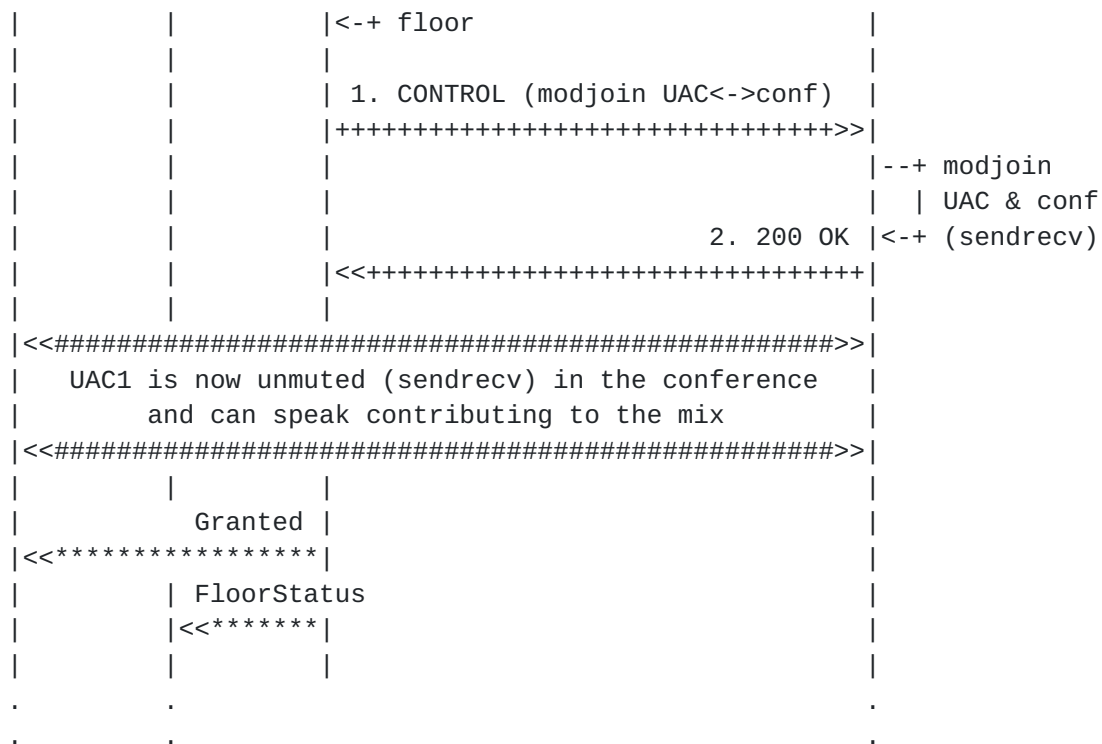


Figure 43: Floor Control: Framework Transactions

As it can easily be evinced from the above diagram, the complex interaction at the BFCP level only results in a single transaction at the MEDIACTRL level. In fact, the purpose of the BFCP transactions is to moderate access to the audio resource, which means providing the event trigger to MEDIACTRL-based conference manipulation transactions. Before being granted the floor, the FCP UAC is excluded from the conference mix at the MEDIACTRL level (recvonly). As soon as the floor has been granted, the FCP UAC is included in the mix. In MEDIACTRL words:

- o since the FCP UAC must be included in the audio mix, a `<modifyjoin>` is sent to the MS in a `CONTROL` directive; the `<modifyjoin>` has as identifiers the `connectionid` associated with the FCP UAC (`e1e1427c:1c998d22`) and the `conferenceid` of the mix (`cf45ee2`); the `<stream>` element tells the MS that the audio media stream between the two must become bidirectional (`sendrecv`), changing the previous status (`recvonly`); please notice that in this case only audio was involved in the conference; in case video was involved as well, and video had not to be changed, a `<stream>` directive for video had to be placed in the request as well in order to maintain its current status.

1. AS -> MS (CFW CONTROL)

CFW gh67ffg56w21 CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 182

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <modifyjoin id1="e1e1427c:1c998d22" id2="cf45ee2">
    <stream media="audio" direction="sendrecv"/>
  </modifyjoin>
</mscmixer>
```

2. AS <- MS (CFW 200 OK)

CFW gh67ffg56w21 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. Additional Scenarios

This section includes additional scenarios that can be of interest when dealing with AS<->MS flows. The aim of the following subsections is to present the use of peculiar features provided by the IVR package, specifically variable announcements, VCR (video cassette recorder) prompts, parallel playback, recurring dialogs and custom grammars. To describe how call flows involving such features might happen, three sample scenarios have been chosen:

1. Voice Mail (variable announcements for digits, VCR controls);
2. Current Time (variable announcements for date and time, parallel playback).
3. DTMF-driven Conference Manipulation (recurring dialogs, custom grammars).

6.4.1. Voice Mail

An application that typically makes use of the services an 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, control 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 interactions as possible that may happen between the AS and the MS in such a context. The covered scenario, depicted as a sequence diagram in Figure 44, 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 with an announcement giving him the amount of available new messages in the mailbox; after that, the UAC can 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 or organize them, 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 already been introduced in previous sections, and so representing them would be redundant. This means the presented call flows assume that some messages have already been recorded, and that they are available at reachable locations. The example also assumes that the AS has placed the recordings in its own storage facilities, considering it is not safe to rely upon the internal MS storage which is likely to be temporary.

UAC	AS	MS
	A1. CONTROL (play variables and	
	collect the user's choice)	
	++++++>>>	
		prepare &


```

|                                     | --+ start
|                                     | | the
|                                     | A2. 200 OK |<-+ dialog
|                                     |<<+++++
|                                     |
|<<#####
| "You have five messages ..."
|<<#####
|                                     |
|                                     | B1. CONTROL (<collectinfo>) |
|                                     |<<+++++
|                                     | B2. 200 OK
|                                     |+++++>>
|                                     |
|                                     | C1. CONTROL (VCR for chosen msg) |
|                                     |+++++>>
|                                     |
|                                     | prepare &
|                                     | --+ start
|                                     | | the
|                                     | C2. 200 OK |<-+ dialog
|                                     |<<+++++
|                                     |
|<<#####
| "Hi there, I tried to call you but..."
|<<#####
|                                     | --+
|                                     | | handle
|                                     | | VCR-
|#####>> | | driven
| The UAC controls the playout using DTMF | | (DTMF)
|#####>> | | playout
|                                     |<-+
|                                     | D1. CONTROL (<dtmfnotify>) |
|                                     |<<+++++
|                                     | D2. 200 OK
|                                     |+++++>>
|                                     |
| .
| (other events are received in the meanwhile) |
| .
|                                     | E1. CONTROL (<controlinfo>) |
|                                     |<<+++++
|                                     | E2. 200 OK
|                                     |+++++>>
|                                     |
| .
| .

```

Figure 44: Voice Mail: Framework Transactions

The framework transaction steps are described in the following:

- o 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 loc="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...");a DTMF collection is requested as well (<collect>) to be taken after the prompts have been played; the AS is only interested in a single digit (maxdigits=1);
- o the transaction is handled by the MS and, in case everything works 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 200 OK message (A2);
- o the AS then waits for the dialog to end in order to retrieve the results it is interested in (in this case, the DTMF tone the UAC chooses, since it will affect which message will have to be played subsequently);
- o 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 operation 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;
- o 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;

- o the MS prepares the dialog and, when the playout starts, notifies it in a terminating 200 OK message (C2); at this point, the UAC is presented with the prompt, and can make use of DTMF digits to control the playback;
- o 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);
- o 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: 429

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart connectionid="10514b7f:6a900179">
    <dialog>
      <prompt>
        <media
          loc="http://www.example.net/prompts/vm-youhave.wav"
          type="audio/x-wav"/>
        <variable value="5" type="digits"/>
        <media
          loc="http://www.example.net/prompts/vm-messages.wav"
          type="audio/x-wav"/>
      </prompt>
      <collect maxdigits="1" escapekey="*"
        cleardigitbuffer="true"/>
    </dialog>
  </dialogstart>
</mscivr>
```

A2. AS <- MS (CFW 200 OK)

CFW 2f931de22820 200

Timeout: 10

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>
```

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: 423

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart connectionid="10514b7f:6a900179">
    <dialog>
      <prompt bargein="false">
        <media
loc="http://www.example.com/messages/recording-4ca9fc2.mpg"/>
        </prompt>
      <control gotostartkey="1" gotoendkey="3">
```



```
        ffkey="6" rwkey="4" pausekey="7" resumekey="9"
        volupkey="#" voldownkey="*"/>
    </dialog>
  <subscribe>
    <dtmfsub matchmode="control"/>
  </subscribe>
</dialogstart>
</mscivr>
```

C2. AS <- MS (CFW 200 OK)

```
-----
CFW 3140c24614bb 200
Timeout: 10
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>
```

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="2008-12-16T12:58:36Z"/>
  </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="2008-12-16T12:58:36Z"/>
        <controlmatch dtmf="4" timestamp="2008-12-16T12:58:37Z"/>
        <controlmatch dtmf="7" timestamp="2008-12-16T12:58:38Z"/>
        <controlmatch dtmf="9" timestamp="2008-12-16T12:58:40Z"/>
      </controlinfo>
    </dialogexit>
  </event>
</mscivr>
```

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

CFW 3ffab81c21e9 200

6.4.2. Current Time

An interesting scenario to realize with the help of the MS provided features is what is typically called 'Current Time'. A UAC calls a URI, which presents the caller with the current date and time. As it can easily be deduced by the very nature of the application, variable announcements play an important role in this scenario. In fact, rather than having the AS build an announcement according to the current time using different framework messages, it is much easier to rely upon the variable announcements mechanism provided by the IVR package, which includes ways to deal with dates and times in several fashions.

To make the scenario more interesting and have it cover more functionality, the application is also assumed to have a background music played during the announcement. Considering that most of the announcements will be variable, a means is needed to have more streams played in parallel on the same connection. This can be achieved in two different ways:

1. two separate and different dialogs, playing respectively the variable announcements and the background track;

2. a single dialog implementing a parallel playback.

The first approach assumes the available MS implements implicit mixing, which may or may not be supported, since it's a recommended feature but not a mandatory one. The second approach instead assumes the MS implements support for more streams of the same media type (in this case audio) in the same dialog, which, exactly as implicit mixing, is not to be given for granted. Considering the first approach is quite straightforward to understand, the presented scenario makes use of the second one, and assumes the available MS supports parallel playback of more audio tracks within the context of the same dialog.

That said, the covered scenario, depicted as a sequence diagram in Figure 45, will be the following:

1. The UAC INVITES a URI associated with the Current Time application, 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 announcement including: (i) a voice stating the current date and time; (ii) a background music track; (iii) a mute background video track.

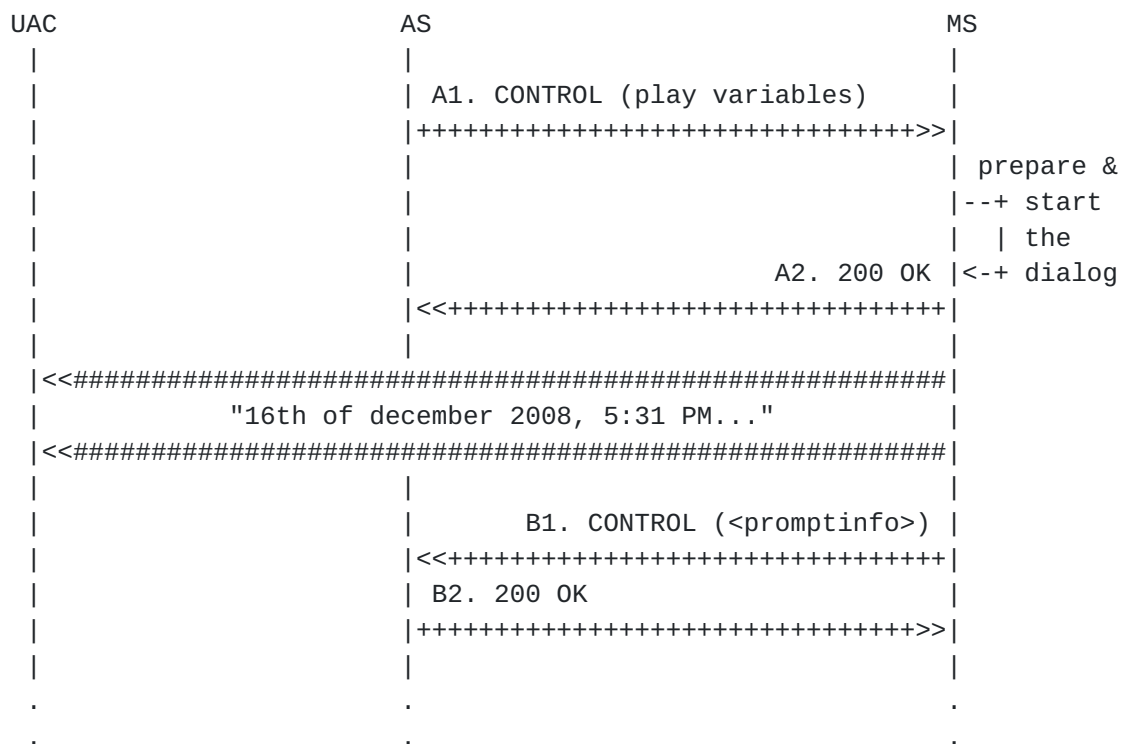


Figure 45: Current Time: Framework Transactions

The framework transaction steps are described in the following:

- o The first transaction (A1) is addressed to the IVR package (msc-ivr); it is basically a 'playannouncement' dialog, but, unlike all the scenarios presented so far, it includes directives for a parallel playback, as indicated by the 'par' element; there are three flows to play in parallel:
 - * a sequence ('seq') of variable and static announcements (the actual time and date);
 - * a music track ('media=music.wav') to be played in background at a lower volume (soundLevel=50%);
 - * a mute background video track (media=clock.mpg).
 The global announcement ends when the longest of the three parallel steps ends (endsync=last); this means that, if one of the steps ends before the others, the step is muted for the rest of the playback. About the series of static and variable announcements, in this example this is requested by the AS:
 - * play a wav file ("Tuesday...");
 - * play a date ("16th of december 2008..."), by building it (variable: date with a ymd=year/month/day format);
 - * play a time ("5:31 PM..."), by building it (variable: time with a t12=12 hour day format, am/pm).
- o the transaction is extended by the MS (A2) and, in case everything went fine (i.e., the MS retrieved all the audio files and successfully built the variable ones, and it supports parallel playback as requested), the dialog is started; its start is reported, together with the associated identifier (415719e) to the AS in a terminating REPORT message (A3);
- o the AS acks the REPORT (A4), and waits for the dialog to end in order to conclude the application, or proceed to further steps if required by the application itself;
- o when the last of the three parallel announcements ends, the dialog terminates, and an event (B1) is triggered to the AS with the relevant information (promptinfo); the AS acks (B2) and terminates the scenario.

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

```
CFW 0c7680191bd2 CONTROL
Control-Package: msc-ivr/1.0
Content-Type: application/msc-ivr+xml
Content-Length: 506
```

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



```

<dialogstart connectionid="21c8e07b:055a893f">
  <dialog>
    <prompt bargein="true">
      <par endsync="last">
        <seq>
          <media loc="http://www.example.com/day-2.wav"/>
          <variable value="2008-12-16" type="date" format="ymd"/>
          <variable value="17:31" type="time" format="t12"/>
        </seq>
        <media loc="http://www.example.com/music.wav"
          soundLevel="50%"/>
        <media loc="http://www.example.com/clock.mpg"/>
      </par>
    </prompt>
  </dialog>
</dialogstart>
</mscivr>

```

A2. AS <- MS (CFW 200 OK)

```

CFW 0c7680191bd2 200
Timeout: 10
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="415719e"/>
</mscivr>

```

B1. AS <- MS (CFW CONTROL event)

```

CFW 4481ca0c4fca 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="5db01f4">
    <dialogexit status="1" reason="Dialog successfully completed">
      <promptinfo duration="8046" termmode="completed"/>
    </dialogexit>
  </event>
</mscivr>

```

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

CFW 4481ca0c4fca 200

6.4.3. DTMF-driven Conference Manipulation

To complete the scenarios presented in [Section 6.3](#), this section deals with how the AS can make use of the MS in order to detect DTMF tones from conference participants, and take actions on the conference accordingly. A typical example is when participants in a conference are provided with specific codes to:

- o mute/unmute themselves in the conference;
- o change their volume in the conference, or the volume of the conference itself;
- o change the video layout in the conference, if allowed;
- o kick abusing users from the conference;

and so on. To achieve all this, the simplest thing an AS can do is to prepare a recurring DTMF collection for each participant with specific grammars to match. In case the collected tones match the grammar, the MS would notify them to the AS, and start the collection again. Upon receival of <collectinfo> events, the AS would in turn originate the proper related request, e.g., a <modifyjoin> on the participant's stream with the conference. This is made possible by three features provided by the IVR package:

1. the 'repeatCount' attribute;
2. the subscription mechanism;
3. the Speech Recognition Grammar Specification (SRGS) [[SRGS](#)].

The first allows for recurring instances of the same dialog without the need of additional requests upon completion of the dialog itself. In fact, the 'repeatCount' attribute indicates how many times the dialog has to be repeated: when the attribute has the value 0, it means that the dialog has to be repeated indefinitely, meaning that it's up to the AS to destroy it by means of a <dialogterminate> request when the dialog isn't needed anymore. The second, instead, allows the AS to subscribe to events related to the IVR package without waiting for the dialog to end, e.g., matching DTMF collections in this case. The last, finally, allows for custom matching grammars to be specified: this way, only a subset of the possible DTMF strings can be specified, so that only the matches the AS is interested in are reported. Different grammars other than SRGS may be supported by the MS, which achieve the same result: anyway, this document will only describe the use of an SRGS grammar, since support for SRGS is mandated in the IVR package specification.

To identify a single sample scenario, we assume a participant has already successfully joined a conference, e.g., as detailed in Figure 32. Besides, we assume the following codes are to be provided within the conference to participants in order to let them take advantage of advanced features:

1. *6 to mute/unmute themselves (on/off trigger);
2. *1 to lower their own volume in the conference, and *3 to raise it;
3. *7 to lower the volume of the conference stream they are receiving, and *9 to raise it;
4. *0 to leave the conference.

This means that six different codes are supported, and are to be matched in the requested DTMF collection. All other codes are collected by the MS, but discarded, and no event is triggered to the AS. Considering all the codes have the '*' (star) DTMF code in common, the following is an example of an SRGS grammar that may be used in the request by the AS:

```
<grammar mode="dtmf" version="1.0"
  xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="digit">
    <one-of>
      <item>0</item>
      <item>1</item>
      <item>3</item>
      <item>6</item>
      <item>7</item>
      <item>9</item>
    </one-of>
  </rule>
  <rule id="action" scope="public">
    <item>
      *
      <item><ruleref uri="#digit"/></item>
    </item>
  </rule>
</grammar>
```

As it can be deduced by looking at the grammar, the presented SRGS XML code specifies exactly the requirements for the collections to match: the rule is to match any string which has a star ('*') followed by just one of any supported digit (0, 1, 3, 6, 7, 9). Such grammar, as stated in the IVR package specification, may be provided

either inline in the request itself or referenced externally by means of the 'src' attribute. In the scenario example, we'll put it inline, but an external reference to the same document would achieve exactly the same result.

Figure 46 shows how the AS might request the recurring collection for a UAC: as already anticipated before, the example assumes the UAC is already a participant in the conference.

UAC	AS	MS
	A1. CONTROL (recurring collection)	
	++++++	
		--+ start
		the
	A2. 200 OK	<-+ dialog
	<<+++++	
	#####>>	
	Recurring DTMF digit collection starts	--+ get
	#####>>	DTMF
		<-+ digits
	B1. CONTROL (dtmfinfo=*1)	
	<<+++++	
	B2. 200 OK	--+ get
	++++++	DTMF
		<-+ digits
	C1. CONTROL (modifyjoin UAC1-->conf)	
	++++++	
		--+ modify
		UAC
	C2. 200 OK	<-+ volume
	<<+++++	
	#####>>	
	Volume of UAC in conference is lowered	
	#####>>	
	D1. CONTROL (dtmfinfo=*9)	
	<<+++++	
	D2. 200 OK	--+ get
	++++++	DTMF
		<-+ digits
	E1. CONTROL (modifyjoin UAC1<--conf)	
	++++++	
		--+ modify


```

|                                     | | conf
|                                     | E2. 200 OK |<-+ volume
|<<+++++|
|<<#####|
| Now UAC can hear the conference mix at a higher volume |
|<<#####|
|                                     |
|                                     | F1. CONTROL (dtmfinfo=*6) |
|<<+++++|
| F2. 200 OK | --+ get
| ++++++>> | | DTMF
|                                     |<-+ ditigs
| G1. CONTROL (modifyjoin UAC1-->conf) |
| ++++++>> |
|                                     | --+ mute
|                                     | | UAC in
|                                     | G2. 200 OK |<-+ conf
|<<+++++|
|                                     |
|#####>>|
| UAC is now muted in the conference |
|#####>>|
|                                     |
|                                     | H1. CONTROL (dtmfinfo=*0) |
|<<+++++|
| H2. 200 OK | --+ get
| ++++++>> | | DTMF
|                                     |<-+ ditigs
| I1. CONTROL (destroy DTMF dialog) |
| ++++++>> |
|                                     | --+ delete
|                                     | | the
|                                     | I2. 200 OK |<-+ dialog
|<<+++++| (DTMF
|                                     | collection
|                                     | stops)
| J1. CONTROL (dialogexit) |
|<<+++++|
| J2. 200 OK |
| ++++++>> |
|#####>>|
| No more tones from UAC are collected |
|#####>>|
|                                     |
| K1. CONTROL (unjoin UAC1<-X->conf) |
| ++++++>> |

```

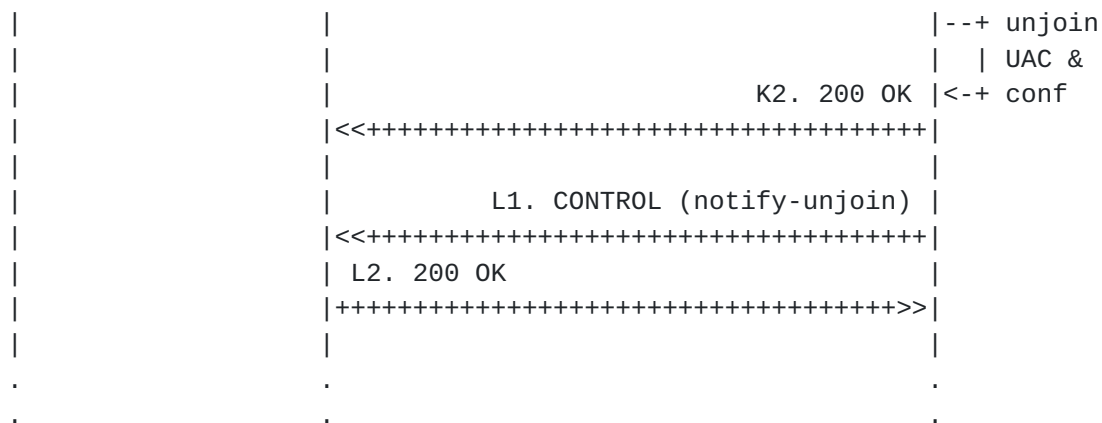



Figure 46: DTMF-driven Conference Manipulation: Framework Transactions

As it can be deduced 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: in fact, 'dtmfnotify' events are used to take actions according to the purpose the DTMF codes are meant for. The framework transaction steps are the following:

- o The UAC is already in the conference, and so the AS starts a recurring collect with a grammar to match. This is done by placing a CONTROL request addressed to the IVR package (A1); the operation to implement is a <collect>, and we are only interested in two-digit DTMF strings (maxdigits). The AS is not interested in a DTMF terminator (termchar is set to a non-conventional DTMF character), and the DTMF escape key is set to '#' (the default is '*', which would conflict with the code syntax for the conference, and so needs to be changed). A custom SRGS grammar is provided inline (<grammar> with mode=dtmf). The whole dialog is to be repeated indefinitely (dialog has repeatCount=0), and the AS wants to be notified when matching collections occur (dtmfsub with matchmode=collect).
- o The request is handled by the MS as already explained in previous sections (A2), and then successfully started (dialogid=01d1b38). This means the MS has started collecting DTMF tones from UAC.
- o The MS collects a matching DTMF string from UAC (*1). Since the AS subscribed to this kind of event, a CONTROL event notification (dtmfnotify) is triggered by the MS (B1), including the collected tones. Since the dialog is recurring, the MS immediately restarts the collection.
- o The AS acks the event (B2), and in its business logic understands that the code '*1' means that the UAC wants its own volume to be lowered in the conference mix. The AS is able to associate the

event with the right UAC by referring to the attached dialogid (01d1b38). It then acts accordingly, by sending a <modifyjoin> (C1) which does exactly this: the provided <stream> child element instructs the MS to modify the volume of the UAC-->conference audio flow (setgain=-5dB sendonly). Notice that the "setgain" value is the absolute volume level; if the user's request is to lower the volume level, the AS must remember the previously set volume level and from it calculate the new volume level. Notice how the request also includes directives upon the inverse direction. This verbose approach is needed, since otherwise the MS would not only change the volume in the requested direction, but also disable the media flow in the other direction: having a proper <stream> addressing the UAC<--conf media flow as well ensures that this doesn't happen.

- o The MS successfully enforces the requested operation (C2), changing the volume.
- o A new matching DTMF string from UAC is collected (*9). As before, an event is triggered to the AS (D1).
- o The AS acks the event (D2), and matches the new code ('*9') with its related operation (raise the volume of the conference mix for UAC), taking the proper action. A different <modifyjoin> is sent (E1) with the new instructions (setgain=+3dB recvonly). The same considerations about how the incremental operation should be mapped to the command apply here as well. Also notice how a <stream> for the inverse direction (sendonly) is provided again just as a placeholder, which basically instructs the MS that the settings for that direction are not to be changed, maintaining the previous directives of (C1).
- o The MS successfully enforces this requested operation as well (E2), changing the volume in the specified direction.
- o At this point, a further matching DTMF string from UAC is collected (*6), and sent to the AS (F1).
- o After the required ack (F2), the AS reacts by implementing the action associated with the new code ('*6'), by which UAC requested to be muted within the conference. A new <modifyjoin> is sent (G1) with a properly constructed payload (setstate=mute sendonly), and the MS enforces it (G2).
- o A last (in the scenario) matching DTMF string is collected by the MS (*0). As with all the previous codes, this string is notified to the AS (H1).
- o The AS acks the event (H2), and understands the UAC wants to leave the conference now (since the code is *0). This means that a series of actions must be taken, namely:
 - * actually stopping the recurring collection, since it's not needed anymore;
 - * unjoin UAC from the conference it is in;

- * additional operations might be considered, e.g., a global announcement stating UAC is leaving, but are left out for the sake of conciseness);
the former is accomplished by means of a <dialogterminate> request (I1) to the IVR package (dialogid=01d1b38); the latter by means of an 'unjoin' request (K1) to the Mixer package instead.
- o The <dialogterminate> request is handled by the MS (I2), and the dialog is terminated successfully. As soon as the dialog has actually been terminated, a 'dialogexit' event is triggered as well to the AS (J1). This event has no report upon the result of the last iteration (since the dialog was terminated abruptly with an immediate=true) and is acked by the AS (J2) to finally complete the dialog lifetime.
- o The <unjoin> request, instead, is immediately enforced (K2). As a consequence of the unjoin operation, an 'unjoin-notify' event notification is triggered by the MS (L1) to confirm to the AS that the requested entities are not attached to each other anymore. The status in the event is set to 0 which, as stated in the specification, means the join has been terminated by an <unjoin> request. The ack of the AS (L2) concludes this scenario.

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

CFW 238e1f2946e8 CONTROL

Control-Package: msc-ivr/1.0

Content-Type: application/msc-ivr+xml

Content-Length: 809

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart connectionid="14849028:37fc2523">
    <dialog repeatCount="0">
      <collect maxdigits="2" termchar="A" escapekey="#">
        <grammar>
          <grammar version="1.0" mode="dtmf"
            xmlns="http://www.w3.org/2001/06/grammar">
            <rule id="digit">
              <one-of>
                <item>0</item>
                <item>1</item>
                <item>3</item>
                <item>6</item>
                <item>7</item>
                <item>9</item>
              </one-of>
            </rule>
            <rule id="action" scope="public">
```



```
        <example>*3</example>
        <one-of>
          <item>
            *
            <ruleref uri="#digit"/>
          </item>
        </one-of>
      </rule>
    </grammar>
  </grammar>
</collect>
</dialog>
<subscribe>
  <dtmfsub matchmode="collect"/>
</subscribe>
</dialogstart>
</mscivr>
```

A2. AS <- MS (CFW 200 OK)

```
CFW 238e1f2946e8 200
Timeout: 10
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="01d1b38"/>
</mscivr>
```

B1. AS <- MS (CFW CONTROL dtmfnotify event)

```
CFW 1dd62e043c00 CONTROL
Control-Package: msc-ivr/1.0
Content-Type: application/msc-ivr+xml
Content-Length: 180

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <event dialogid="01d1b38">
    <dtmfnotify matchmode="collect" dtmf="*1"
      timestamp="2008-12-17T17:20:53Z"/>
  </event>
</mscivr>
```

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

CFW 1dd62e043c00 200

C1. AS -> MS (CFW CONTROL, modifyjoin with setgain)

CFW 0216231b1f16 CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 290

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <modifyjoin id1="873975758:a5105056" id2="54b4ab3">
    <stream media="audio" direction="sendonly">
      <volume controltype="setgain" value="-5"/>
    </stream>
    <stream media="audio" direction="recvonly"/>
  </modifyjoin>
</mscmixer>
```

C2. AS <- MS (CFW 200 OK)

CFW 0216231b1f16 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>
```

D1. AS <- MS (CFW CONTROL dtmfnotify event)

CFW 4d674b3e0862 CONTROL

Control-Package: msc-ivr/1.0

Content-Type: application/msc-ivr+xml

Content-Length: 180

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <event dialogid="01d1b38">
    <dtmfnotify matchmode="collect" dtmf="*9"
      timestamp="2008-12-17T17:20:57Z"/>
  </event>
</mscivr>
```

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

CFW 4d674b3e0862 200

E1. AS -> MS (CFW CONTROL, modifyjoin with setgain)

CFW 140e0f763352 CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 292

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <modifyjoin id1="873975758:a5105056" id2="54b4ab3">
    <stream media="audio" direction="recvonly">
      <volume controltype="setgain" value="+3"/>
    </stream>
    <stream media="audio" direction="sendonly"/>
  </modifyjoin>
</mscmixer>
```

E2. AS <- MS (CFW 200 OK)

CFW 140e0f763352 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>
```

F1. AS <- MS (CFW CONTROL dtmfnotify event)

CFW 478ed6f1775b CONTROL

Control-Package: msc-ivr/1.0

Content-Type: application/msc-ivr+xml

Content-Length: 180

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <event dialogid="01d1b38">
    <dtmfnotify matchmode="collect" dtmf="*6"
      timestamp="2008-12-17T17:21:02Z"/>
  </event>
</mscivr>
```


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

CFW 478ed6f1775b 200

G1. AS -> MS (CFW CONTROL, modifyjoin with setstate)

CFW 7fdcc2331bef CONTROL
Control-Package: msc-mixer/1.0
Content-Type: application/msc-mixer+xml
Content-Length: 295

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
 <modifyjoin id1="873975758:a5105056" id2="54b4ab3">
 <stream media="audio" direction="sendonly">
 <volume controltype="setstate" value="mute"/>
 </stream>
 <stream media="audio" direction="recvonly"/>
 </modifyjoin>
</mscmixer>

G2. AS <- MS (CFW 200 OK)

CFW 7fdcc2331bef 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>

H1. AS <- MS (CFW CONTROL dtmfnotify event)

CFW 750b917a5d4a CONTROL
Control-Package: msc-ivr/1.0
Content-Type: application/msc-ivr+xml
Content-Length: 180

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
 <event dialogid="01d1b38">
 <dtmfnotify matchmode="collect" dtmf="*0"
 timestamp="2008-12-17T17:21:05Z"/>
 </event>
</mscivr>

H2. AS -> MS (CFW 200, ACK to 'CONTROL event')

CFW 750b917a5d4a 200

I1. AS -> MS (CFW CONTROL, dialogterminate)

CFW 515f007c5bd0 CONTROL
Control-Package: msc-ivr/1.0
Content-Type: application/msc-ivr+xml
Content-Length: 128

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
 <dialogterminate dialogid="01d1b38" immediate="true"/>
</mscivr>

I2. AS <- MS (CFW 200 OK)

CFW 515f007c5bd0 200
Timeout: 10
Content-Type: application/msc-ivr+xml
Content-Length: 140

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
 <response status="200" reason="Dialog terminated"
 dialogid="01d1b38"/>
</mscivr>

J1. AS <- MS (CFW CONTROL dialogexit event)

CFW 76adc41122c1 CONTROL
Control-Package: msc-ivr/1.0
Content-Type: application/msc-ivr+xml
Content-Length: 155

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
 <event dialogid="01d1b38">
 <dialogexit status="0" reason="Dialog terminated"/>
 </event>
</mscivr>

J2. AS -> MS (CFW 200, ACK to 'CONTROL event')

CFW 76adc41122c1 200

K1. AS -> MS (CFW CONTROL, unjoin)

CFW 4e6afb6625e4 CONTROL
Control-Package: msc-mixer/1.0
Content-Type: application/msc-mixer+xml
Content-Length: 127

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
 <unjoin id1="873975758:a5105056" id2="54b4ab3"/>
</mscmixer>

K2. AS <- MS (CFW 200 OK)

CFW 4e6afb6625e4 200
Timeout: 10
Content-Type: application/msc-mixer+xml
Content-Length: 122

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

L1. AS <- MS (CFW CONTROL unjoin-notify event)

CFW 577696293504 CONTROL
Control-Package: msc-mixer/1.0
Content-Type: application/msc-mixer+xml
Content-Length: 157

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
 <event>
 <unjoin-notify status="0"
 id1="873975758:a5105056" id2="54b4ab3"/>
 </event>
</mscmixer>

L2. AS -> MS (CFW 200, ACK to 'CONTROL event')

CFW 577696293504 200

7. Media Resource Brokering

All the flows presented so far describe the interaction between a single AS and a single MS. This is the most simple topology that can be envisaged in a MEDIACTRL-compliant architecture, but it's not the only allowed one. [\[RFC5567\]](#) presents several possible topologies, potentially involving several AS and several MS as well. To properly allow for such topologies, an additional element has been introduced in the MEDIACTRL architecture, called Media Resource Broker (MRB). Such an entity, and the protocols needed to interact with it, has been standardized in [\[RFC6917\]](#).

A MRB is basically a locator that is aware of a pool of MS, and makes them available to interested AS according to their requirements. For this reason, two different interfaces have been introduced:

- o Publishing Interface ([Section 7.1](#)), which allows a MRB to subscribe for notifications at the MS it is handling (e.g., available and occupied resources, current state, etc.);
- o Consumer Interface ([Section 7.2](#)), which allows an interested AS to query a MRB for a MS capable to fulfill its requirements.

The flows in the following sections will present some typical use case scenarios involving a MRB, and the different topologies it has been conceived to work in.

Additionally, a few considerations on the handling of media dialogs whenever a MRB is involved are presented in [Section 7.3](#).

7.1. Publishing Interface

A MRB makes use of the MS's publishing interface to acquire relevant information. This publishing interface, as specified in [\[RFC6917\]](#), is made available as a control package for the Media Control Channel Framework. This means that, in order to receive information from a MS, a MRB must negotiate a control channel as explained in [Section 5](#). This package allows a MRB to request information from a MS, be it as a direct request/answer or by subscribing for events.

Of course, considering the MRB is interested in the publishing interface, the already mentioned negotiation must be changed in order to take into account the need for the MRB control package. The name of this package is 'mrbs-publish/1.0', which means the SYNC might look like the following:

1. MRB -> MS (CFW SYNC)

CFW 6b8b4567327b SYNC

Dialog-ID: z9hG4bK-4542-1-0

Keep-Alive: 100

Packages: msc-ivr/1.0,msc-mixer/1.0,mrp-publish/1.0

2. MRB <- MS (CFW 200)

CFW 6b8b4567327b 200

Keep-Alive: 100

Packages: msc-ivr/1.0,msc-mixer/1.0,mrp-publish/1.0

Supported: msc-example-pkg/1.0

The meaning of this negotiation has already been presented. It is enough to point out that, in this case, the MRB adds a new item to the 'Packages' it needs support for (mrp-publish/1.0). In this case, the MS supports it, and in fact it is added to the negotiated packages in the reply:

Packages: msc-ivr/1.0,msc-mixer/1.0,mrp-publish/1.0

^^^^^^^^^^^^^^^^^^^^

The MS of [Section 5](#), instead, did not have support for that package, since only 'msc-example-pkg/1.0' was part of the 'Supported' list.

Figure 47 presents a ladder diagram of a typical interaction based on the MRB control package.

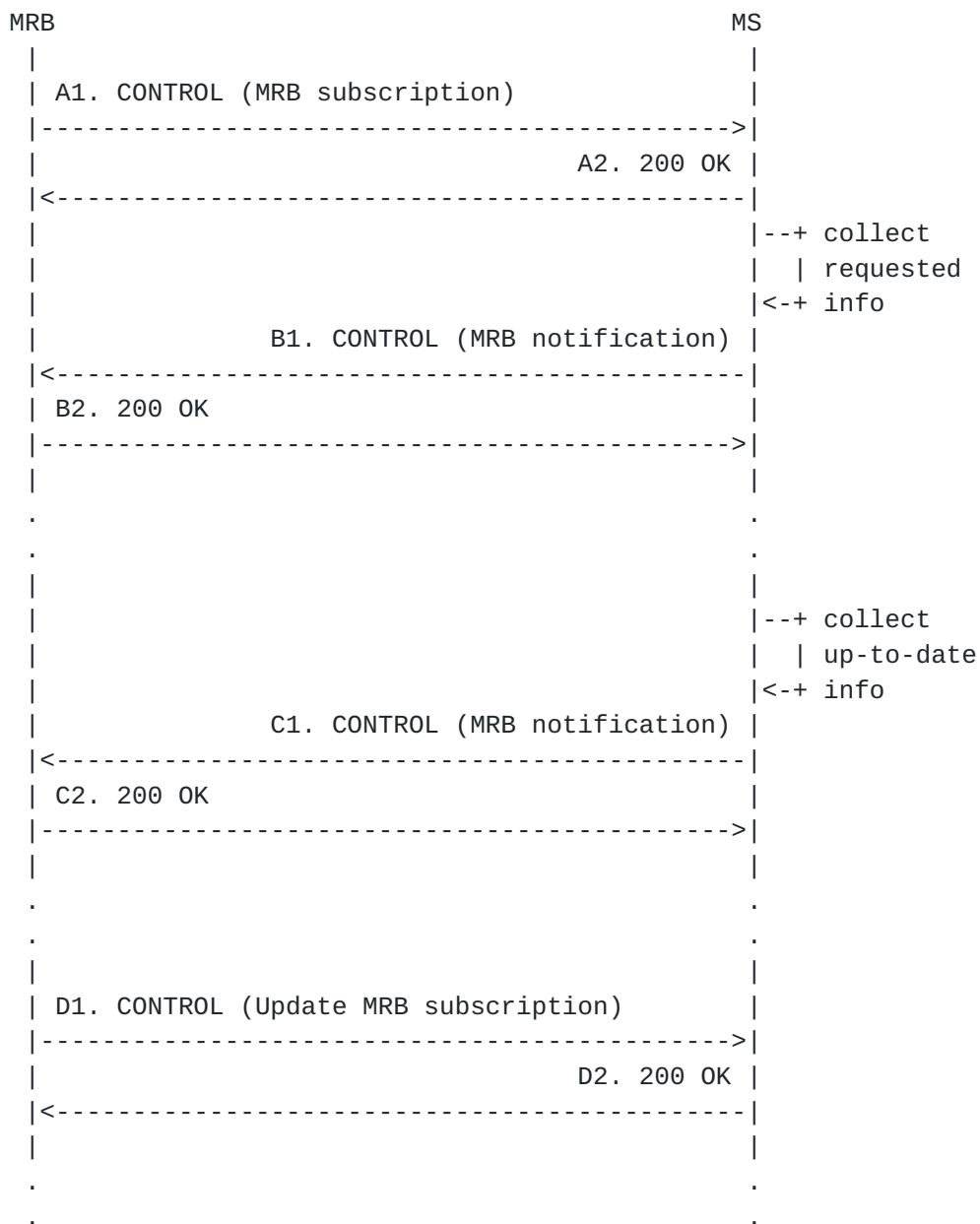


Figure 47: Media Resource Brokering: Subscription and Notification

In this example, the MRB subscribes for information at the specified MS, and events are triggered at a regular, negotiated, basis. All these messages flow through the control channel as all the messages in this document do. The framework transaction steps are the following:

- o The MRB sends a new CONTROL message (A1) addressed to the MRB package (mrbs-publish/1.0); it is a subscription for information (<subscription>), and the MRB is asking to be notified at least

every 10 minutes (<minfrequency>), or if required every 30 seconds at max; besides, the subscription must last 30 minutes (<expires>) after which no notification must be sent anymore;

- o The MS acknowledges the request (A2), and notifies the success of the request in a 200 OK message (<mrbresponse>);
- o The MS prepares and sends the first notification to the MRB (B1); as what happened with other packages as well, the notification has been sent as a MS-generated CONTROL message; it is a notification related to the request in the first message, considering the 'id' matches (p0T65U) that one; all the info the MRB subscribed for is provided in the payload;
- o the MRB acknowledges the notification (B2), and uses the retrieved info to update its information as part of its business logic;
- o the same happens at the required frequency, with up-to-date information;
- o after a while, the MRB updates its subscription (D1) to get more frequent updates (minfrequency=1, an update every second at least); the MS accepts the update (D2), even if it adjusts the frequency in the reply according to its policies (minfrequency=30, lower rate); the notifications keep on going, but at the newer frequency rate; the expiration is also updated accordingly (600 seconds again, since the update refreshes it).

A1. MRB -> MS (CONTROL, publish request)

CFW l1dc30BZ0biC CONTROL
Control-Package: mrb-publish/1.0
Content-Type: application/mrb-publish+xml
Content-Length: 337

```
<mrbpublish version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-publish">
  <mrbrequest>
    <subscription action="create" seqnumber="1" id="p0T65U">
      <expires>60</expires>
      <minfrequency>600</minfrequency>
      <maxfrequency>30</maxfrequency>
    </subscription>
  </mrbrequest>
</mrbpublish>
```

A2. MRB <- MS (200 to CONTROL, request accepted)

CFW l1dc30BZ0biC 200
Timeout: 10
Content-Type: application/mrb-publish+xml

Content-Length: 139

```
<mrbpublish version="1.0" xmlns="urn:ietf:params:xml:ns:mrbs-publish">
  <mrbsresponse status="200" reason="OK: Request accepted"/>
</mrbpublish>
```

B1. MRB <- MS (CONTROL, event notification from MS)

CFW 03fff52e7b7a CONTROL

Control-Package: mrbs-publish/1.0

Content-Type: application/mrbs-publish+xml

Content-Length: 4157

```
<mrbpublish version="1.0"
  xmlns="urn:ietf:params:xml:ns:mrbs-publish">
  <mrbsnotification seqnumber="1" id="p0T65U">
    <media-server-id>a1b2c3d4</media-server-id>
    <supported-packages>
      <package name="msc-ivr/1.0"/>
      <package name="msc-mixer/1.0"/>
      <package name="mrbs-publish/1.0"/>
      <package name="msc-example-pkg/1.0"/>
    </supported-packages>
    <active-rtp-sessions>
      <rtp-codec name="audio/basic">
        <decoding>10</decoding>
        <encoding>20</encoding>
      </rtp-codec>
    </active-rtp-sessions>
    <active-mixer-sessions>
      <active-mix conferenceid="7cfigs43">
        <rtp-codec name="audio/basic">
          <decoding>3</decoding>
          <encoding>3</encoding>
        </rtp-codec>
      </active-mix>
    </active-mixer-sessions>
    <non-active-rtp-sessions>
      <rtp-codec name="audio/basic">
        <decoding>50</decoding>
        <encoding>40</encoding>
      </rtp-codec>
    </non-active-rtp-sessions>
    <non-active-mixer-sessions>
      <non-active-mix available="15">
        <rtp-codec name="audio/basic">
```



```
        <decoding>15</decoding>
        <encoding>15</encoding>
    </rtp-codec>
</non-active-mix>
</non-active-mixer-sessions>
<media-server-status>active</media-server-status>
<supported-codecs>
    <supported-codec name="audio/basic">
        <supported-codec-package name="msc-ivr/1.0">
            <supported-action>encoding</supported-action>
            <supported-action>decoding</supported-action>
        </supported-codec-package>
        <supported-codec-package name="msc-mixer/1.0">
            <supported-action>encoding</supported-action>
            <supported-action>decoding</supported-action>
        </supported-codec-package>
    </supported-codec>
</supported-codecs>
<application-data>TestbedPrototype</application-data>
<file-formats>
    <supported-format name="audio/x-wav">
        <supported-file-package>
            msc-ivr/1.0
        </supported-file-package>
    </supported-format>
</file-formats>
<max-prepared-duration>
    <max-time max-time-seconds="3600">
        <max-time-package>msc-ivr/1.0</max-time-package>
    </max-time>
</max-prepared-duration>
<dtmf-support>
    <detect>
        <dtmf-type package="msc-ivr/1.0" name="RFC4733"/>
        <dtmf-type package="msc-mixer/1.0" name="RFC4733"/>
    </detect>
    <generate>
        <dtmf-type package="msc-ivr/1.0" name="RFC4733"/>
        <dtmf-type package="msc-mixer/1.0" name="RFC4733"/>
    </generate>
    <passthrough>
        <dtmf-type package="msc-ivr/1.0" name="RFC4733"/>
        <dtmf-type package="msc-mixer/1.0" name="RFC4733"/>
    </passthrough>
</dtmf-support>
<mixing-modes>
    <audio-mixing-modes>
        <audio-mixing-mode package="msc-ivr/1.0"> \
```



```
        nbest \
    </audio-mixing-mode>
</audio-mixing-modes>
<video-mixing-modes activespeakermix="true" vas="true">
    <video-mixing-mode package="msc-mixer/1.0"> \
        single-view \
    </video-mixing-mode>
    <video-mixing-mode package="msc-mixer/1.0"> \
        dual-view \
    </video-mixing-mode>
    <video-mixing-mode package="msc-mixer/1.0"> \
        dual-view-crop \
    </video-mixing-mode>
    <video-mixing-mode package="msc-mixer/1.0"> \
        dual-view-2x1 \
    </video-mixing-mode>
    <video-mixing-mode package="msc-mixer/1.0"> \
        dual-view-2x1-crop \
    </video-mixing-mode>
    <video-mixing-mode package="msc-mixer/1.0"> \
        quad-view \
    </video-mixing-mode>
    <video-mixing-mode package="msc-mixer/1.0"> \
        multiple-5x1 \
    </video-mixing-mode>
    <video-mixing-mode package="msc-mixer/1.0"> \
        multiple-3x3 \
    </video-mixing-mode>
    <video-mixing-mode package="msc-mixer/1.0"> \
        multiple-4x4 \
    </video-mixing-mode>
</video-mixing-modes>
</mixing-modes>
<supported-tones>
    <supported-country-codes>
        <country-code package="msc-ivr/1.0">GB</country-code>
        <country-code package="msc-ivr/1.0">IT</country-code>
        <country-code package="msc-ivr/1.0">US</country-code>
    </supported-country-codes>
    <supported-h248-codes>
        <h248-code package="msc-ivr/1.0">cg/*</h248-code>
        <h248-code package="msc-ivr/1.0">biztn/ofque</h248-code>
        <h248-code package="msc-ivr/1.0">biztn/erwt</h248-code>
        <h248-code package="msc-mixer/1.0">conftn/*</h248-code>
    </supported-h248-codes>
</supported-tones>
<file-transfer-modes>
    <file-transfer-mode package="msc-ivr/1.0" name="HTTP"/>
```



```

</file-transfer-modes>
<asr-tts-support>
  <asr-support>
    <language xml:lang="en"/>
  </asr-support>
  <tts-support>
    <language xml:lang="en"/>
  </tts-support>
</asr-tts-support>
<vxml-support>
  <vxml-mode package="msc-ivr/1.0" support="rfc6231"/>
</vxml-support>
<media-server-location>
  <civicAddress xml:lang="it">
    <country>IT</country>
    <A1>Campania</A1>
    <A3>Napoli</A3>
    <A6>Via Claudio</A6>
    <HNO>21</HNO>
    <LMK>University of Napoli Federico II</LMK>
    <NAM>Dipartimento di Informatica e Sistemistica</NAM>
    <PC>80210</PC>
  </civicAddress>
</media-server-location>
<label>TestbedPrototype-01</label>
<media-server-address>
  sip:MediaServer@ms.example.net
</media-server-address>
<encryption/>
</mrnotification>
</mrbpublish>

```

B2. MRB -> MS (200 to CONTROL)

CFW 03fff52e7b7a 200

(C1 and C2 omitted for brevity)

D1. MRB -> MS (CONTROL, publish request)

CFW pyu788fc32wa CONTROL
Control-Package: mrb-publish/1.0

Content-Type: application/mrb-publish+xml

Content-Length: 342

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbpublish version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-publish">
  <mrbrequest>
    <subscription action="update" seqnumber="2" id="p0T65U">
      <expires>600</expires>
      <minfrequency>1</minfrequency>
    </subscription>
  </mrbrequest>
</mrbpublish>
```

D2. MRB <- MS (200 to CONTROL, request accepted)

CFW pyu788fc32wa 200

Timeout: 10

Content-Type: application/mrb-publish+xml

Content-Length: 332

```
<mrbpublish version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-publish">
  <mrbresponse status="200" reason="OK: Request accepted">
    <subscription action="create" seqnumber="2" id="p0T65U">
      <expires>600</expires>
      <minfrequency>30</minfrequency>
    </subscription>
  </mrbresponse>
</mrbpublish>
```

7.2. Consumer Interface

While the Publishing interface is used by a MS to publish its functionality and up-to-date information to a MRB, the Consumer interface is used by an interested AS to get access to a resource. An AS can make use of the Consumer interface to contact a MRB and describe the resources it needs: the MRB then replies with the needed information, specifically the address of an MS that is capable to meet the requirements.

However, unlike the Publishing interface, the Consumer interface is not specified as a Control Package. It is, instead, conceived as an XML-based protocol that can be transported by means of either HTTP or SIP, as it will be shown in the following sections.

As specified in [[RFC6917](#)], the Consumer interface can be involved in

basically two topologies: a Query mode and an Inline mode. In the Query mode ([Section 7.2.1](#)), the Consumer requests and responses are conveyed by means of HTTP: once the AS gets the answer, the usual MEDIACTRL interactions occur between the AS and the MS chosen by the MRB. In the Inline mode, instead, the MRB is in the path between the AS and the pool of MS it is handling. In this case, an AS can place Consumer requests using SIP as a transport, by means of a multipart payload ([Section 7.2.2](#)) containing the Consumer request itself and an SDP related to either the creation of a control channel or to a UAC media dialog. This is called Inline-aware mode, since it assumes that the interested AS knows a MRB is in place and knows how to talk to it. Anyway, the MRB is also conceived to work with AS that are unaware of its functionality, i.e., which are not aware of the Consumer interface: in this kind of scenario, the Inline mode is still used, but with the AS thinking the MRB it is talking to is actually an MS. This approach is called Inline-unaware mode ([Section 7.2.3](#)).

[7.2.1](#). Query Mode

As anticipated in the previous section, in the Query mode the AS sends Consumer requests by means of HTTP. Specifically, an HTTP POST is used to convey the request. The MRB is assumed to send its response by means of an HTTP 200 OK reply. Since a successful Consumer response contains information to contact a specific MS (the MS the MRB has deemed best capable to fulfill the AS's requirements), an AS can subsequently directly contact the MS as already described in the previous sections of the document. This means that, in the Query mode, the MRB acts purely as a locator, and then the AS and the MS can talk 1:1.

Figure 48 presents a ladder diagram of a typical Consumer request in the Query topology:

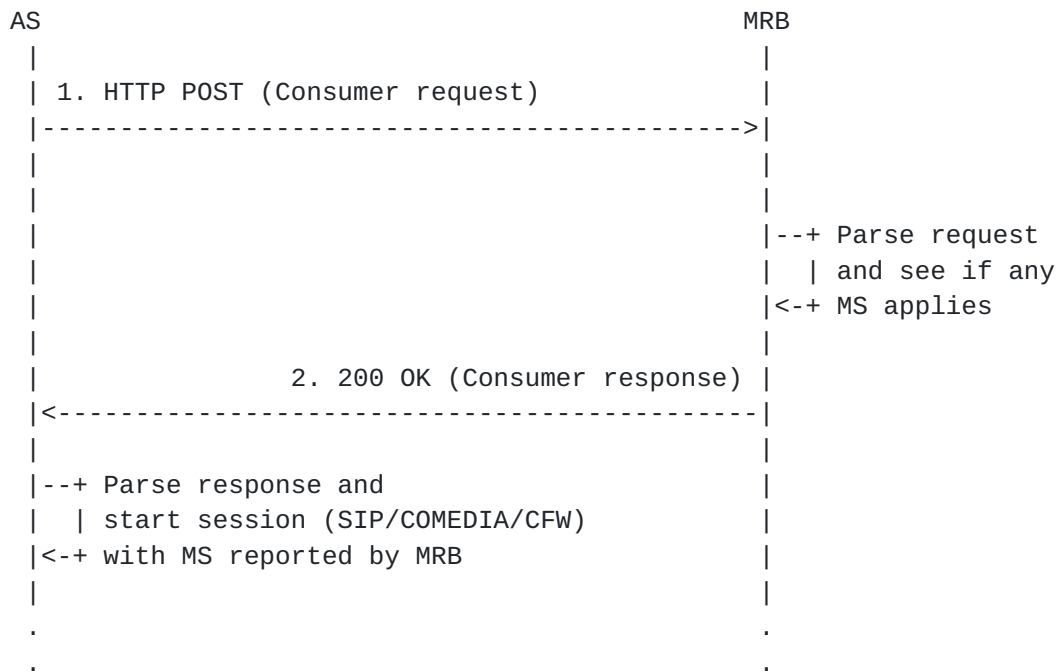


Figure 48: Media Resource Brokering: Query Mode

In this example, the AS is interested in an MS meeting a defined set of requirements:

1. it must support both the IVR and Mixer packages;
2. it must provide at least 10 G.711 encoding/decoding RTP sessions for IVR purposes;
3. it must support HTTP-based streaming and support for the audio/x-wav file format in the IVR package.

These requirements are properly formatted according to the MRB Consumer syntax. The framework transaction steps are the following:

- o The AS sends an HTTP POST message to the MRB (1); the payload is, of course, the Consumer request, which is reflected by the Content-Type header (application/mrb-consumer+xml); the Consumer request (<mediaResourceRequest>, uniquely identified by its 'id' attribute set to the random value 'n3un93wd'), includes some general requirements (<generalInfo>) and some IVR-specific requirements (<ivrInfo>); the general part of the requests contains the set of required packages (<packages>); the IVR-specific section, instead, contains requirements concerning the number of required IVR sessions (<ivr-sessions>), the file formats that are to be supported (<file-formats>) and the required file transfer capabilities (<file-transfer-modes>);

- o the MRB gets the request and parses it; then, according to its business logic, it realizes it can't find a single MS capable of targeting the request, and as a consequence picks two MS which can handle respectively 60 and 40 of the requested sessions; it prepares a Consumer response (2) to provide the AS with the requested information; the response (<mediaResourceResponse>, which includes the same 'id' attribute as the request) is a success (status=200), and includes the relevant information (<response-session-info>); specifically, the response includes transaction-related information (the same session-id and seq provided by the AS in its request, to allow proper request/response matching) together with information on the duration of the reservation (expires=3600, after an hour the request will expire) and the SIP addresses of the chosen MS.

Notice how the sequence number the MRB returned is not 1: according to the MRB specification, this is the starting value to increment for the sequence number to be used in subsequent requests. This means that, should the AS want to update or remove the session, it should use 10 as a value for the sequence number in the related request. This random value for the first sequence number is, according to the MRB security considerations, also a means to help preventing a malicious entity to mess with or disrupt another AS session with the MRB: in fact, sequence number in requests and responses have to match, and a failure to provide the correct sequence number would result in a failure and a 405 error message.

1. AS -> MRB (HTTP POST, Consumer request)

```
-----
POST /Mrb/Consumer HTTP/1.1
Content-Length: 893
Content-Type: application/mrb-consumer+xml
Host: mrb.example.com:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.0.1 (java 1.5)
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbconsumer version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-consumer">
  <mediaResourceRequest id="n3un93wd">
    <generalInfo>
      <packages>
        <package>msc-ivr/1.0</package>
        <package>msc-mixer/1.0</package>
      </packages>
    </generalInfo>
    <ivrInfo>
```



```
<ivr-sessions>
  <rtp-codec name="audio/basic">
    <decoding>100</decoding>
    <encoding>100</encoding>
  </rtp-codec>
</ivr-sessions>
<file-formats>
  <required-format name="audio/x-wav"/>
</file-formats>
<file-transfer-modes>
  <file-transfer-mode package="msc-ivr/1.0" name="HTTP"/>
</file-transfer-modes>
</ivrInfo>
</mediaResourceRequest>
</mrbcconsumer>
```

2. AS <- MRB (200 to POST, Consumer response)

```
-----
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5
Server: Sun GlassFish Communications Server 1.5
Content-Type: application/mrb-consumer+xml; charset=ISO-8859-1
Content-Length: 1146
Date: Thu, 28 Jul 2011 10:34:45 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbcconsumer version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-consumer">
  <mediaResourceResponse reason="Resource found" status="200"
    id="n3un93wd">
    <response-session-info>
      <session-id>z603G3yaUzM8</session-id>
      <seq>9</seq>
      <expires>3600</expires>
      <media-server-address
        uri="sip:MediaServer@ms.example.com:5080">
        <ivr-sessions>
          <rtp-codec name="audio/basic">
            <decoding>60</decoding>
            <encoding>60</encoding>
          </rtp-codec>
        </ivr-sessions>
      </media-server-address>
      <media-server-address
        uri="sip:OtherMediaServer@pool.example.net:5080">
        <ivr-sessions>
          <rtp-codec name="audio/basic">
            <decoding>40</decoding>
```



```
        <encoding>40</encoding>
      </rtp-codec>
    </ivr-sessions>
  </media-server-address>
</response-session-info>
</mediaResourceResponse>
</mrbcconsumer>
```

For the sake of conciseness, the subsequent steps are not presented. They are, however, very trivial, since they basically consist in the AS issuing a COMEDIA negotiation with either of the obtained MS, as already presented in the first part of the document. The same can be said with respect to attaching UAC media dialogs: in fact, since after the Query the AS<->MS interaction becomes 1:1, UAC media dialogs can be redirected directly to the proper MS using the 3PCC approach, e.g., as in Figure 10.

7.2.2. Inline-aware Mode

Unlike the Query mode, in the Inline-aware mode (IAMM) the AS sends Consumer requests by means of SIP. Of course, saying that the transport changes from HTTP to SIP is not as trivial as it seems. In fact, HTTP and SIP behave in a very different way, and this is reflected in the way the Inline-aware mode is conceived.

An AS willing to issue a Consumer request by means of SIP, has to do so by means of an INVITE. As specified in [\[RFC6917\]](#), the payload of the INVITE can't only contain the Consumer request itself: in fact, the Consumer request is assumed to be carried within a SIP transaction. Besides, a Consumer session is not strictly associated with the lifetime of any SIP transaction, meaning Consumer requests belonging to the same session may be transported over different SIP messages. An hangup on any of these SIP dialogs would not affect the Consumer session by itself.

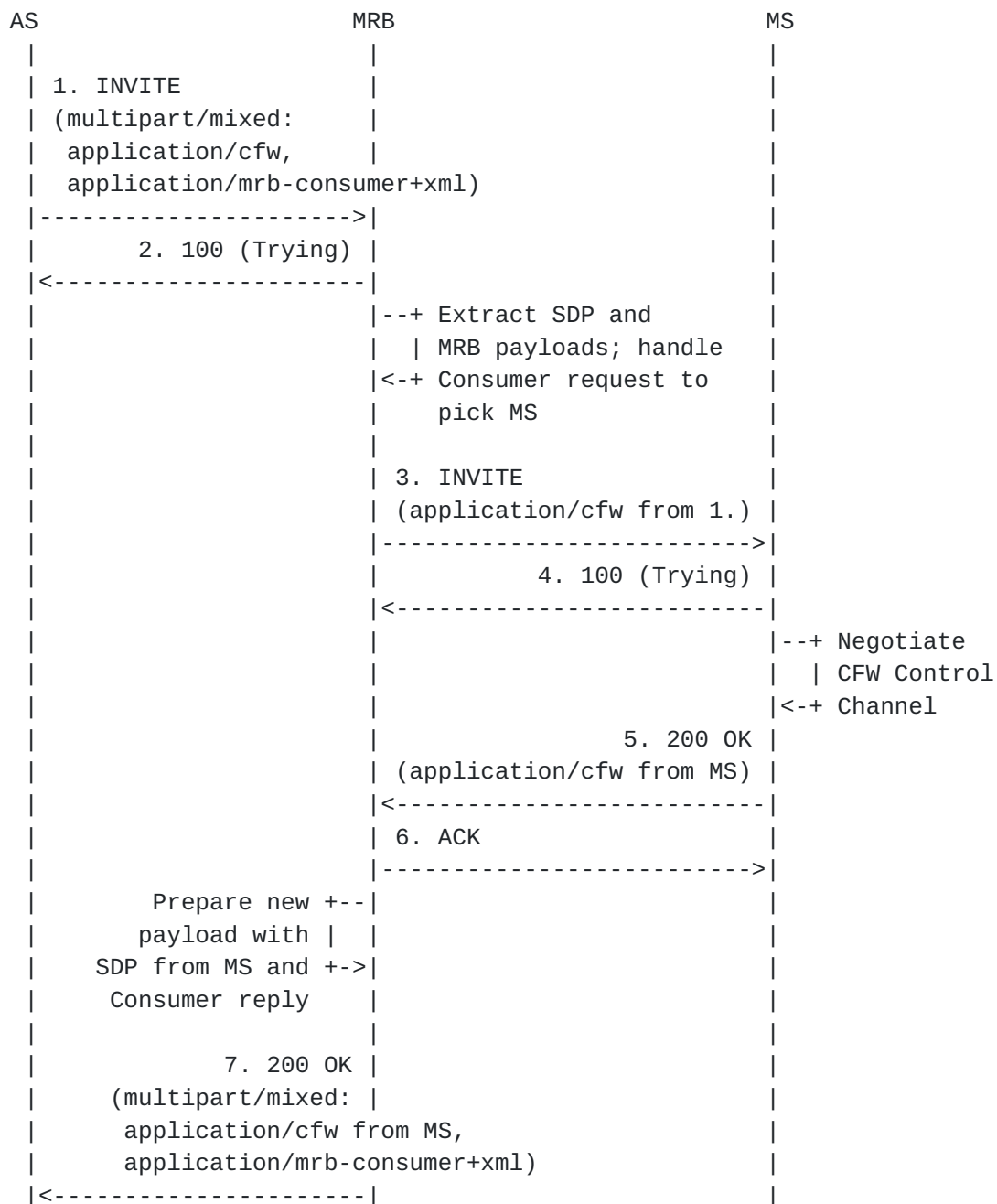
That said, as documented in [\[RFC6230\]](#), [\[RFC6917\]](#) envisages two kinds of SIP dialogs a Consumer request may be sent over: a SIP control dialog (a SIP dialog the AS sends in order to set up a control channel) or a UAC media dialog (a SIP dialog the AS sends to attach a UAC to a MS). In both cases, the AS would prepare a multipart/mixed payload to achieve both ends, i.e., receiving a reply to its Consumer request and effectively carrying on the negotiation described in the SDP payload.

The behaviours in the two cases, which are called respectively CFW-based and Media dialog-based approach, are only slightly different, but both will be presented to clarify how they could be exploited.

To make things clearer for the reader, the same consumer request as the one presented in the Query mode will be sent, in order to clarify how the behaviour of the involved parties may differ.

7.2.2.1. Inline-aware Mode: CFW-based approach

Figure 49 presents a ladder diagram of a typical Consumer request in the CFW-based Inline-aware topology:




```

| 8. ACK |
|----->|
|
|--+ Read Cons. reply |
| | and use SDP to |
|<--+ create CFW Chn. |
|
|
|<<##### TCP CONNECTION #####>>|
|
| CFW SYNC |
|+++++++>|
|
. . .
. . .

```

Figure 49: Media Resource Brokering: CFW-based Inline-aware Mode

As anticipated, to make the understanding of the scenario easier we assume the AS is interested in exactly the same set of requirements as presented in [Section 7.2.1](#). This means that the Consumer request originated by the AS will be the same as before, with only the transport/topology changing.

Please note that, to ease the reading of the protocol contents, a simple 'Part' is used whenever a boundary for a 'multipart/mixed' payload is provided, instead of the actual boundary that would be inserted in the SIP messages.

The framework transaction steps (for simplicity only the payloads and not the complete SIP transactions are reported) are the following:

1. AS -> MRB (INVITE multipart/mixed)

```

-----
[.]
Content-Type: multipart/mixed;boundary="Part"

--Part
Content-Type: application/sdp

v=0
o=- 2890844526 2890842807 IN IP4 as.example.com
s=MediaCtrl
c=IN IP4 as.example.com
t=0 0

```



```
m=application 48035 TCP cfw
a=connection:new
a=setup:active
a=cfw-id:vF0zD4xzUAW9
```

--Part

Content-Type: application/mrb-consumer+xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbconsumer version="1.0"
  xmlns="urn:ietf:params:xml:ns:mrb-consumer">
  <mediaResourceRequest id="fr34asx1">
    <generalInfo>
      <packages>
        <package>msc-ivr/1.0</package>
        <package>msc-mixer/1.0</package>
      </packages>
    </generalInfo>
    <ivrInfo>
      <ivr-sessions>
        <rtp-codec name="audio/basic">
          <decoding>100</decoding>
          <encoding>100</encoding>
        </rtp-codec>
      </ivr-sessions>
      <file-formats>
        <required-format name="audio/x-wav"/>
      </file-formats>
      <file-transfer-modes>
        <file-transfer-mode package="msc-ivr/1.0" name="HTTP"/>
      </file-transfer-modes>
    </ivrInfo>
  </mediaResourceRequest>
</mrbconsumer>
```

--Part

3. MRB -> MS (INVITE SDP only)

[...]

Content-Type: application/sdp

```
v=0
o=- 2890844526 2890842807 IN IP4 as.example.com
s=MediaCtrl
c=IN IP4 as.example.com
```



```
t=0 0
m=application 48035 TCP cfw
a=connection:new
a=setup:active
a=cfw-id:vF0zD4xzUAW9
```

5. MRB <- MS (200 OK SDP)

```
-----
[...]  
Content-Type: application/sdp  
  
v=0  
o=lminiero 2890844526 2890842808 IN IP4 ms.example.net  
s=MediaCtrl  
c=IN IP4 ms.example.net  
t=0 0  
m=application 7575 TCP cfw  
a=connection:new  
a=setup:passive  
a=cfw-id:vF0zD4xzUAW9
```

7. AS <- MRB (200 OK multipart/mixed)

```
-----
[...]  
Content-Type: multipart/mixed;boundary="Part"  
  
--Part  
Content-Type: application/sdp  
  
v=0  
o=lminiero 2890844526 2890842808 IN IP4 ms.example.net  
s=MediaCtrl  
c=IN IP4 ms.example.net  
t=0 0  
m=application 7575 TCP cfw  
a=connection:new  
a=setup:passive  
a=cfw-id:vF0zD4xzUAW9  
  
--Part  
Content-Type: application/mrb-consumer+xml  
  
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<mrbconsumer version="1.0"
```



```
xmlns="urn:ietf:params:xml:ns:mrbc-consumer">
<mediaResourceResponse reason="Resource found" status="200"
    id="fr34asx1">
  <response-session-info>
    <session-id>z603G3yaUzM8</session-id>
    <seq>9</seq>
    <expires>3600</expires>
    <media-server-address
      uri="sip:MediaServer@ms.example.com:5080">
      <connection-id>32pbdxZ8:KQw677BF</connection-id>
      <ivr-sessions>
        <rtp-codec name="audio/basic">
          <decoding>60</decoding>
          <encoding>60</encoding>
        </rtp-codec>
      </ivr-sessions>
    </media-server-address>
    <media-server-address
      uri="sip:OtherMediaServer@pool.example.net:5080">
      <ivr-sessions>
        <rtp-codec name="audio/basic">
          <decoding>40</decoding>
          <encoding>40</encoding>
        </rtp-codec>
      </ivr-sessions>
    </media-server-address>
  </response-session-info>
</mediaResourceResponse>
</mrbcconsumer>

--Part
```

The sequence diagram and the dumps effectively show the different approach with respect with the Query mode: the SIP INVITE the AS sends (1.) includes both a Consumer request (the same as before), and an SDP to negotiate a CFW channel with a MS. The MRB takes care of the request exactly as before (provisioning two MS instances) but with a remarkable difference: first of all it picks one of the two MS on behalf of the AS (negotiating the control channel in steps 3 to 6) and only then replies to the AS with both the MS-side of the SDP negotiation (with info on how to set up the control channel) and the Consumer response itself.

The Consumer response is also slightly different by itself: in fact, as it can be seen in 7., there's an additional element (<connection-id>) the MRB has added to the message. This element

contains the 'connection-id' the AS and MS would have build out of the From and To tags as explained in the previous sections, had the AS contacted the MS directly: since the MRB has actually done the negotiation on the AS behalf, without this information the AS and MS would refer to different connectionid attributes to target the same dialog, thus causing the CFW protocol not to behave as expected. This aspect will be more carefully described in the next section, for the media dialog-based approach, since the 'connection-id' attribute is strictly related to media sessions.

For the sake of conciseness, the following steps are not presented. Anyway, they are quite trivial: in fact, as shown in the flow, the SIP negotiation has resulted in both the AS and the chosen MS negotiating a Control Channel. This means that the AS is only left to instantiate the Control Channel and sending CFW requests according to its application logic.

Besides, it is worthwhile to highlight the fact that, as in the Query example, the AS gets the addresses of both the chosen MS in this example as well, since a Consumer transaction has taken place. This means that, just as in the Query case, any UAC media dialog can be redirected directly to the proper MS using the 3PCC approach, e.g., as in Figure 10, rather than using the MRB again as a Proxy/B2BUA. Of course, a separate SIP control dialog would be needed before attempting to use the second MS instance.

7.2.2.2. Inline-aware Mode: Media dialog-based approach

As anticipated, there's a second way to take advantage of the IAMM mode, that is exploiting SIP dialogs related to UAC media dialogs as 'vessels' for Consumer messages. As it will be clearer in the sequence diagram and protocol dumps, the scenario does not differ much from the one presented in [Section 7.2.2.1](#) with respect to the Consumer request/response, but a dedicated paragraph may be useful in order to understand how they may differ with respect to the management of the media dialog itself and any CFW control channel that may be involved.

Figure 50 presents a ladder diagram of a typical Consumer request in the media dialog-based Inline-aware topology:




```

| 2. 100 Trying |
|<-----|
|
| 3. INVITE
| (multipart/mixed:
|   audio/video from 1.,
|   application/mrb-consumer+xml)
|----->
| 4. 100 (Trying) |
|<-----|
|
|                                     |--+ Extract SDP and
|                                     | MRB payloads; handle
|<--+ Consumer request to
|    pick Media Servers
|
| 5. INVITE
| (audio/video from 3.)
|----->
| 6. 100 (Trying) |
|<-----|
|                                     +--|
|                                     | Handle media dialog |
|                                     | (connection-id) +->|
|
|                                     7. 200 OK
|                                     | (audio/video from MS)
|<-----|
| 8. ACK
|----->
|
| Prepare new +--|
| payload with |
| SDP from MS and +->|
| Consumer reply
|
| 9. 200 OK
| (multipart/mixed:
|   audio/video from MS,
|   application/mrb-consumer+xml)
|<-----|
| 10. ACK
|----->
|
| |--+ Read Cons. reply
|   | and send SDP
|<--+ back to UAC
|
| 11. 200 OK
| (audio/video from MS)
|<-----|
| 12. ACK
|

```

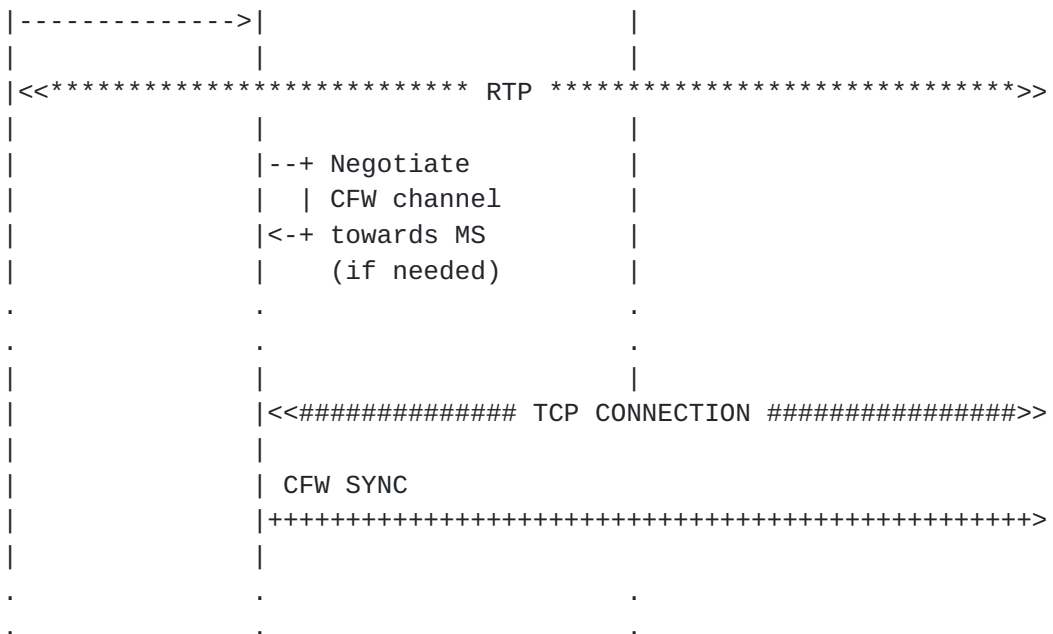



Figure 50: Media Resource Brokering: Media dialog-based Inline-aware Mode

As anticipated, to make the understanding of the scenario easier we assume the AS is interested in exactly the same set of requirements as presented in [Section 7.2.1](#). This means that the Consumer request originated by the AS will be the same as before, with only the transport/topology changing.

Again, please note that, to ease the reading of the protocol contents, a simple 'Part' is used whenever a boundary for a 'multipart/mixed' payload is provided, instead of the actual boundary that would be inserted in the SIP messages.

The framework transaction steps (for simplicity only the the relevant headers and payloads, and not the complete SIP transactions, are reported) are the following:

1. UAC -> AS (INVITE with media SDP)

```

-----
[.]
From: <sip:lminiero@users.example.com>;tag=1153573888
To: <sip:mediactrlDemo@as.example.com>
[.]
Content-Type: application/sdp

```



```
v=0
o=lminiero 123456 654321 IN IP4 203.0.113.2
s=A conversation
c=IN IP4 203.0.113.2
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
```

3. AS -> MRB (INVITE multipart/mixed)

```
-----
[...]  
From: <sip:ApplicationServer@as.example.com>;tag=fd4fush5  
To: <sip:Mrb@mrbs.example.org>  
[...]  
Content-Type: multipart/mixed;boundary="Part"
```

```
--Part  
Content-Type: application/sdp
```

```
v=0
o=lminiero 123456 654321 IN IP4 203.0.113.2
s=A conversation
c=IN IP4 203.0.113.2
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
```

```
--Part  
Content-Type: application/mrb-consumer+xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<mrbconsumer version="1.0"  
  xmlns="urn:ietf:params:xml:ns:mrb-consumer">  
  <mediaResourceRequest id="bnv3xc45">  
    <generalInfo>  
      <packages>
```



```

        <package>msc-ivr/1.0</package>
        <package>msc-mixer/1.0</package>
    </packages>
</generalInfo>
<ivrInfo>
    <ivr-sessions>
        <rtp-codec name="audio/basic">
            <decoding>100</decoding>
            <encoding>100</encoding>
        </rtp-codec>
    </ivr-sessions>
    <file-formats>
        <required-format name="audio/x-wav"/>
    </file-formats>
    <file-transfer-modes>
        <file-transfer-mode package="msc-ivr/1.0" name="HTTP"/>
    </file-transfer-modes>
</ivrInfo>
</mediaResourceRequest>
</mrbcconsumer>

--Part

```

5. MRB -> MS (INVITE SDP only)

```

[.]
From: <sip:Mrb@mrbc.example.org:5060>;tag=32pbdxZ8
To: <sip:MediaServer@ms.example.com:5080>
[.]
Content-Type: application/sdp

v=0
o=lminiero 123456 654321 IN IP4 203.0.113.2
s=A conversation
c=IN IP4 203.0.113.2
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

```


7. MRB <- MS (200 OK SDP)

[...]
From: <sip:Mrb@mr.example.org:5060>;tag=32pbdxZ8
To: <sip:MediaServer@ms.example.com:5080>;tag=KQw677BF
[...]
Content-Type: application/sdp

v=0
o=lminiero 123456 654322 IN IP4 203.0.113.1
s=MediaCtrl
c=IN IP4 203.0.113.1
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

9. AS <- MRB (200 OK multipart/mixed)

[...]
From: <sip:ApplicationServer@as.example.com>;tag=fd4fush5
To: <sip:Mrb@mr.example.org>;tag=117652221
[...]
Content-Type: multipart/mixed;boundary="Part"

--Part
Content-Type: application/sdp

v=0
o=lminiero 123456 654322 IN IP4 203.0.113.1
s=MediaCtrl
c=IN IP4 203.0.113.1
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
```

--Part

Content-Type: application/mrb-consumer+xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbconsumer version="1.0"
  xmlns="urn:ietf:params:xml:ns:mrb-consumer" >
  <mediaResourceResponse reason="Resource found" status="200"
    id="bnv3xc45">
    <response-session-info>
      <session-id>z1skKYZQ3eFu</session-id>
      <seq>9</seq>
      <expires>3600</expires>
      <media-server-address
        uri="sip:MediaServer@ms.example.com:5080">
        <connection-id>32pbdxZ8:KQw677BF</connection-id>
        <ivr-sessions>
          <rtp-codec name="audio/basic">
            <decoding>60</decoding>
            <encoding>60</encoding>
          </rtp-codec>
        </ivr-sessions>
      </media-server-address>
      <media-server-address
        uri="sip:OtherMediaServer@pool.example.net:5080">
        <ivr-sessions>
          <rtp-codec name="audio/basic">
            <decoding>40</decoding>
            <encoding>40</encoding>
          </rtp-codec>
        </ivr-sessions>
      </media-server-address>
    </response-session-info>
  </mediaResourceResponse>
</mrbconsumer>
```

--Part

11. UAC <- AS (200 OK SDP)

```
-----
[...]  
From: <sip:lminiero@users.example.com>;tag=1153573888  
To: <sip:mediactrlDemo@as.example.com>;tag=bcd47c32  
[...]  
Content-Type: application/sdp  
  
v=0  
o=lminiero 123456 654322 IN IP4 203.0.113.1  
s=MediaCtrl  
c=IN IP4 203.0.113.1  
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
```

The first obvious difference is that the first INVITE (1.) is not originated by the AS itself (willing to set up a control channel in the previous example) but by an authorized UAC (e.g., to take advantage of a media service provided by the AS). As such, the first INVITE only contains an SDP to negotiate an audio and video channel. The AS in its business logic needs to attach this UAC to a MS according to some specific requirements (e.g., the called URI is associated to a specific service), and as such prepares a Consumer request to be sent to the MRB in order to obtain a valid MS for the purpose: as before, the Consumer request is sent together with the SDP to the MRB (3.). The MRB extracts the Consumer payload and takes care of it as usual: it picks two MS instances, and attaches the UAC to the first one (5.). Once the MS has successfully negotiated the audio and video streams (7.), the MRB takes note of the 'connection-id' associated with this call (which will be needed afterwards in order to manipulate the audio and video streams for this user) and sends back to the AS both the SDP returned by the MS and the Consumer response (9.). The AS extracts the Consumer response and takes note of both the MS instances it has been given and the connection-id information: it then completes the scenario by sending back to the UAC the MS returned by the MS (11.).

At this point, the UAC has successfully been attached to a MS. The AS only needs to set up a control channel to that MS, if needed; this step may not be required, especially if the Consumer request is an update to an existing session rather than the preparation of a new one. Assuming a control channel towards that MS doesn't exist yet, the AS creates it as usual, by sending an INVITE directly to the MS it has the address of. Once done with that, it can start manipulating the audio and video streams of the UAC: to do so, it refers to the 'connection-id' element as reported by the MRB, rather than relying on the one it is aware of. In fact, the AS is aware of a connection-id (fd4fush5:117652221, built out of the messages exchanged with the MRB), while the MS is aware of another one (32pbdxZ8:KQw677BF, built out of the MRB-MS interaction). The right one is of course the one the MS is aware of, and as such the AS refers to that one, which the MRB added to the Consumer response just for the purpose.

7.2.3. Inline-unaware Mode

While in the Inline-aware mode the AS knows it is sending an INVITE to a MRB and not to a MS, and acts accordingly (using the multipart/mixed payload to query for a MS able to fulfill its requirements) in the Inline-unaware mode (IUMM) it is not. This means that a MRB-unaware AS having access to a MRB talks to it as if it were a generic MEDIACTRL MS: i.e., AS negotiates a Control Channel directly with the MRB, and attaches its media dialogs there as well. Of course, considering the MRB doesn't provide any MS functionality by itself, it must act as a Proxy/B2BUA between the AS and a MS for what concerns both the Control Channel Dialog and the Media Dialogs. According to implementation or deployment choices, simple redirects could also be exploited for the purpose.

The problem is that, without any Consumer request being placed by the MRB-unaware AS, the MRB can't rely on AS-originated directives to pick one MS rather than another. In fact, the MRB can't know what the AS is looking for. The MRB is then assumed to pick one according to its logic, which is implementation specific.

Figure 51 presents a ladder diagram of a typical Consumer request in the Inline-unaware topology:

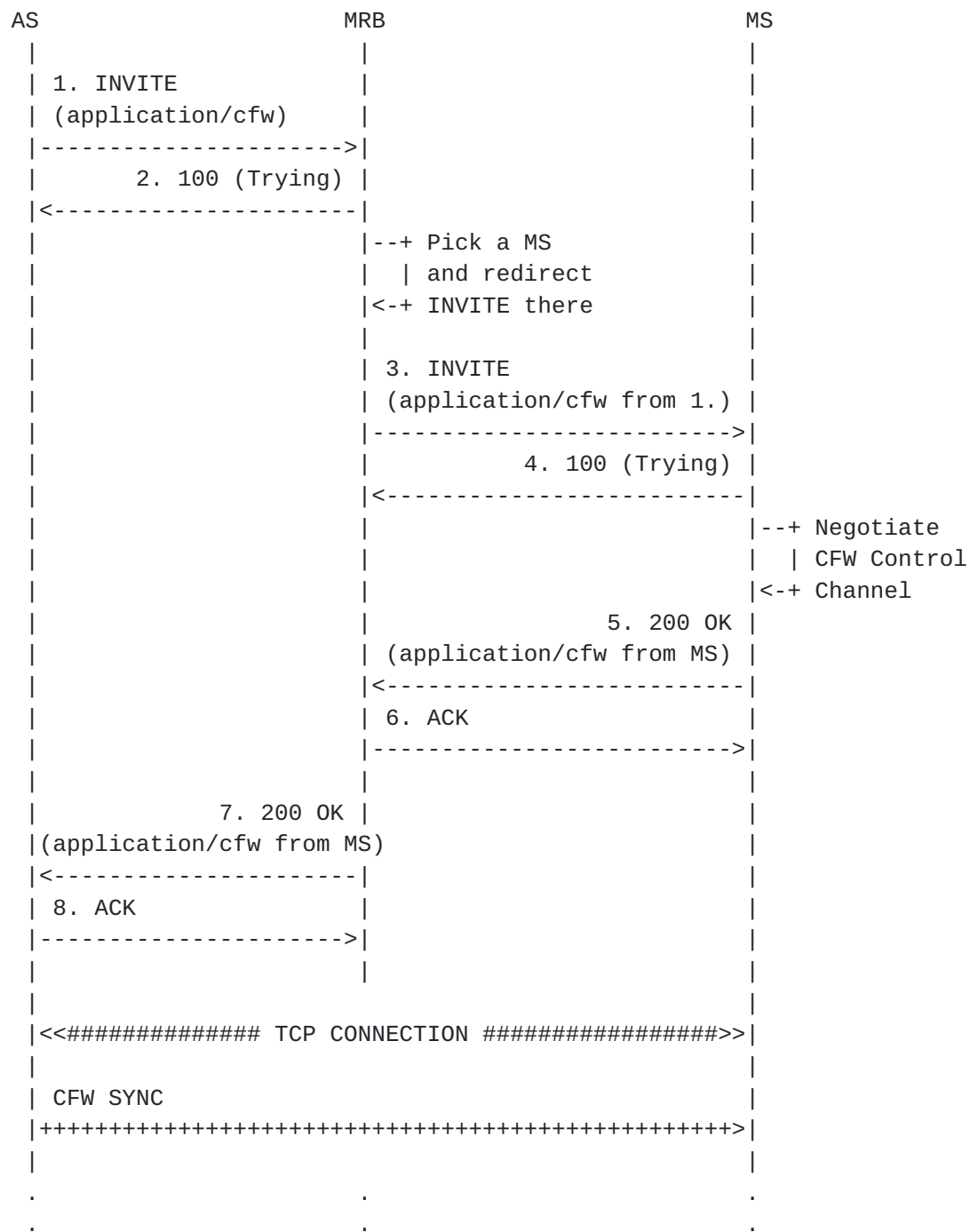


Figure 51: Media Resource Brokering: Inline-unaware Mode

As it can be evinced from the diagram, in this topology the MRB basically acts as a 3PCC between the AS and the chosen MS.

The same can be said with respect to attaching UAC media dialogs. The MRB remembers the MS it has chosen for the AS and, for every UAC

media dialog the AS tries to attach to the MRB, it makes sure it is actually redirected to the MS somehow.

No content for the presented messages is provided in this section, as in the IUMM mode no Consumer transaction is involved. In this example, a simple [[RFC6230](#)] Control Channel negotiation occurs where the MRB acts as an intermediary, that is, picking a MS for the AS according to some logic. In this case, in fact, the AS does not support the MRB specification, and so just tries to set up a control channel the way it knows.

It is worth pointing out that the MRB may actually enforce its decision about the MS to grant to the AS in different ways: specifically, the sentence "redirect the INVITE" that is used in Figure 51 does not necessarily mean that a SIP 302 message should be used for the purpose. A simple way to achieve this may be provisioning the unaware AS with different URIs, all actually transparently handled by the MRB itself: this would allow the MRB to simply map those URIs to different MS instances. Besides, the SIP 'Contact' header may also be used by the MRB in a reply to an INVITE coming from an AS to provide the actual URI the chosen MS might be reached on. A motivation for such a discussion and more details about this are provided in [Section 7.3.2](#).

[7.3.](#) Handling media dialogs

It is worthwhile to spend a few words to address how media dialogs would be managed whenever a MRB is involved in the scenarios. In fact, the presence of a MRB may introduce an additional complexity compared to the quite straightforward 1:1 AS-MS topology.

[7.3.1.](#) Query and Inline-aware mode

Normally, especially in the Query and IAMM case, the MRB would only handle Consumer requests by an AS, and after that the AS and the Media Server picked by the MRB for a specific request would talk directly to each other by means of SIP. This is made possible by the fact that the AS gets the MS SIP URI in reply to its request. In this case, an AS can simply relay media dialogs associated with that session to the right MS to have them handled accordingly. Of course, in order for this to work it is assumed the AS creates a control channel to a chosen MS before it has any requests to service.

An example of such scenario is presented in Figure 52. Please notice that this diagram, as the ones that will follow in this section, is simplified with respect to the actual protocol interactions: for instance, the whole SIP transactions are not presented, and only the originating messages are presented in order to clarify the scenario

in a simple way.

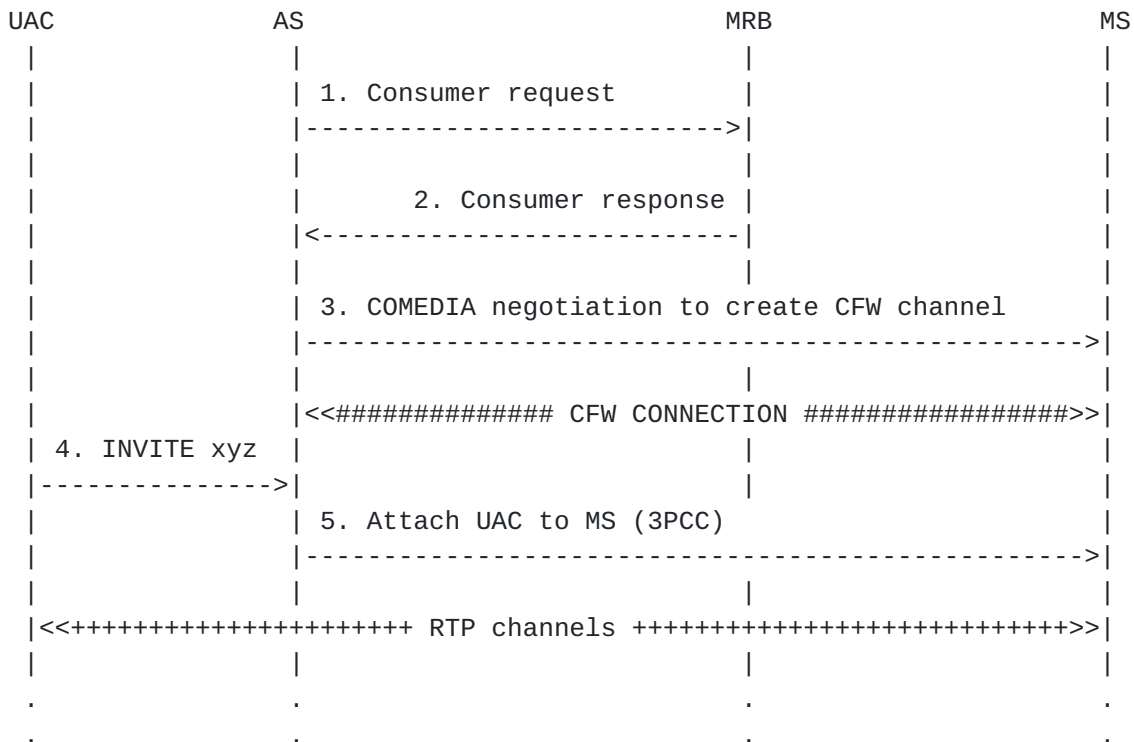


Figure 52: Handling media dialogs in Query/IAMM

As it can be evinced by looking at the diagram, the interactions among the components is quite straightforward: the AS knows which MS it has been assigned to (as a consequence of the MRB Consumer Request, whether it has been achieved by means of HTTP or SIP) and so it can easily attach any UAC accessing its functionality to the MS itself, and manipulate its media connections by making use of the CFW control channel as usual.

In such a scenario, the MRB is only involved as a locator: once the MRB provides the AS with the URI to the required resource, it doesn't interfere with the following interactions, if not for monitoring (e.g., by exploiting the Publishing information reported by the MS). As a consequence, the scenario basically becomes 1:1 between the AS and the MS again.

Nevertheless, there are cases when having a MRB in the SIP signalling path as well might be a desired feature: e.g., for more control over the use of the resources. Considering how the Consumer interface has been envisaged, this feature is easily achievable, with no change on

the protocol required at all. Specifically, in order to achieve such a functionality, in response to a Consumer request the MRB may reply with, instead of the MS SIP URI as before, a URI it is responsible for, and map it with the actual MS URI in its business logic, transparently to the AS itself. This way the AS would interact with the MRB as if it were the MS itself.

With respect to the previous figure, Figure 53 shows how the scenario would change in that case.

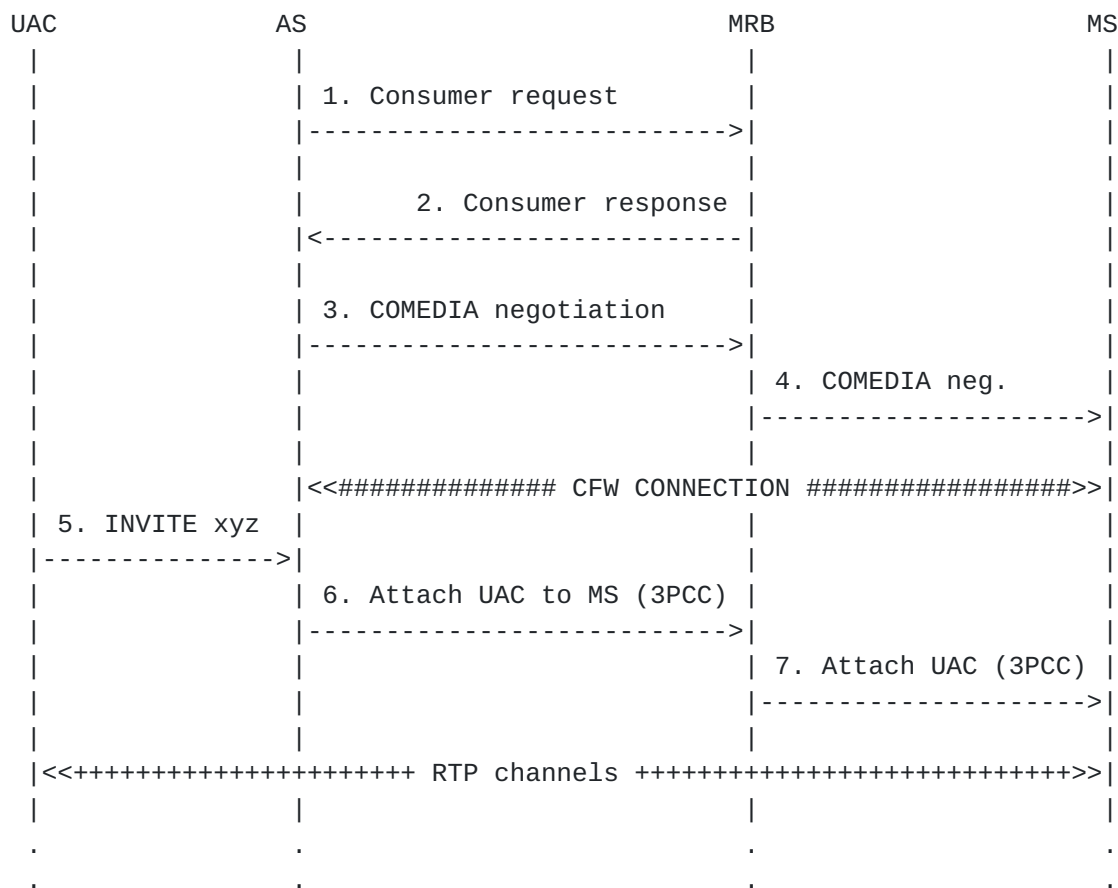


Figure 53: Handling media dialogs in Query/IAMM: MRB in the signalling path

This time, even though the MRB has picked a specific MS after a request from an AS, it replies with another SIP URI, an URI it would reply to itself: the AS would contact that URI in order to negotiate the control channel, and the MRB would proxy/forward the request to the actual MS transparently. Eventually, the control channel would be instantiated between the AS and the MS. The same happens for UACs

handled by the AS: the AS would forward the calls to the URI it has been provided with, the one handled by the MRB, which would in turn relay the call to the MS in order to have the proper RTP channels created between the UAC and the MS.

This scenario is not very different from the previous one: what changes is that the MRB is now on the signalling path for both the SIP control dialog and the SIP media dialogs, allowing it to have more control on the resources (e.g., triggering a BYE if a resource has expired). There are several possible approaches a MRB might take to allocate URIs to map with a requested MS. An example might be making use of SIP URI parameters to generate multiple SIP URIs that are unique but which all route to the same host and port, e.g., sip:MrbToMs@mr.example.com:5080;p=1234567890. Alternatively, the MRB might simply allocate a pool of URIs it would be responsible of, and manage the associations with the requested MS services accordingly.

7.3.2. Inline-unaware mode

As mentioned previously, in the IUMM case the AS would interact with the MRB as if it were the MS itself. One might argue that this would make the AS act as in the IAMM case: nevertheless, this is not the case, considering there the AS actually provided the MRB with information about the resources it required, leading to a proper MS to be picked, while in the IUMM case the MRB would have to pick a MS with no help from the AS at all.

That said, the IUMM case is also very interesting with respect to the media dialog management. In fact, in the MRB-unaware mode there would be no Consumer request and an AS would actually see the MRB as an MS. This means that, unlike the previous scenarios, being there no AS<->MRB interaction and as such no MS selection process the MRB would likely be in the signaling path anyway, at least when the AS first shows up. The MRB could either redirect the AS to a MS directly or transparently act a proxy/B2BUA and contact a MS (according to implementation-specific policies) on behalf of the unaware AS.

While apparently not a problem, this raises an issue when the same unaware AS has several sessions with different MS: the AS would only see one "virtual" MS (the MRB) and so it would relay all calls there, making it hard for the MRB to understand where these media dialogs should belong: specifically, whether the UAC calling belongs to the AS application logic leading to MS1 or MS2, or wherever else.

One possible, very simple approach to take care of this issue, is to always relay the SIP dialogs from the same unaware AS to the same MS.

It is depicted in Figure 54.

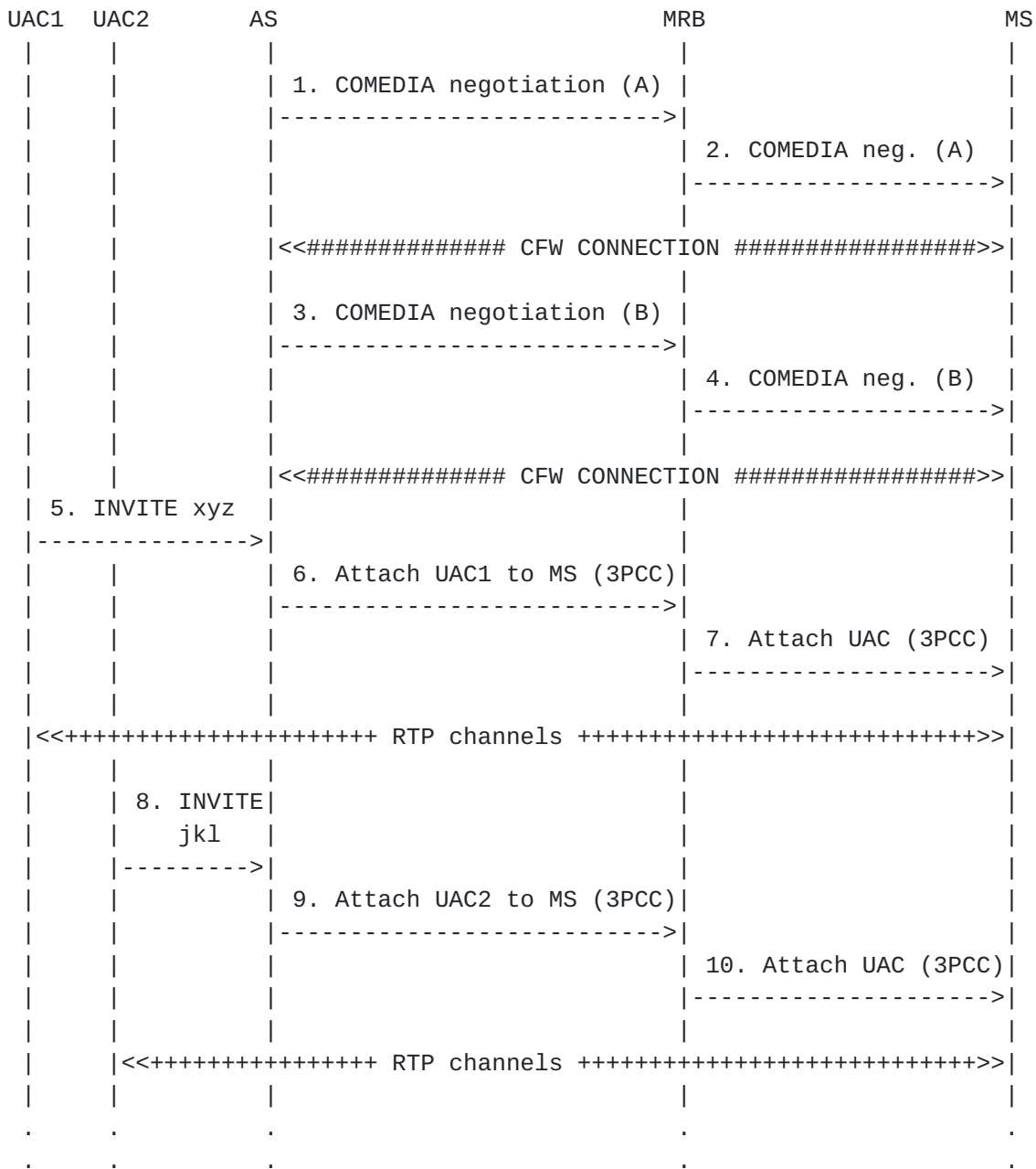


Figure 54: Handling media dialogs in IUMM: always the same MS

In this example, the AS creates two different Control Channels sessions (A and B) to address two different business logic implementations: e.g., the AS SIP URI 'xyz' (associated with CFW session A) may be an IVR pizza ordering application, while the AS SIP

URI 'jkl' (associated with CFW session B) may be associated with a conference room. It's quite clear, then, that if the MRB forwarded the two CFW sessions to two different MS, the handling of UAC media dialogs would prove troublesome, considering the MRB would have a hard way figuring out whether UAC1 should be attached to the MS managing CFW session A or the MS managing CFW session B. Forwarding all CFW sessions and UAC media dialogs coming from the same MRB-unaware AS to the same MS would instead work as expected: the MRB would in fact leave the mapping of media dialogs and CFW sessions up to the AS.

This approach, while very simple and indeed not very scalable, would actually help take care of the issue: in fact, no matter how many separate control channels the AS might have with the MRB/MS (in this example, the control channel A would be mapped to application xyz, and B to application jkl), the termination point would still always be the same MS, which would consequently be the destination for all media dialogs as well.

To overcome the scalability limitations of such an approach, though, at least in regard to the MRB being in the SIP signalling path for all calls, a different approach needs to be exploited. In fact, especially in the case of different applications handled by the same unaware AS, it makes sense to try and exploit different MS for the purpose, and correctly track media dialogs being forwarded accordingly. This means that the MRB must find a way to somehow redirect the unaware AS to different MS when it predicts or realizes that a different application logic is being involved.

To do so, the MRB might make use of different approaches. One of them is by making use of redirection, e.g., by means of a SIP 302 message in reply to a control channel negotiation originated by an unaware AS. Such an approach is depicted in Figure 55.

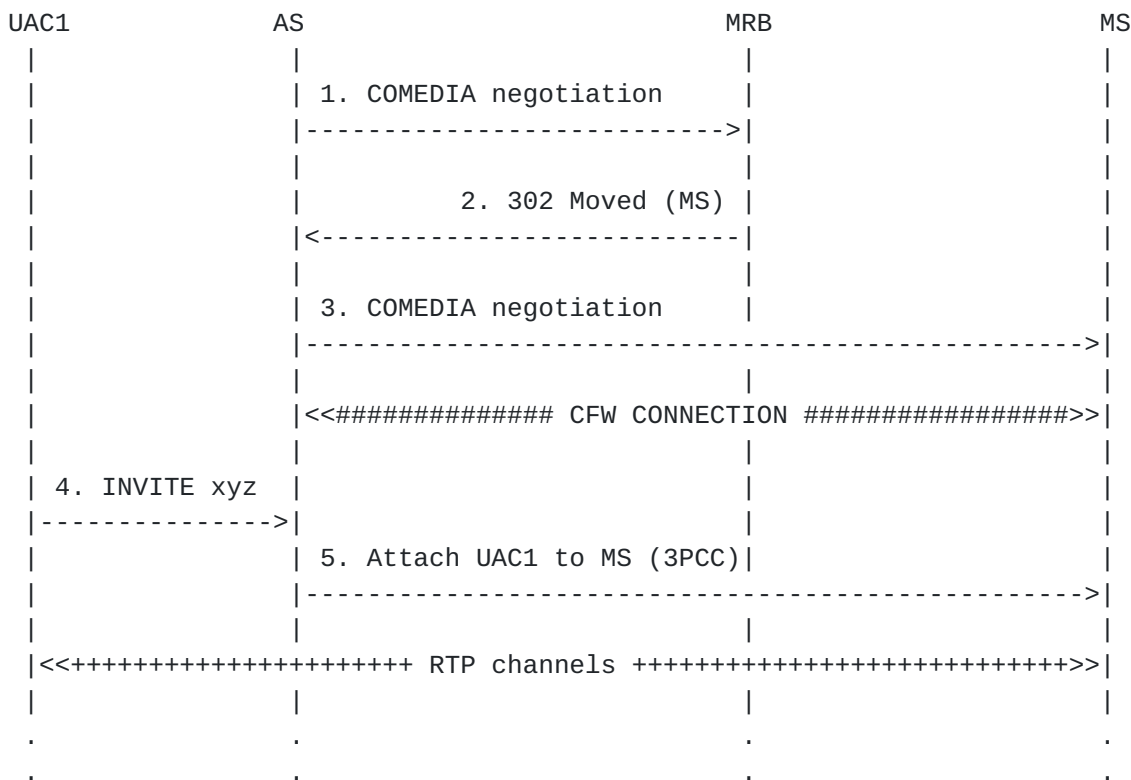


Figure 55: Handling media dialogs in IUMM: redirection

With this approach, the MRB might redirect the AS to a specific MS whenever a new control channel is to be created, and as a consequence the AS would redirect the related calls there. This is similar to the first approach of the Query/IAMM case, with the difference that no Consumer request would be involved. The scenario would again fallback to a 1:1 topology between the AS and the MS, making the interactions quite simple.

Just as before, anyway, the MRB might be interested in being in the signalling path for the SIP dialogs, instead of just acting as a locator. A third potential approach could be implementing the "virtual" URIs handled by the MRB, as described in the previous section. Rather than resorting to explicit redirection, or always using the same MS, the MRB may redirect new SIP control dialogs to one of its own URIs, using the same approach previously presented in Figure 53. Such an approach applied to the IUMM case is depicted in Figure 56.

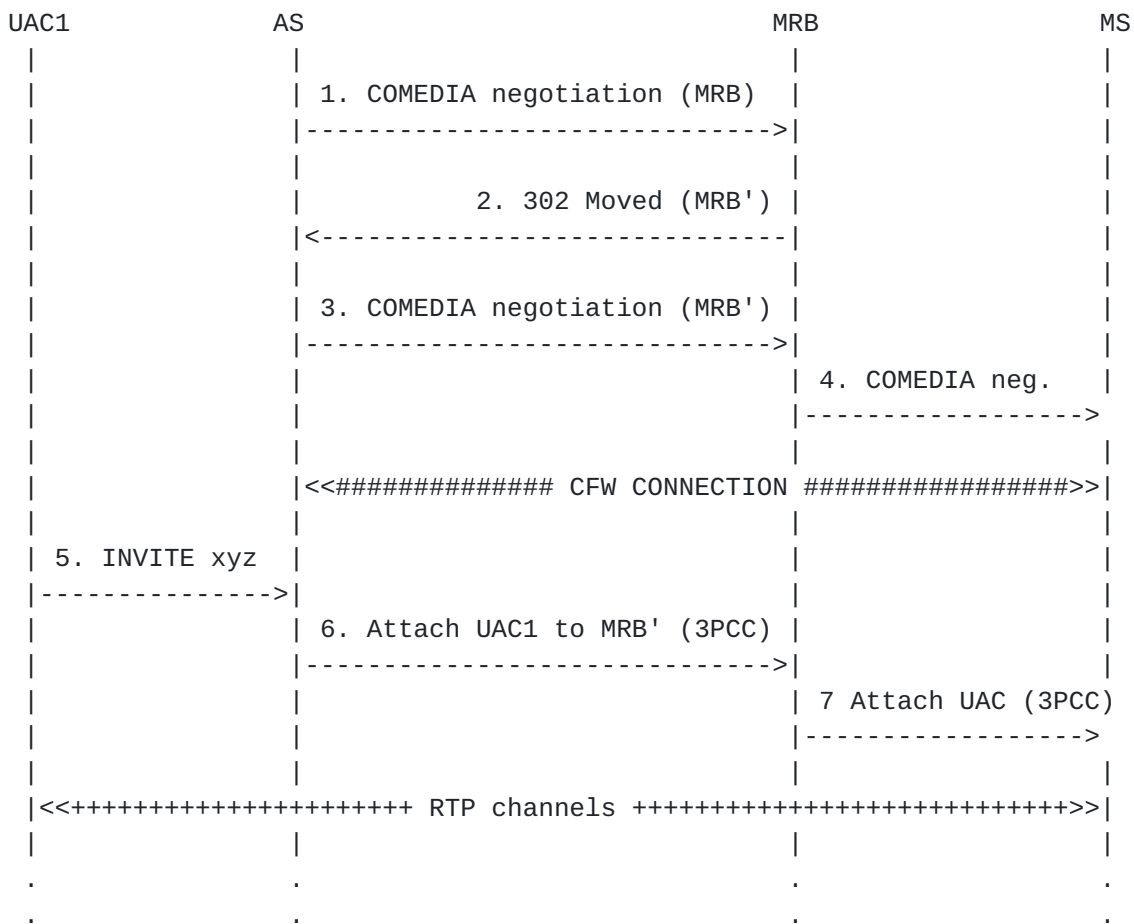


Figure 56: Handling media dialogs in IUMM: MRB in the signalling path

It is worth pointing out that in both cases, though, that there are scenarios where there could be no assurance that the 302 sent by the MRB would be seen by the AS: in fact, should a proxy be between the AS and the MRB, such a proxy could act on the 302 itself. To properly cope with such an issue, the MRB might also make use of the 'Contact' header in the SIP responses to the INVITE to address the right MS: while the AS is not required to use the info in such a header to reach the MS, it could be reasonable to exploit it for the purpose as it would take care of the proxy scenario mentioned above.

To conclude, there is a further approach a MRB might try to exploit to take care of the IUMM case. Since, as explained before, the issues related to the IUMM case mostly relate to the fact that the MRB is seen as a single MS instance by the AS, a simple way to overcome this might be to make the MRB look as a set of different MS right away: this can be done by simply provisioning the unaware AS with a series of different URIs, all handled by the MRB itself acting as a pool of "virtual" MS. This way, the AS may be designed to use

different MS for different classes of call, e.g., for different applications it is managing (two in the example presented in this section), and as such would contact two different of the provisioned URIs to create two distinct control channels towards two different MS. Considering both the URIs would be handled by the MRB, the MRB can use them to determine which MS each call should be directed to. Expanding on Figure 54 by removing the constraint to always use the same MS, this new scenario might look as depicted in Figure 57.

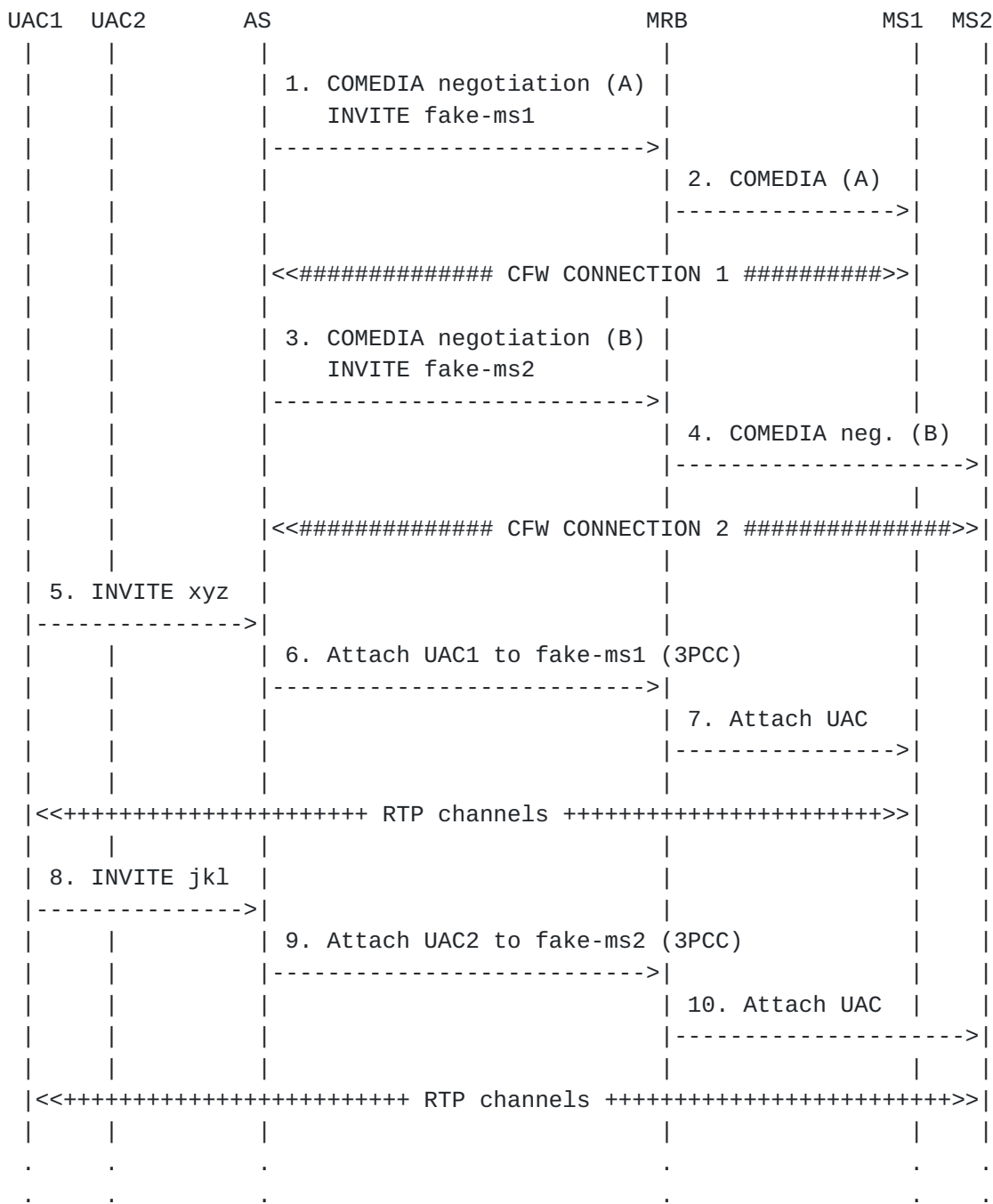


Figure 57: Handling media dialogs in IUMM: provisioned URIs

In this new example, we still assume that the same unaware AS is handling two different applications, still associated with the same URIs as before. This time, though, we also assume the AS has been designed to try and make use of different MS instances to handle the two very different applications it is responsible for. Besides, we also assume it has been configured to be able to use two different

MS, respectively reachable at SIP URI 'fake-ms1' and 'fake-ms2', both actually handled by the MRB transparently. This results, just as before, in two different control channels (A and B) being created, but towards two different MS: specifically, the MRB makes sure that, for this AS, the control channel negotiation towards 'fake-ms1' is actually redirected to MS1; at the same time, 'fake-ms2' is associated with MS2. Once the AS has set up the control channels with both the MS, it is ready to handle media dialogs. UAC1 calls the SIP URI 'xyz' on the AS to order a pizza: the AS attaches the media dialog to the MS it knows is responsible for that branch of application logic, that is 'fake-ms1'; the MRB in turn makes sure it reaches the right MS instance, MS1. Later on, a different user, UAC2, calls SIP URI 'jkl' to join a conference room: this time the AS attaches this new media dialog to the MS instance handling the conference application, that is 'fake-ms2'; again, the MRB makes sure it is actually MS2 to receive the dialog.

Again, this diagram is only meant to describe how the MRB might enforce its decisions: Just as described in the previous examples, the MRB may choose to either act as a proxy/B2BUA in between the AS and the MS instances, or redirect the AS to the right MS instances when they're first contacted (e.g., by means of the Contact header and/or a SIP redirect as explained before) and let the AS attach the media dialogs by itself.

7.3.3. CFW Protocol Bhaviour

As shown in the previous diagrams, no matter what the topology, the AS and MS usually end up with a direct connection with respect to the CFW control channel. As such, it can be expected that the CFW protocol continue to work as it should, and as a consequence all the call flows presented in this document can easily be reproduced in those circumstances as well.

One aspect needs to be taken in very good care, nevertheless. It's worthwhile to remind that both the AS and the MS make use of some SIP-related information to address the entities they manipulate. It's the case, for instance, of the 'connectionid' element both the AS and the MS refer to when addressing a specific UAC: this 'connectionid', as explained at the beginning of this draft, is constructed by concatenating the From: and To: tags extracted from a SIP header, specifically from the headers of the AS<->MS leg that allows an UAC to be attached to the MS. The presence of an additional component in the path between the AS and the MS, the MRB, might alter these tags, thus causing the AS to make use of tags (AS<->MRB) different than the ones used by the MS (MRB<->MS). This would result in the AS and MS using different 'connectionid' identifiers to address actually the same UAC, thus preventing the protocol to work as expected. As a

consequence, it's very important that any MRB implementation take very good care of preserving the integrity of the involved SIP headers when proxying/forwarding SIP dialogs between AS and MS, in order not to break the protocol behaviour.

Let's take, for instance, the scenario depicted in Figure 53, especially steps 6 and 7 which specifically address an UAC being attached by an AS to a MS via the MRB. Let's assume that what is presented in Figure 58 is what happens to the From: and To: headers when dealing with the 3PCC approach to attach a specific UAC to the MS in that case.

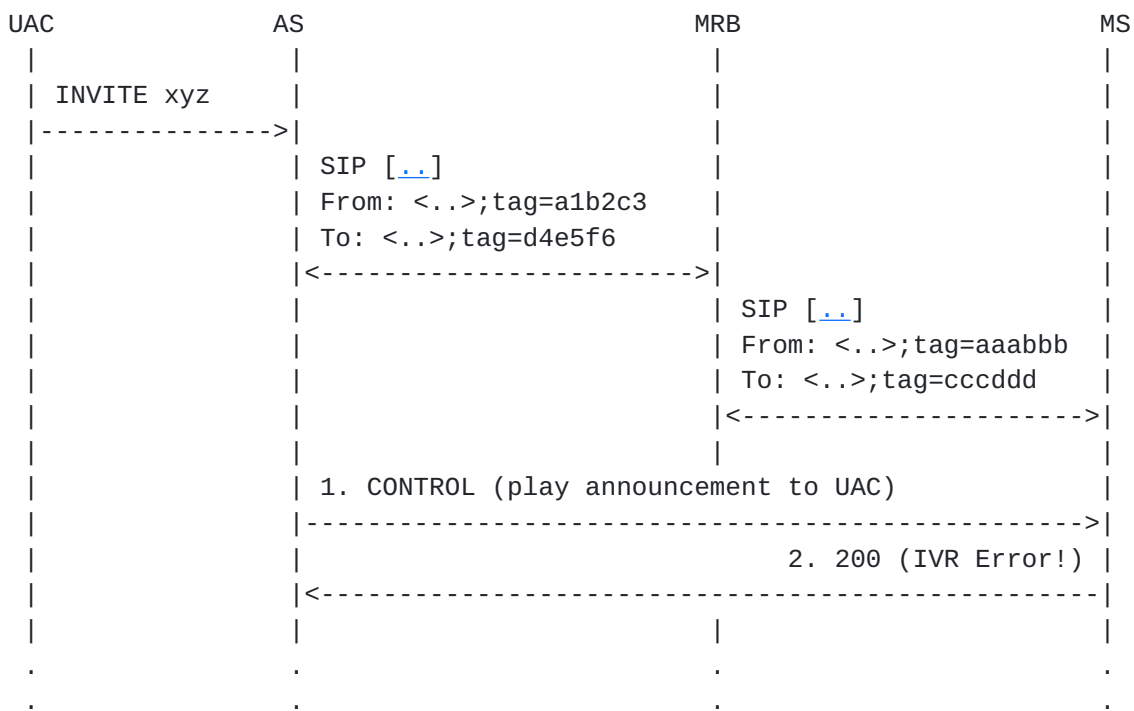


Figure 58: CFW protocol behaviour in case of manipulated tags

In the example, once done with the 3PCC and the UAC being attached to the MS, the AS and the MS end up with different assumptions with respect to the 'connectionid' addressing UAC. In fact, the AS builds a 'connectionid' using the tags it is aware of (a1b2c3:d4e5f6), while the MS builds a different one, considering it got different information from the MRB (aaabbbb:cccddee).

As a consequence, when the AS tries to play an announcement to the UAC using the connectionid it correctly constructed, the MS just as correctly replies with an error, since it doesn't know that

identifier. This is the correct protocol behaviour, caused by a misuse of the information needed for it to work as expected.

1. AS -> MS (CFW CONTROL, play)

CFW ffhg45dzf123 CONTROL

Control-Package: msc-ivr/1.0

Content-Type: application/msc-ivr+xml

Content-Length: 284

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogstart connectionid="a1b2c3:d4e5f6">
    <dialog>
      <prompt>
        <media loc="http://www.example.net/hello.wav"/>
      </prompt>
    </dialog>
  </dialogstart>
</mscivr>
```

2. AS <- MS (CFW 200 OK)

CFW ffhg45dzf123 200

Timeout: 10

Content-Type: application/msc-ivr+xml

Content-Length: 148

```
<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <response status="407" reason="connectionid does not exist"
    dialogid=""/>
</mscivr>
```

In an even worse scenario, the connectionid might actually exist but mapped to a different UAC: in such a case, the transaction would succeed, but a completely different UAC would be involved in the scenarios, thus causing a silent failure neither the AS nor the MS would be aware of.

That said, a proper management of these sensitive pieces of information by the MRB would prevent such failure scenarios to happen. It has already been described how this issue is taken care of in the IAMM case (both CFW-based and media dialog-based). Addressing this issue for the IUMM case is not documented in [\[RFC6917\]](#) as explicitly out of purpose, and as such may be implementation

specific.

The same applies to SDP fields as well: in fact, the AS and MS make use of ad-hoc SDP attributes to instantiate a control channel, as they make use of SDP labels to address specific media connections of a UAC media dialog when a fine-grain approach is needed. As a consequence, any MRB implementation should limit any SDP manipulation as much as possible, or at least take very good care in not causing changes that could break the expected CFW protocol behaviour.

8. Security Considerations

All the MEDIACTRL documents have strong statements regarding security considerations within the context of the interactions occurring at all the levels among the involved parties. Considering the sensitive nature of the interaction between AS and MS, particular efforts have been devoted in providing guidance upon securing what flows through a Control Channel. In fact, transactions concerning dialogs, connections and mixes are quite strongly related to resources actually being deployed and made use of in the MS. This means that it is in the interest of both AS and MS that resources created and handled by an entity are not unwillingly manipulated by a potentially malicious third party.

Considering that strong statements are already provided in the aforementioned documents, and that these documents already give good guidance to implementors with respect to these issues, this section will only provide the reader with some MEDIACTRL call flows, showing how a single secured MS is assumed to reply to different AS when receiving requests that may cross the bounds each AS is constrained within. It is the case, for instance, of generic auditing requests, or explicit conference manipulation requests where the involved identifiers are not part of the context of the originating AS.

To address a very specific scenario, let's assume that two different AS, AS1 and AS2, have established a Control Channel with the same MS. Considering the SYNC transaction an AS and a MS use to set up a control channel, the MS is able to discern the requests coming from AS1 from the ones coming from AS2: in fact, as explained in [Section 5.1](#) and [Section 5.2](#), an AS and a MS negotiate a cfw-id attribute in the SDP, and the same value is subsequently used in the SYNC message on the control channel that is created after the negotiation, thus reassuring both the AS and the MS about the fact that the control channel they share is actually the one they negotiated in the first place.

Let's also assume that AS1 has created a conference mix

(confid=74b6d62) to which it has attached some participants within the context of its business logic, while AS2 has created a currently active IVR dialog (dialogid=dfg3252) with a user agent it is handling (237430727:a338e95f); besides, AS2 has also joined two connections to each other (1:75d4dd0d and 1:b9e6a659). As it is clear, it is highly desirable that AS1 is not aware of what AS2 is doing with the MS and viceversa, and that they are not allowed to manipulate the other's resources. The following transactions will occur, and it will be shown how the MS is assumed to reply in all the cases in order to avoid security issues:

1. AS1 places a generic audit request to both the mixer and IVR packages;
2. AS2 places a generic audit request to both the mixer and IVR packages;
3. AS1 tries to terminate the dialog created by AS2 (6791fee);
4. AS2 tries to join a user agent it handles (1:272e9c05) to the conference mix created by AS1 (74b6d62);

A sequence diagram of the mentioned transactions is depicted in Figure 59:

AS1	AS2	MS
A1. CONTROL (IVR audit)		
++++++		++++++
		A2. 200 OK
<<+++++		++++++
B1. CONTROL (Mixer audit)		
++++++		++++++
		B2. 200 OK
<<+++++		++++++
	C1. CONTROL (IVR audit)	
	++++++	++++++
		C2. 200 OK
	<<+++++	++++++
	D1. CONTROL (Mixer audit)	
	++++++	++++++
		D2. 200 OK
	<<+++++	++++++
E1. CONTROL (dialogterminate)		
++++++		++++++
		E2. 403 Forbidden
<<+++++		++++++
	F1. CONTROL (join UAC&conf[AS1])	
	++++++	++++++
		F2. 403 Forbidden
	<<+++++	++++++
.	.	.
.	.	.

Figure 59: Security Considerations: Framework Transaction

The expected outcome of the transaction is the MS partially "lying" to both AS1 and AS2 when replying to the audit requests (not all the identifiers are reported, but only the ones each AS is directly involved in), and the MS denying the requests for the unauthorized operations (403). Looking at each transaction separately:

- o In the first transaction (A1), AS1 places a generic <audit> request to the IVR package; the request is generic since no attributes are passed as part of the request, meaning AS1 is interested to both the MS capabilities and all the dialogs the MS

is currently handling; as it can be seen in the reply (A2), the MS only reports in the <auditresponse> the package capabilities, while the <dialogs> element is empty; this is because the only dialog the MS is handling has actually been created by AS2, which causes the MS not to report the related identifier (6791fee) to AS1; in fact, AS1 could use that identifier to manipulate the dialog, e.g., by tearing it down thus causing the service to be interrupted without AS2's intervention;

- o In the second transaction, instead (B1), AS1 places an identical <audit> request to the mixer package; the request is again generic, meaning AS1 is interested to both the package capabilities and all the mixers and connections the package is handling at the moment; this time, the MS does not only report capabilities (B2), but information about mixers and connections as well; nevertheless, this information is not complete; in fact, only information about mixers and connections originated by AS1 are reported (mixer 74b6d62 and its participants), while the ones originated by AS2 are omitted in the report; the motivation is the same as before;
- o In the third and fourth transactions (C1 and D1), it's AS2 placing an <audit> request to both the IVR and mixer packages; just as for what happened in the previous transactions, the audit requests are generic; looking at the replies (C2 and D2), it's obvious that the capabilities section is identical to the replies given to AS1; in fact, the MS has no reason to "lie" about what it can do; the <dialogs> and <mixers> sections, instead, are totally different; AS2 in fact receives information about its own IVR dialog (6791fee), which was omitted in the reply to AS1, while it only receives information about the only connection it created (1:75d4dd0d and 1:b9e6a659) without any details related to the mixers and connections originated by AS1;
- o In the fifth transaction (E1) AS1, instead of just auditing the packages, tries to terminate (<dialogterminate>) the dialog created by AS2 (6791fee); since the identifier has not been reported by the MS in the reply to the previous audit request, we assume AS1 got it by a different out of band mechanism; this is assumed to be an unauthorized operation, considering the mentioned dialog is outside the bounds of AS1; for this reason MS, instead of handling the syntactically correct request, replies (E2) with a framework level 403 message (Forbidden), leaving the dialog untouched;
- o Similarly in the sixth and last transaction (F1) AS2 tries to attach (<join>) one of the UACs it is handling to the conference mix created by AS1 (74b6d62); just as in the previous transaction, the identifier is assumed to have been accessed by AS2 through some out of band mechanism, considering that MS didn't report it in the reply to the previous audit request; while one of the identifiers (the UAC) is actually handled by AS2, the other (the

conference mix) is not, and this is again considered by the MS as AS2 stepping outside of its bounds; for the same reason as before, MS replies again (F2) with a framework level 403 message (Forbidden), leaving the mix and the UAC unjoined.

A1. AS1 -> MS (CFW CONTROL, audit IVR)

CFW 140e0f763352 CONTROL
Control-Package: msc-ivr/1.0
Content-Type: application/msc-ivr+xml
Content-Length: 81

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
 <audit/>
</mscivr>

A2. AS1 <- MS (CFW 200, auditresponse)

CFW 140e0f763352 200
Timeout: 10
Content-Type: application/msc-ivr+xml
Content-Length: 1419

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
 <auditresponse status="200">
 <capabilities>
 <dialoglanguages/>
 <grammartypes/>
 <recordtypes>
 <mimetype>audio/x-wav</mimetype>
 <mimetype>video/mpeg</mimetype>
 </recordtypes>
 <prompttypes>
 <mimetype>audio/x-wav</mimetype>
 <mimetype>video/mpeg</mimetype>
 </prompttypes>
 <variables>
 <variabletype type="date"
 desc="value formatted as YYYY-MM-DD">
 <format desc="month year day">mdy</format>
 <format desc="year month day">ymd</format>
 <format desc="day month year">dmy</format>
 <format desc="day month">dm</format>
 </variabletype>
 <variabletype type="time" desc="value formatted as HH:MM">


```

        <format desc="24 hour format">t24</format>
        <format desc="12 hour format with am/pm">t12</format>
    </variabletype>
    <variabletype type="digits" desc="value formatted as D+">
        <format desc="general digit string">gen</format>
        <format desc="cardinal">crn</format>
        <format desc="ordinal">ord</format>
    </variabletype>
</variables>
<maxpreparedduration>60s</maxpreparedduration>
<maxrecordduration>1800s</maxrecordduration>
<codecs>
    <codec name="audio"><subtype>basic</subtype></codec>
    <codec name="audio"><subtype>gsm</subtype></codec>
    <codec name="video"><subtype>h261</subtype></codec>
    <codec name="video"><subtype>h263</subtype></codec>
    <codec name="video"><subtype>h263-1998</subtype></codec>
    <codec name="video"><subtype>h264</subtype></codec>
</codecs>
</capabilities>
<dialogs>
</dialogs>
</auditresponse>
</mscivr>

```

B1. AS1 -> MS (CFW CONTROL, audit mixer)

```

-----
CFW 0216231b1f16 CONTROL
Control-Package: msc-mixer/1.0
Content-Type: application/msc-mixer+xml
Content-Length: 87

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
    <audit/>
</mscmixer>

```

B2. AS1 <- MS (CFW 200, auditresponse)

```

-----
CFW 0216231b1f16 200
Timeout: 10
Content-Type: application/msc-mixer+xml
Content-Length: 903

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
    <auditresponse status="200">
        <capabilities>

```



```

    <codecs>
      <codec name="audio"><subtype>basic</subtype></codec>
      <codec name="audio"><subtype>gsm</subtype></codec>
      <codec name="video"><subtype>h261</subtype></codec>
      <codec name="video"><subtype>h263</subtype></codec>
      <codec name="video"><subtype>h263-1998</subtype></codec>
      <codec name="video"><subtype>h264</subtype></codec>
    </codecs>
  </capabilities>
</mixers>
  <conferenceaudit conferenceid="74b6d62">
    <participants>
      <participant id="1864574426:e2192766"/>
      <participant id="1:5a97fd79"/>
    </participants>
    <video-layout min-participants="1">
      <quad-view/>
    </video-layout>
  </conferenceaudit>
  <joinaudit id1="1864574426:e2192766" id2="74b6d62"/>
  <joinaudit id1="1:5a97fd79" id2="74b6d62"/>
</mixers>
</auditresponse>
</mscmixer>

```

C1. AS2 -> MS (CFW CONTROL, audit IVR)

```

-----
CFW 0216231b1f16 CONTROL
Control-Package: msc-ivr/1.0
Content-Type: application/msc-ivr+xml
Content-Length: 81

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <audit/>
</mscivr>

```

C2. AS2 <- MS (CFW 200, auditresponse)

```

-----
CFW 0216231b1f16 200
Timeout: 10
Content-Type: application/msc-ivr+xml
Content-Length: 1502

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <auditresponse status="200">
    <capabilities>

```



```
<dialoglanguages/>
<grammartypes/>
<recordtypes>
  <mimetype>audio/wav</mimetype>
  <mimetype>video/mpeg</mimetype>
</recordtypes>
<prompttypes>
  <mimetype>audio/wav</mimetype>
  <mimetype>video/mpeg</mimetype>
</prompttypes>
<variables>
  <variabletype type="date"
    desc="value formatted as YYYY-MM-DD">
    <format desc="month year day">mdy</format>
    <format desc="year month day">ymd</format>
    <format desc="day month year">dmy</format>
    <format desc="day month">dm</format>
  </variabletype>
  <variabletype type="time" desc="value formatted as HH:MM">
    <format desc="24 hour format">t24</format>
    <format desc="12 hour format with am/pm">t12</format>
  </variabletype>
  <variabletype type="digits" desc="value formatted as D+">
    <format desc="general digit string">gen</format>
    <format desc="cardinal">crn</format>
    <format desc="ordinal">ord</format>
  </variabletype>
</variables>
<maxpreparedduration>60s</maxpreparedduration>
<maxrecordduration>1800s</maxrecordduration>
<codecs>
  <codec name="audio"><subtype>basic</subtype></codec>
  <codec name="audio"><subtype>gsm</subtype></codec>
  <codec name="video"><subtype>h261</subtype></codec>
  <codec name="video"><subtype>h263</subtype></codec>
  <codec name="video"><subtype>h263-1998</subtype></codec>
  <codec name="video"><subtype>h264</subtype></codec>
</codecs>
</capabilities>
<dialogs>
  <dialogaudit dialogid="6791fee" state="started"
    connectionid="237430727:a338e95f"/>
</dialogs>
</auditresponse>
</mscivr>
```



```
-----
CFW 515f007c5bd0 CONTROL
Control-Package: msc-mixer/1.0
Content-Type: application/msc-mixer+xml
Content-Length: 87

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <audit/>
</mscmixer>
```

D2. AS2 <- MS (CFW 200, auditresponse)

```
-----
CFW 515f007c5bd0 200
Timeout: 10
Content-Type: application/msc-mixer+xml
Content-Length: 548

<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
<auditresponse status="200">
  <capabilities>
    <codecs>
      <codec name="audio"><subtype>basic</subtype></codec>
      <codec name="audio"><subtype>gsm</subtype></codec>
      <codec name="video"><subtype>h261</subtype></codec>
      <codec name="video"><subtype>h263</subtype></codec>
      <codec name="video"><subtype>h263-1998</subtype></codec>
      <codec name="video"><subtype>h264</subtype></codec>
    </codecs>
  </capabilities>
  <mixers>
    <joinaudit id1="1:75d4dd0d" id2="1:b9e6a659"/>
  </mixers>
</auditresponse>
</mscmixer>
```

E1. AS1 -> MS (CFW CONTROL, dialogterminate)

```
-----
CFW 7fdcc2331bef CONTROL
Control-Package: msc-ivr/1.0
Content-Type: application/msc-ivr+xml
Content-Length: 127

<mscivr version="1.0" xmlns="urn:ietf:params:xml:ns:msc-ivr">
  <dialogterminate dialogid="6791fee" immediate="true"/>
</mscivr>
```


E2. AS1 <- MS (CFW 403 Forbidden)

CFW 7fdcc2331bef 403

F1. AS2 -> MS (CFW CONTROL, join to conference)

CFW 140e0f763352 CONTROL

Control-Package: msc-mixer/1.0

Content-Type: application/msc-mixer+xml

Content-Length: 117

```
<mscmixer version="1.0" xmlns="urn:ietf:params:xml:ns:msc-mixer">
  <join id1="1:272e9c05" id2="74b6d62"/>
</mscmixer>
```

F2. AS2 <- MS (CFW 403 Forbidden)

CFW 140e0f763352 403

9. IANA Considerations

This document has no actions for IANA.

10. Change Summary

Note to RFC Editor: Please remove this whole section.

The following are the major changes between the 12 and the 13 versions of the draft:

- o Fixed some nits in the document.
- o Addressed additional comments and typos reported by Dale Worley from his review on the mailing list.
- o Clarified the IUMM scenario with respect to how media dialogs are handled.
- o Removed reference to expired [draft-boulton-mmusic-sdp-control-package-attribute](#) and related example.

The following are the major changes between the 11 and the 12 versions of the draft:

- o Fixed some nits in the document.
- o Addressed additional comments and typos reported by Dale Worley from his review on the mailing list.

The following are the major changes between the 10 and the 11 versions of the draft:

- o Updated references: [RFC6917](#) (was MRB draft).
- o Addressed comments and typos reported by Dale Worley from his review on the mailing list.

The following are the major changes between the 09 and the 10 versions of the draft:

- o Clarified how XML elements may be splitted across lines because of the 72 characters limitation.

The following are the major changes between the 08 and the 09 versions of the draft:

- o Aligned all the examples to the latest package schemas (MRB in particular), and validated them.
- o Updated references: [RFC6505](#) (was mixer package).
- o Changed the term "call leg" to "media dialog".

The following are the major changes between the 07 and the 08 versions of the draft:

- o Aligned all the examples to the latest package schemas (MRB in particular), and validated them.
- o Removed useless backslashes from XML examples.
- o Updated references and applied fixes suggested by the Idnits tool.

The following are the major changes between the 06 and the 07 versions of the draft:

- o Updated references: [RFC6230](#) (was control framework draft) and [RFC6231](#) (was IVR package).
- o Aligned all the examples to the latest package schemas (MRB in particular), and validated them.
- o Modified Publish example by showing the use of the new minfrequency and maxfrequency elements.
- o Modified IAMM Consumer example to show two different approaches, CFW-based and Call-leg based.
- o Enriched the connection-id issue section, by providing examples on the Consumer connection-id element.

- o Clarified that solving the connection-id issue in the IUMM case is implementation specific.

The following are the major changes between the 05 and the 06 versions of the draft:

- o Aligned all the examples to the latest package schemas, and validated them.

The following are the major changes between the 04 and the 05 versions of the draft:

- o Added the missing <encoding> and <decoding> elements to the <rtp-codec> instances, where needed (MRB section);
- o Validated all the examples according to the schemas, and fixed implementation and snippets where needed.

The following are the major changes between the 03 and the 04 versions of the draft:

- o corrected flow in [Section 6.3.2](#);
- o updated examples with respect to package names (version added) and codec names (MIME type);
- o added a new Publishing request to the existing dump to address a subscription update;
- o added a completely new section ([Section 7.3](#)) to address the call legs management in presence of MRBs;

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

- o enriched MRB section text;
- o updated MRB Publishing scenario;
- o added MRB Consumer scenarios, both Inline and Query ([Section 7](#));

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

- o changed the m-line of COMEDIA negotiation according to [\[RFC6230\]](#);
- o changed the token used to build connection identifiers from '~' to ':';

- o corrected the flow presented in [Section 6.4.3](#), where messages E1 and G1 were more verbose than needed;
- o added placeholder section for Media Resource Brokering ([Section 7](#))

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

- o updated the flows according to the latest drafts;
- o corrected the reference to the Conferencing Scenarios RFC (4597 instead of 4579);
- o added K-ALIVE example and some common mistakes in the Control Channel Establishment section;
- o added text, diagrams and flows to the Sidebars scenario;
- o added text, diagrams and flows to the Floor Control scenario; Floor Control scenario moved at the end of the conferencing section;
- o added text to the Security Considerations;

[11.](#) Acknowledgements

The authors would like to thank Dale Worley for the thorough review of the whole document, and for contributing text to make the document easier to read.

[12.](#) References

[12.1.](#) Normative References

- [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.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", [RFC 3264](#), June 2002.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, [RFC 3550](#), July 2003.

- [RFC4574] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", [RFC 4574](#), August 2006.
- [RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", [RFC 4145](#), September 2005.
- [RFC4572] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", [RFC 4572](#), July 2006.
- [RFC6230] Boulton, C., Melanchuk, T., and S. McGlashan, "Media Control Channel Framework", [RFC 6230](#), May 2011.
- [RFC6231] McGlashan, S., Melanchuk, T., and C. Boulton, "An Interactive Voice Response (IVR) Control Package for the Media Control Channel Framework", [RFC 6231](#), May 2011.
- [RFC6505] McGlashan, S., Melanchuk, T., and C. Boulton, "A Mixer Control Package for the Media Control Channel Framework", [RFC 6505](#), March 2012.
- [RFC6917] Boulton, C., Miniero, L., and G. Munson, "Media Resource Brokering", [RFC 6917](#), April 2013.
- [RFC5239] Barnes, M., Boulton, C., and O. Levin, "A Framework for Centralized Conferencing", [RFC 5239](#), June 2008.
- [RFC4582] Camarillo, G., Ott, J., and K. Drage, "The Binary Floor Control Protocol (BFCP)", [RFC 4582](#), November 2006.
- [RFC4583] Camarillo, G., "Session Description Protocol (SDP) Format for Binary Floor Control Protocol (BFCP) Streams", [RFC 4583](#), November 2006.

12.2. Informative References

- [RFC2606] Eastlake, D. and A. Panitz, "Reserved Top Level DNS Names", [BCP 32](#), [RFC 2606](#), June 1999.
- [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.
- [SRGS] Hunt, A. and S. McGlashan, "Speech Recognition Grammar Specification Version 1.0", W3C Recommendation, March 2004.

[RFC4597] Even, R. and N. Ismail, "Conferencing Scenarios",
[RFC 4597](#), August 2006.

[RFC5567] Melanchuk, T., "An Architectural Framework for Media
Server Control", [RFC 5567](#), June 2009.

Authors' Addresses

Alessandro Amirante
University of Napoli
Via Claudio 21
Napoli 80125
Italy

Email: alessandro.amirante@unina.it

Tobia Castaldi
Meetecho
Via Carlo Poerio 89
Napoli 80100
Italy

Email: tcastaldi@meetecho.com

Lorenzo Miniero
Meetecho
Via Carlo Poerio 89
Napoli 80100
Italy

Email: lorenzo@meetecho.com

Simon Pietro Romano
University of Napoli
Via Claudio 21
Napoli 80125
Italy

Email: spromano@unina.it

