Internet Engineering Task Force (IETF)
Request for Comments: 6917
Category: Standards Track
ISSN: 2070-1721

C. Boulton
NS-Technologies
L. Miniero
Meetecho
G. Munson
AT&T

April 2013

Media Resource Brokering

Abstract

The MediaCtrl working group in the IETF has proposed an architecture for controlling media services. The Session Initiation Protocol (SIP) is used as the signaling protocol that provides many inherent capabilities for message routing. In addition to such signaling properties, a need exists for intelligent, application-level media service selection based on non-static signaling properties. This is especially true when considered in conjunction with deployment architectures that include 1:M and M:N combinations of Application Servers and Media Servers. This document introduces a Media Resource Broker (MRB) entity, which manages the availability of Media Servers and the media resource demands of Application Servers. The document includes potential deployment options for an MRB and appropriate interfaces to Application Servers and Media Servers.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6917.

Table of Contents

1. Introduction

As IP-based multimedia infrastructures mature, the complexity and demands from deployments increase. Such complexity will result in a wide variety of capabilities from a range of vendors that should all be interoperable using the architecture and protocols produced by the MediaCtrl working group. It should be possible for a controlling entity to be assisted in Media Server selection so that the most appropriate resource is selected for a particular operation. The importance increases when one introduces a flexible level of deployment scenarios, as specified in RFC 5167 [RFC5167] and RFC 5567 [RFC5567]. These documents make statements like "it should be possible to have a many-to-many relationship between Application Servers and Media Servers that use this protocol". This leads to the following deployment architectures being possible when considering media resources, to provide what can be effectively described as media resource brokering.

The simplest deployment view is illustrated in Figure 1.

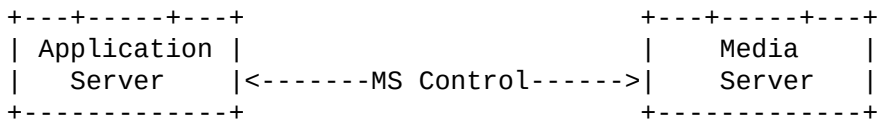


Figure 1: Basic Architecture

This simply involves a single Application Server and Media Server. Expanding on this view, it is also possible for an Application Server to control multiple (greater than 1) Media Server instances at any one time. This deployment view is illustrated in Figure 2. Typically, such architectures are associated with application logic that requires high-demand media services. It is more than possible

that each Media Server possesses a different media capability set. Media Servers may offer different media services as specified in the MediaCtrl architecture document [RFC5567]. A Media Server may have similar media functionality but may have different capacity or media codec support.

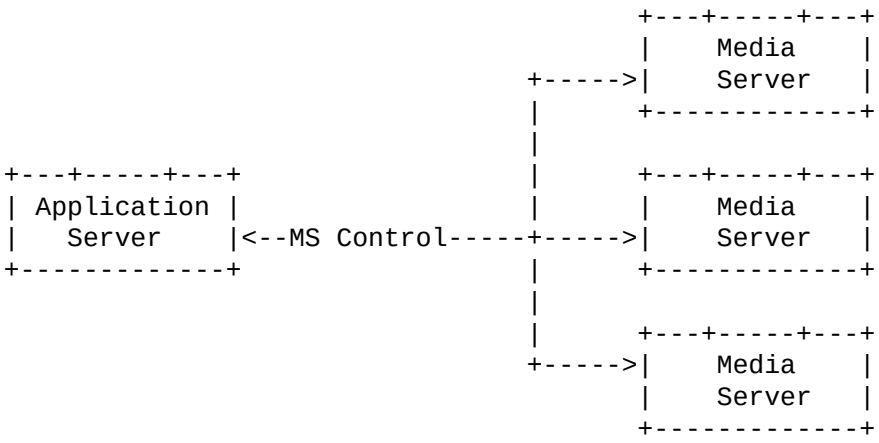


Figure 2: Multiple Media Servers

Figure 3 conveys the opposite view to that in Figure 2. In this model, there are a number of (greater than 1) Application Servers, possibly supporting dissimilar applications, controlling a single Media Server. Typically, such architectures are associated with application logic that requires low-demand media services.

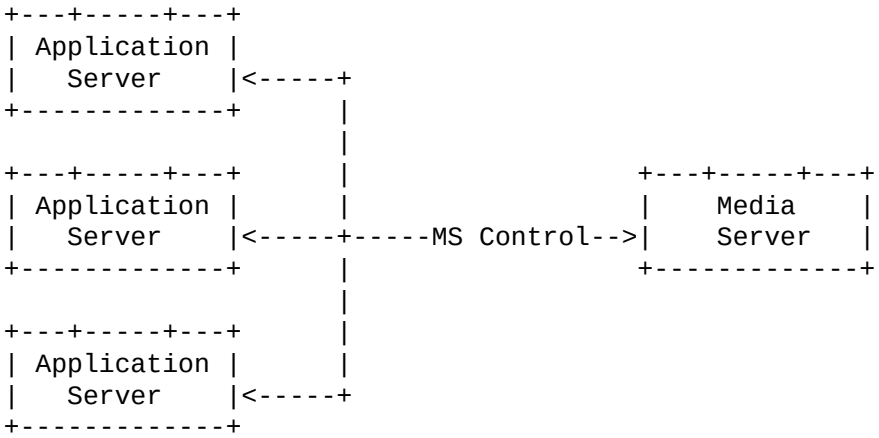


Figure 3: Multiple Application Servers

The final deployment view is the most complex (Figure 4). In this model (M:N), there exist any number of Application Servers and any number of Media Servers. It is again possible in this model that Media Servers might not be homogeneous, and they might have different capability sets and capacities.

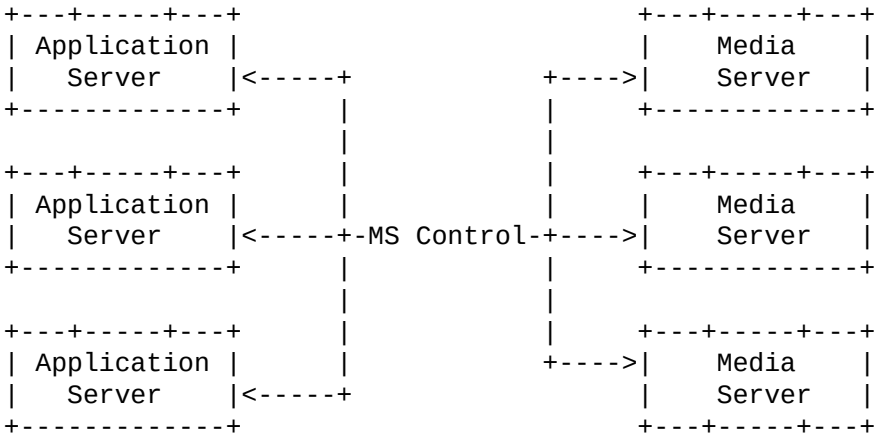


Figure 4: Many-to-Many Architecture

The remaining sections in this specification will focus on a new entity called a Media Resource Broker (MRB), which can be utilized in the deployment architectures described previously in this section. The MRB entity provides the ability to obtain media resource information and appropriately allocate (broker) on behalf of client applications.

The high-level deployment options discussed in this section rely on network architecture and policy to prohibit inappropriate use. Such policies are out of scope for this document.

This document will take a look at the specific problem areas related to such deployment architectures. It is recognized that the solutions proposed in this document should be equally adaptable to all of the previously described deployment models. It is also recognized that the solution is far more relevant to some of the previously discussed deployment models and can almost be viewed as redundant on others.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document inherits terminology proposed in RFC 5567 [RFC5567] and in "Media Control Channel Framework" [RFC6230]. In addition, the following terms are defined for use in this document and for use in the context of the MediaCtrl working group in the IETF:

Media Resource Broker (MRB): A logical entity that is responsible for both collection of appropriate published Media Server (MS) information and selecting appropriate Media Server resources on behalf of consuming entities.

Query MRB: An instantiation of an MRB (see previous definition) that provides an interface for an Application Server to retrieve the address of an appropriate Media Server. The result returned to the Application Server can be influenced by information contained in the query request.

In-line MRB: An instantiation of an MRB (see previous definition) that directly receives requests on the signaling path. There is no separate query.

CFW: Media Control Channel Framework, as specified in [RFC6230].

Within the context of In-line MRBs, additional terms are defined:

In-line Aware MRB Mode (IAMM): Defined in Section 5.2.2.1.

In-line Unaware MRB Mode (IUMM): Defined in Section 5.3.

The document will often specify when a specific identifier in a protocol message needs to be unique. Unless stated otherwise, such uniqueness will always be within the scope of the Media Servers controlled by the same MRB. The interaction between different MRB instances, e.g., the partitioning of a logical MRB, is out of scope for this document.

3. Problem Discussion

As discussed in Section 1, a goal of the MediaCtrl working group is to produce a solution that will service a wide variety of deployment architectures. Such architectures range from the simplest 1:1 relationship between Media Servers and Application Servers to potentially linearly scaling 1:M, M:1, and M:N deployments.

Managing such deployments is itself non-trivial for the proposed solution until an additional number of factors that increase complexity are included in the equation. As Media Servers evolve, it must be taken into consideration that, where many can exist in a deployment, they may not have been produced by the same vendor and may not have the same capability set. It should be possible for an Application Server that exists in a deployment to select a media service based on a common, appropriate capability set. In conjunction with capabilities, it is also important to take available resources into consideration. The ability to select an appropriate media service function is an extremely useful feature but becomes even more powerful when considered with available resources for servicing a request.

In conclusion, the intention is to create a toolkit that allows MediaCtrl deployments to effectively utilize the available media resources. It should be noted that in the simplest deployments where only a single Media Server exists, an MRB function is probably not required. Only a single capability set exists, and resource availability can be handled using the appropriate underlying signaling, e.g., SIP response. This document does not prohibit such uses of an MRB; it simply provides the tools for various entities to interact where appropriate. It is also worth noting that the functions specified in this document aim to provide a 'best effort' view of media resources at the time of request for initial Media Server routing decisions. Any dramatic change in media capabilities or capacity after a request has taken place should be handled by the underlying protocol.

It should be noted that there may be additional information that is desirable for the MRB to have for purposes of selecting a Media Server resource, such as resource allocation rules across different applications, planned or unplanned downtime of Media Server resources, the planned addition of future Media Server resources, or Media Server resource capacity models. How the MRB acquires such information is outside the scope of this document. The specific techniques used for selecting an appropriate media resource by an MRB is also outside the scope of this document.

4. Deployment Scenario Options

Research into media resource brokering concluded that a couple of high-level models provided an appropriate level of flexibility. The general principles of "in-line" and "query" MRB concepts are discussed in the rest of this section. It should be noted that while the interfaces are different, they both use common underlying mechanisms defined in this specification.

4.1. Query MRB

The "Query" model for MRB interactions provides the ability for a client of media services (for example, an Application Server) to "ask" an MRB for an appropriate Media Server, as illustrated in Figure 5.

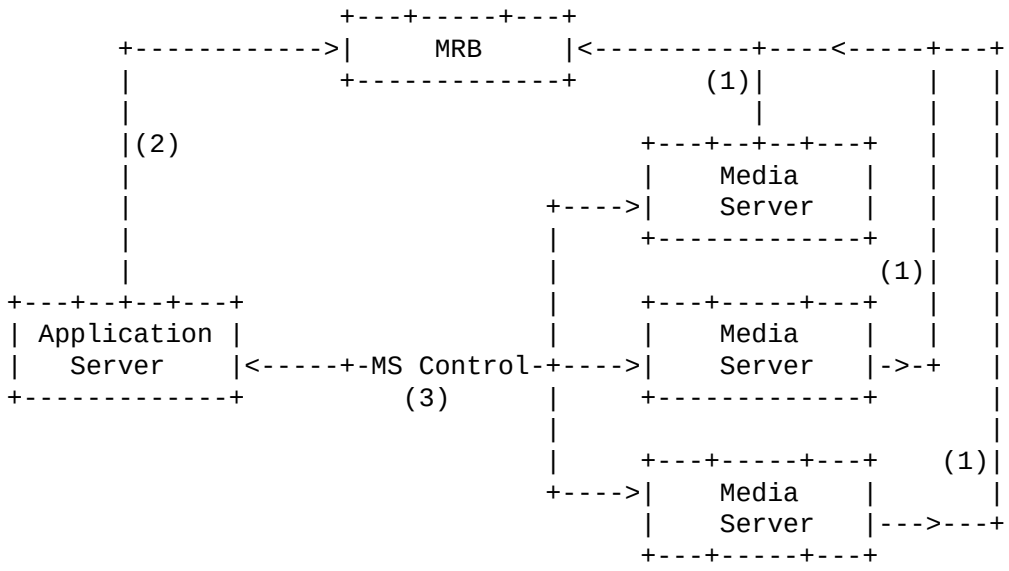


Figure 5: Query MRB

In this deployment, the Media Servers use the Media Server Resource Publish interface, as discussed in Section 5.1, to convey capability sets as well as resource information. This is depicted by (1) in Figure 5. It is then the MRB's responsibility to accumulate all appropriate information relating to media services in the logical deployment cluster. The Application Server (or other media services client) is then able to query the MRB for an appropriate resource (as identified by (2) in Figure 5). Such a query would carry specific information related to the media service required and enable the MRB to provide increased accuracy in its response. This particular interface is discussed in "Media Service Resource Consumer Interface" (Section 5.2). The Application Server is then able to direct control commands (for example, create a conference) and media dialogs to the appropriate Media Server, as shown by (3) in Figure 5. Additionally, with Query mode, the MRB is not directly in the signaling path between the Application Server and the selected Media Server resource.

4.1.1. Hybrid Query MRB

As mentioned previously, it is the intention that a toolkit is provided for MRB functionality within a MediaCtrl architecture. It is expected that in specific deployment scenarios the role of the MRB might be co-hosted as a hybrid logical entity with an Application Server, as shown in Figure 6.

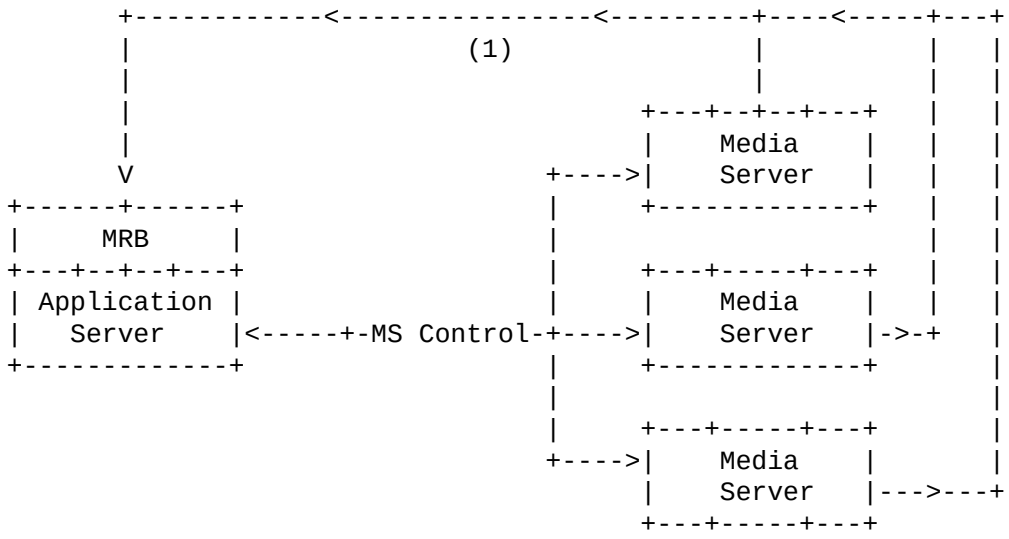


Figure 6: Hybrid Query MRB - Application Server Hosted

This diagram is identical to that in Figure 5 with the exception that the MRB is now hosted on the Application Server. The Media Server Publish interface is still being used to accumulate resource information at the MRB, but as it is co-hosted on the Application Server, the Media Server Consumer interface has collapsed. It might still exist within the Application Server/MRB interaction, but this is an implementation issue. This type of deployment suits a single Application Server environment, but it should be noted that a Media Server Consumer interface could then be offered from the hybrid if required.

In a similar manner, the Media Server could also act as a hybrid for the deployment cluster, as illustrated in Figure 7.

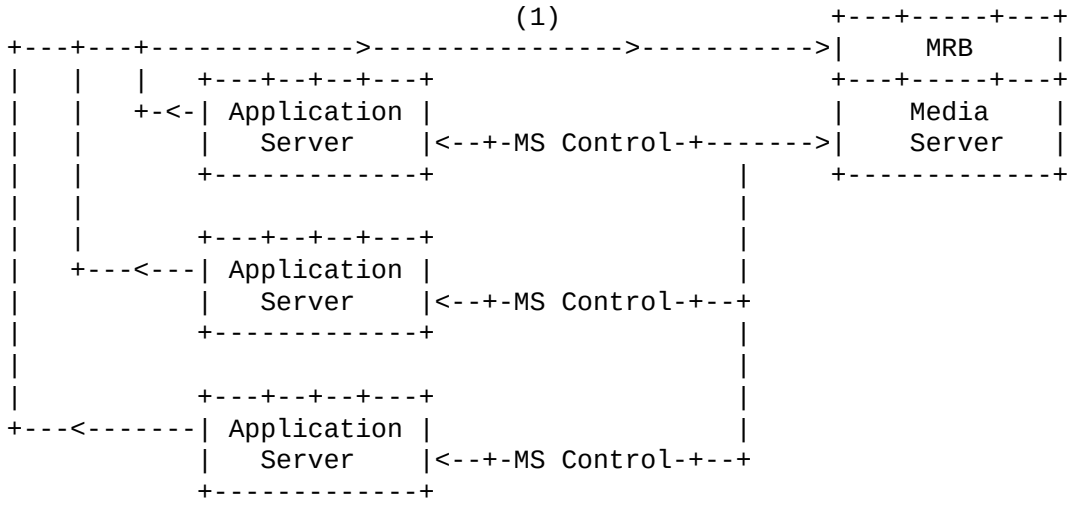


Figure 7: Hybrid Query MRB - MS Hosted

In this example, the MRB has collapsed and is co-hosted by the Media Server. The Media Server Consumer interface is still available to the Application Servers (1) to query Media Server resources. The Media Server Publish interface has collapsed onto the Media Server. It might still exist within the Media Server/MRB interaction, but this is an implementation issue. This type of deployment suits a single Media Server environment, but it should be noted that a Media Server Publish interface could then be offered from the hybrid if required. A typical use case scenario for such a topology would be a single Media Server representing a pool of MSs in a cluster. In this case, the MRB would actually be handling a cluster of Media Servers, rather than one.

4.2. In-Line MRB

The "In-line" MRB is architecturally different from the "Query" model discussed in the previous section. The concept of a separate query disappears. The client of the MRB simply uses the media resource control and media dialog signaling to involve the MRB. This type of deployment is illustrated in Figure 8.

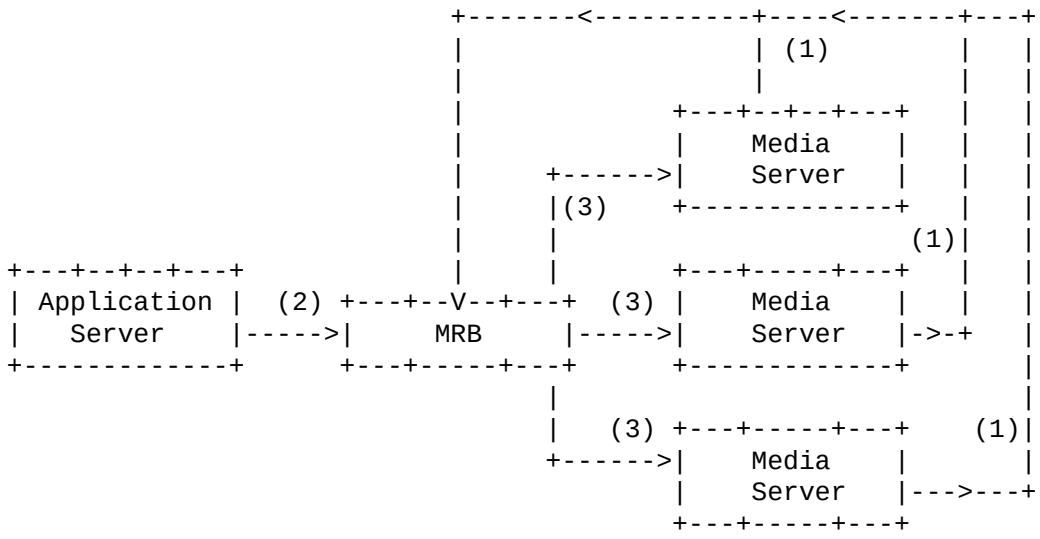


Figure 8: In-Line MRB

The Media Servers still use the Media Server Publish interface to convey capabilities and resources to the MRB, as illustrated by (1). The Media Server Control Channels (and media dialogs as well, if required) are sent to the MRB (2), which then selects an appropriate Media Server (3) and remains in the signaling path between the Application Server and the Media Server resources.

The In-line MRB can be split into two distinct logical roles that can be applied on a per-request basis. They are:

In-line Unaware MRB Mode (IUMM): Allows an MRB to act on behalf of clients requiring media services who are not aware of an MRB or its operation. In this case, the Application Server does not provide explicit information on the kind of Media Server resource it needs (as in Section 5.2), and the MRB is left to deduce it by potentially inspecting other information in the request from the Application Server (for example, Session Description Protocol (SDP) content, or address of the requesting Application Server, or additional Request-URI parameters as per RFC 4240 [RFC4240]).

In-line Aware MRB Mode (IAMM): Allows an MRB to act on behalf of clients requiring media services who are aware of an MRB and its operation. In particular, it allows the Application Server to explicitly convey matching characteristics to those provided by Media Servers, as does the Query MRB mode (as in Section 5.2).

In either of the previously described roles, signaling as specified by the Media Control Channel Framework ([RFC6230]) would be involved, and the MRB would deduce that the selected Media Server resources are no longer needed when the Application Server or Media Server terminates the corresponding SIP dialog. The two modes are discussed in more detail in Section 5.3.

5. MRB Interface Definitions

The intention of this specification is to provide a toolkit for a variety of deployment architectures where media resource brokering can take place. Two main interfaces are required to support the differing requirements. The two interfaces are described in the remainder of this section and have been named the Media Server Resource Publish and Media Server Resource Consumer interfaces.

It is beyond the scope of this document to define exactly how to construct an MRB using the interfaces described. It is, however, important that the two interfaces are complimentary so that development of appropriate MRB functionality is supported.

5.1. Media Server Resource Publish Interface

The Media Server Resource Publish interface is responsible for providing an MRB with appropriate Media Server resource information. As such, this interface is assumed to provide both general and specific details related to Media Server resources. This information needs to be conveyed using an industry standard mechanism to provide increased levels of adoption and interoperability. A Control Package for the Media Control Channel Framework will be specified to fulfill this interface requirement. It provides an establishment and monitoring mechanism to enable a Media Server to report appropriate statistics to an MRB. The Publish interface is used with both the Query mode and In-line mode of MRB operation.

As already discussed in Section 1, the MRB view of Media Server resource availability will in reality be approximate -- i.e., partial and imperfect. The MRB Publish interface does not provide an exhaustive view of current Media Server resource consumption; the Media Server may in some cases provide a best-effort computed view of resource consumption parameters conveyed in the Publish interface (e.g., Digital Signal Processors (DSPs) with a fixed number of

streams versus Graphics Processing Units (GPUs) with CPU availability). Media resource information may only be reported periodically over the Publish interface to an MRB.

It is also worth noting that while the scope of the MRB is in providing interested Application Servers with the available resources, the MRB also allows for the retrieval of information about consumed resources. While this is of course a relevant piece of information (e.g., for monitoring purposes), such functionality inevitably raises security considerations, and implementations should take this into account. See Section 12 for more details.

The MRB Publish interface uses the Media Control Channel Framework ([RFC6230]) as the basis for interaction between a Media Server and an MRB. The Media Control Channel Framework uses an extension mechanism to allow specific usages that are known as Control Packages. Section 5.1.1 defines the Control Package that MUST be implemented by any Media Server wanting to interact with an MRB entity.

5.1.1. Control Package Definition

This section fulfills the requirement for information that must be specified during the definition of a Control Framework package, as detailed in Section 8 of [RFC6230].

5.1.1.1. Control Package Name

The Media Channel Control Framework requires a Control Package definition to specify and register a unique name and version.

The name and version of this Control Package is "mrb-publish/1.0".

5.1.1.2. Framework Message Usage

The MRB Publish interface allows a Media Server to convey available capabilities and resources to an MRB entity.

This package defines XML elements in Section 5.1.2 and provides an XML schema in Section 10.

The XML elements in this package are split into requests, responses, and event notifications. Requests are carried in CONTROL message bodies; the <mrbrequest> element is defined as a package request. This request can be used for creating new subscriptions and updating/removing existing subscriptions. Event notifications are also carried in CONTROL message bodies; the <mrbnotification> element is

defined for package event notifications. Responses are carried either in REPORT message or Control Framework 200 response bodies; the <mrbresponse> element is defined as a package-level response.

Note that package responses are different from framework response codes. Framework error response codes (see Section 7 of [RFC6230]) are used when the request or event notification is invalid; for example, a request has invalid XML (400) or is not understood (500). Package-level responses are carried in framework 200 response or REPORT message bodies. This package's response codes are defined in Section 5.1.4.

5.1.1.3. Common XML Support

The Media Control Channel Framework [RFC6230] requires a Control Package definition to specify if the attributes for media dialog or conference references are required.

The Publish interface defined in Section 10 does import and make use of the common XML schema defined in the Media Control Channel Framework.

The Consumer interface defined in Section 11 does import and make use of the common XML schema defined in the Media Control Channel Framework.

5.1.1.4. CONTROL Message Body

A valid CONTROL message body MUST conform to the schema defined in Section 10 and described in Section 5.1.2. XML messages appearing in CONTROL messages MUST contain either an <mrbrequest> or <mrbnotification> element.

5.1.1.5. REPORT Message Body

A valid REPORT message body MUST conform to the schema defined in Section 10 and described in Section 5.1.2. XML messages appearing in REPORT messages MUST contain an <mrbresponse> element.

5.1.1.6. Audit

The 'mrb-publish/1.0' Media Control Channel Framework package does not require any additional auditing capability.

5.1.2. Element Definitions

This section defines the XML elements for the Publish interface Media Control Channel package defined in Section 5.1. The formal XML schema definition for the Publish interface can be found in Section 10.

The root element is <mrbpublish>. All other XML elements (requests, responses, notifications) are contained within it. The MRB Publish interface request element is detailed in Section 5.1.3. The MRB Publish interface notification element is detailed in Section 5.1.5. The MRB Publish interface response element is detailed in Section 5.1.4.

The <mrbpublish> element has the following attributes:

version: a token specifying the mrb-publish package version. The value is fixed as '1.0' for this version of the package. The attribute MUST be present.

The <mrbpublish> element has the following child elements, and there MUST NOT be more than one such child element in any <mrbpublish> message:

<mrbrequest> for sending an MRB request. See Section 5.1.3.

<mrbresponse> for sending an MRB response. See Section 5.1.4.

<mrbnotification> for sending an MRB notification. See Section 5.1.5.

5.1.3. <mrbrequest>

This section defines the <mrbrequest> element used to initiate requests from an MRB to a Media Server. The element describes information relevant for the interrogation of a Media Server.

The <mrbrequest> element has no defined attributes.

The <mrbrequest> element has the following child element:

<subscription> for initiating a subscription to a Media Server from an MRB. See Section 5.1.3.1.

5.1.3.1. <subscription>

The <subscription> element is included in a request from an MRB to a Media Server to provide the details relating to the configuration of updates (known as a subscription session). This element can be used either to request a new subscription or to update an existing one (e.g., to change the frequency of the updates), and to remove ongoing subscriptions as well (e.g., to stop an indefinite update). The MRB will inform the Media Server regarding how long it wishes to receive updates and the frequency that updates should be sent. Updates related to the subscription are sent using the <mrbnotification> element.

The <subscription> element has the following attributes:

id: Indicates a unique token representing the subscription session between the MRB and the Media Server. The attribute MUST be present.

seqnumber: Indicates a sequence number to be used in conjunction with the subscription session ID to identify a specific subscription command. The first subscription MUST contain a non-zero number 'seqnumber', and subsequent subscriptions MUST contain a higher number than the previous 'seqnumber' value. If a subsequent 'seqnumber' is not higher, a 405 response code is generated as per Section 5.1.4. The attribute MUST be present.

action: Provides the operation that should be carried out on the subscription:

* The value of 'create' instructs the Media Server to attempt to set up a new subscription.

* The value of 'update' instructs the Media Server to attempt to update an existing subscription.

* The value of 'remove' instructs the Media Server to attempt to remove an existing subscription and consequently stop any ongoing related notification.

The attribute MUST be present.

The <subscription> element has zero or more of the following child elements:

<expires>: Provides the amount of time in seconds that a subscription should be installed for notifications at the Media Server. Once the amount of time has passed, the subscription expires, and the MRB has to subscribe again if it is still interested in receiving notifications from the Media Server. The element MAY be present.

<minfrequency>: Provides the minimum frequency in seconds that the MRB wishes to receive notifications from the Media Server. The element MAY be present.

<maxfrequency>: Provides the maximum frequency in seconds that the MRB wishes to receive notifications from the Media Server. The element MAY be present.

Please note that these three optional pieces of information provided by the MRB only act as a suggestion: the Media Server MAY change the proposed values if it considers the suggestions unacceptable (e.g., if the MRB has requested a notification frequency that is too high). In such a case, the request would not fail, but the updated, acceptable values would be reported in the <mrbresponse> accordingly.

5.1.4. <mrbresponse>

Responses to requests are indicated by an <mrbresponse> element.

The <mrbresponse> element has the following attributes:

status: numeric code indicating the response status. The attribute MUST be present.

reason: string specifying a reason for the response status. The attribute MAY be present.

The <mrbresponse> element has a single child element:

<subscription> for providing details related to a subscription requested by a Media Server (see below in this section).

The following status codes are defined for 'status':

code	description
200	OK
400	Syntax error
401	Unable to create Subscription
402	Unable to update Subscription
403	Unable to remove Subscription
404	Subscription does not exist
405	Wrong sequence number
406	Subscription already exists
420	Unsupported attribute or element

Table 1: <mrbresponse> Status Codes

If a new subscription request made by an MRB (action='create') has been accepted, the Media Server MUST reply with an <mrbresponse> with status code 200. The same rule applies whenever a request to update (action='update') or remove (action='remove') an existing transaction can be fulfilled by the Media Server.

A subscription request, nevertheless, may fail for several reasons. In such a case, the status codes defined in Table 1 must be used instead. Specifically, if the Media Server fails to handle a request due to a syntax error in the request itself (e.g., incorrect XML, violation of the schema constraints, or invalid values in any of the attributes/elements), the Media Server MUST reply with an <mrbresponse> with status code 400. If a syntactically correct request fails because the request also includes any attribute/element the Media Server doesn't understand, the Media Server MUST reply with an <mrbresponse> with status code 420. If a syntactically correct request fails because the MRB wants to create a new subscription, but the provided unique 'id' for the subscription already exists, the Media Server MUST reply with an <mrbresponse> with status code 406. If a syntactically correct request fails because the MRB wants to update/remove a subscription that doesn't exist, the Media Server MUST reply with an <mrbresponse> with status code 404. If the Media

Server is unable to accept a request for any other reason (e.g., the MRB has no more resources to fulfill the request), the Media Server MUST reply with an <mrbresponse> with status code 401/402/403, depending on the action the MRB provided in its request:

- action='create' --> 401;
- action='update' --> 402;
- action='remove' --> 403;

A response to a subscription request that has a status code of 200 indicates that the request is successful. The response MAY also contain a <subscription> child that describes the subscription. The <subscription> child MAY contain 'expires', 'minfrequency', and 'maxfrequency' values even if they were not contained in the request.

The Media Server can choose to change the suggested 'expires', 'minfrequency', and 'maxfrequency' values provided by the MRB in its <mrbrequest> if it considers them unacceptable (e.g., the requested frequency range is too high). In such a case, the response MUST contain a <subscription> element describing the subscription as the Media Server accepted it, and the Media Server MUST include in the <subscription> element all of those values that it modified relative to the request, to inform the MRB about the change.

5.1.5. <mrbnotification>

The <mrbnotification> element is included in a request from a Media Server to an MRB to provide the details relating to current status. The Media Server will inform the MRB of its current status as defined by the information in the <subscription> element. Updates are sent using the <mrbnotification> element.

The <mrbnotification> element has the following attributes:

id: indicates a unique token representing the session between the MRB and the Media Server and is the same as the one appearing in the <subscription> element. The attribute MUST be present.

seqnumber: indicates a sequence number to be used in conjunction with the subscription session ID to identify a specific notification update. The first notification update MUST contain a non-zero number 'seqnumber', and subsequent notification updates MUST contain a higher number than the previous 'seqnumber' value. If a subsequent 'seqnumber' is not higher, the situation should be

considered an error by the entity receiving the notification update. How the receiving entity deals with this situation is implementation specific. The attribute MUST be present.

It's important to point out that the 'seqnumber' that appears in an <mrbnotification> is not related to the 'seqnumber' appearing in a <subscription>. In fact, the latter is associated with subscriptions and would increase at every command issued by the MRB, while the former is associated with the asynchronous notifications the Media Server would trigger according to the subscription and as such would increase at every notification message to enable the MRB to keep track of them.

The following sub-sections provide details of the child elements that make up the contents of the <mrbnotification> element.

5.1.5.1. <media-server-id>

The <media-server-id> element provides a unique system-wide identifier for a Media Server instance. The element MUST be present and MUST be chosen such that it is extremely unlikely that two different Media Servers would present the same id to a given MRB.

5.1.5.2. <supported-packages>

The <supported-packages> element provides the list of Media Control Channel packages supported by the Media Server. The element MAY be present.

The <supported-packages> element has no attributes.

The <supported-packages> element has a single child element:

<package>: Gives the name of a package supported by the Media Server. The <package> element has a single attribute, 'name', which provides the name of the supported Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230].

5.1.5.3. <active-rtp-sessions>

The <active-rtp-sessions> element provides information detailing the current active Real-time Transport Protocol (RTP) sessions. The element MAY be present.

The <active-rtp-sessions> element has no attributes.

The <active-rtp-sessions> element has a single child element:

<rtp-codec>: Describes a supported codec and the number of active sessions using that codec. The <rtp-codec> element has one attribute. The value of the attribute, 'name', is a media type (which can include parameters per [RFC6381]). The <rtp-codec> element has two child elements. The child element <decoding> has as content the decimal number of RTP sessions being decoded using the specified codec, and the child element <encoding> has as content the decimal number of RTP sessions being encoded using the specified codec.

5.1.5.4. <active-mixer-sessions>

The <active-mixer-sessions> element provides information detailing the current active mixed RTP sessions. The element MAY be present.

The <active-mixer-sessions> element has no attributes.

The <active-mixer-sessions> element has a single child element:

<active-mix>: Describes a mixed active RTP session. The <active-mix> element has one attribute. The value of the attribute, 'conferenceid', is the name of the mix. The <active-mix> element has one child element. The child element, <rtp-codec>, contains the same information relating to RTP sessions as that defined in Section 5.1.5.3. The element MAY be present.

5.1.5.5. <non-active-rtp-sessions>

The <non-active-rtp-sessions> element provides information detailing the currently available inactive RTP sessions, that is, how many more RTP streams this Media Server can support. The element MAY be present.

The <non-active-rtp-sessions> element has no attributes.

The <non-active-rtp-sessions> element has a single child element:

<rtp-codec>: Describes a supported codec and the number of non-active sessions for that codec. The <rtp-codec> element has one attribute. The value of the attribute, 'name', is a media type (which can include parameters per [RFC6381]). The <rtp-codec> element has two child elements. The child element <decoding> has as content the decimal number of RTP sessions

available for decoding using the specified codec, and the child element <encoding> has as content the decimal number of RTP sessions available for encoding using the specified codec.

5.1.5.6. <non-active-mixer-sessions>

The <non-active-mixer-sessions> element provides information detailing the current inactive mixed RTP sessions, that is, how many more mixing sessions this Media Server can support. The element MAY be present.

The <non-active-mixer-sessions> element has no attributes.

The <non-active-mixer-sessions> element has a single child element:

<non-active-mix>: Describes available mixed RTP sessions. The <non-active-mix> element has one attribute. The value of the attribute, 'available', is the number of mixes that could be used using that profile. The <non-active-mix> element has one child element. The child element, <rtp-codec>, contains the same information relating to RTP sessions as that defined in Section 5.1.5.5. The element MAY be present.

5.1.5.7. <media-server-status>

The <media-server-status> element provides information detailing the current status of the Media Server. The element MUST be present. It can return one of the following values:

active: Indicates that the Media Server is available for service.

deactivated: Indicates that the Media Server has been withdrawn from service, and as such requests should not be sent to it before it becomes 'active' again.

unavailable: Indicates that the Media Server continues to process past requests but cannot accept new requests, and as such should not be contacted before it becomes 'active' again.

The <media-server-status> element has no attributes.

The <media-server-status> element has no child elements.

5.1.5.8. <supported-codecs>

The <supported-codecs> element provides information detailing the current codecs supported by a Media Server and associated actions. The element MAY be present.

The <supported-codecs> element has no attributes.

The <supported-codecs> element has a single child element:

<supported-codec>: Has a single attribute, 'name', which provides the name of the codec about which this element provides information. A valid value is a media type that, depending on its definition, can include additional parameters (e.g., [RFC6381]). The <supported-codec> element then has a further child element, <supported-codec-package>. The <supported-codec-package> element has a single attribute, 'name', which provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the codec support applies. The <supported-codec-package> element has zero or more <supported-action> children, each one of which describes an action that a Media Server can apply to this codec:

* 'decoding', meaning a decoder for this codec is available;

* 'encoding', meaning an encoder for this codec is available;

* 'passthrough', meaning the Media Server is able to pass a stream encoded using that codec through, without re-encoding.

5.1.5.9. <application-data>

The <application-data> element provides an arbitrary string of characters as application-level data. This data is meant to only have meaning at the application-level logic and as such is not otherwise restricted by this specification. The set of allowed characters is the same as those in XML (viz., tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646 [ISO.10646.2012] (see also Section 2.2 of <http://www.w3.org/TR/xml/>)). The element MAY be present.

The <application-data> element has no attributes.

The <application-data> element has no child elements.

5.1.5.10. <file-formats>

The <file-formats> element provides a list of file formats supported for the purpose of playing media. The element MAY be present.

The <file-formats> element has no attributes.

The <file-formats> element has zero of more the following child elements:

<supported-format>: Has a single attribute, 'name', which provides the type of file format that is supported. A valid value is a media type that, depending on its definition, can include additional parameters (e.g., [RFC6381]). The <supported-format> element then has a further child element, <supported-file-package>. The <supported-file-package> element provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the file format support applies.

5.1.5.11. <max-prepared-duration>

The <max-prepared-duration> element provides the maximum amount of time a media dialog will be kept in the prepared state before timing out (see Section 4.4.2.2.6 of RFC 6231 [RFC6231]. The element MAY be present.

The <max-prepared-duration> element has no attributes.

The <max-prepared-duration> element has a single child element:

<max-time>: Has a single attribute, 'max-time-seconds', which provides the amount of time in seconds that a media dialog can be in the prepared state. The <max-time> element then has a further child element, <max-time-package>. The <max-time-package> element provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the time period applies.

5.1.5.12. <dtmf-support>

The <dtmf-support> element specifies the supported methods to detect Dual-Tone Multi-Frequency (DTMF) tones and to generate them. The element MAY be present.

The <dtmf-support> element has no attributes.

The <dtmf-support> element has zero of more of the following child elements:

<detect>: Indicates the support for DTMF detection. The <detect> element has no attributes. The <detect> element then has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes: 'name' and 'package'. The 'name' attribute provides the type of DTMF being used, and it can only be a case-insensitive string containing either 'RFC4733' [RFC4733] or 'Media' (detecting tones as signals from the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the DTMF type applies.

<generate>: Indicates the support for DTMF generation. The <generate> element has no attributes. The <generate> element then has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes: 'name' and 'package'. The 'name' attribute provides the type of DTMF being used, and it can only be a case-insensitive string containing either 'RFC4733' [RFC4733] or 'Media' (generating tones as signals in the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the DTMF type applies.

<passthrough>: Indicates the support for passing DTMF through without re-encoding. The <passthrough> element has no attributes. The <passthrough> element then has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes: 'name' and 'package'. The 'name' attribute provides the type of DTMF being used, and it can only be a case-insensitive string containing either 'RFC4733' [RFC4733] or 'Media' (passing tones as signals through the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the DTMF type applies.

5.1.5.13. <mixing-modes>

The <mixing-modes> element provides information about the support for audio and video mixing of a Media Server, specifically a list of supported algorithms to mix audio and a list of supported video presentation layouts. The element MAY be present.

The <mixing-modes> element has no attributes.

The <mixing-modes> element has zero or more of the following child elements:

<audio-mixing-modes>: Describes the available algorithms for audio mixing. The <audio-mixing-modes> element has no attributes. The <audio-mixing-modes> element has one child element. The child element, <audio-mixing-mode>, contains a specific available algorithm. Valid values for the <audio-mixing-mode> element are algorithm names, e.g., 'nbest' and 'controller' as defined in [RFC6505]. The element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the algorithm support applies.

<video-mixing-modes>: Describes the available video presentation layouts and the supported functionality related to video mixing. The <video-mixing-modes> element has two attributes: 'vas' and 'activespeakermix'. The 'vas' attribute is of type boolean with a value of 'true' indicating that the Media Server supports automatic Voice Activated Switching. The 'activespeakermix' is of type boolean with a value of 'true' indicating that the Media Server is able to prepare an additional video stream for the loudest speaker participant without its contribution. The <video-mixing-modes> element has one child element. The child element, <video-mixing-mode>, contains the name of a specific video presentation layout. The name may refer to one of the predefined video layouts defined in the XCON conference information data model [RFC6501], or to non-XCON layouts as well, as long as they are properly prefixed according to the schema they belong to. The <video-mixing-mode> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the algorithm support applies.

5.1.5.14. <supported-tones>

The <supported-tones> element provides information about which tones a Media Server is able to play and recognize. In particular, the support is reported by referring to both support for country codes (ISO 3166-1 [ISO.3166-1]) and supported functionality (ITU-T Recommendation Q.1950 [ITU-T.Q.1950]). The element MAY be present.

The <supported-tones> element has no attributes.

The <supported-tones> element has zero or more of the following child elements:

<supported-country-codes>: Describes the supported country codes with respect to tones. The <supported-country-codes> element has no attributes. The <supported-country-codes> element has one child element. The child element, <country-code>, reports support for a specific country code, compliant with the ISO 3166-1 [ISO.3166-1] specification. The <country-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], in which the tones from the specified country code are supported.

<supported-h248-codes>: Describes the supported H.248 codes with respect to tones. The <supported-h248-codes> element has no attributes. The <supported-h248-codes> element has one child element. The child element, <h248-code>, reports support for a specific H.248 code, compliant with the ITU-T Recommendation Q.1950 [ITU-T.Q.1950] specification. The codes can be either specific (e.g., cg/dt to only report the Dial Tone from the Call Progress Tones package) or generic (e.g., cg/* to report all the tones from the Call Progress Tones package), using wildcards. The <h248-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], in which the specified codes are supported.

5.1.5.15. <file-transfer-modes>

The <file-transfer-modes> element allows the Media Server to specify which scheme names are supported for transferring files to a Media Server for each Media Control Channel Framework package type, for example, whether the Media Server supports fetching resources via HTTP, HTTPS, NFS, etc. The element MAY be present.

The <file-transfer-modes> element has no attributes.

The <file-transfer-modes> element has a single child element:

<file-transfer-mode>: Has two attributes: 'name' and 'package'. The 'name' attribute provides the scheme name of the protocol that can be used for file transfer (e.g., HTTP, HTTPS, NFS, etc.); the value of the attribute is case insensitive. The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the specification in the related IANA registry (e.g., "msc-ivr/1.0"), for which the scheme name applies.

It is important to point out that this element provides no information about whether or not the Media Server supports any flavor of live streaming: for instance, a value of "HTTP" for the IVR (Interactive Voice Response) Package would only mean the 'http' scheme makes sense to the Media Server within the context of that package. Whether or not the Media Server can make use of HTTP to only fetch resources, or also to attach an HTTP live stream to a call, is to be considered implementation specific to the Media Server and irrelevant to the Application Server and/or MRB. Besides, the Media Server supporting a scheme does not imply that it also supports the related secure versions: for instance, if the Media Server supports both HTTP and HTTPS, both the schemes will appear in the element. A lack of the "HTTPS" value would need to be interpreted as a lack of support for the 'https' scheme.

5.1.5.16. <asr-tts-support>

The <asr-tts-support> element provides information about the support for Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) functionality in a Media Server. The functionality is reported by referring to the supported languages (using ISO 639-1 [ISO.639.2002] codes) regarding both ASR and TTS. The element MAY be present.

The <asr-tts-support> element has no attributes.

The <asr-tts-support> element has zero or more of the following child elements:

<asr-support>: Describes the available languages for ASR. The <asr-support> element has no attributes. The <asr-support> element has one child element. The child element, <language>, reports that the Media Server supports ASR for a specific language. The <language> element has a single attribute, 'xml:lang'. The attribute 'xml:lang' contains the ISO 639-1 [ISO.639.2002] code of the supported language.

<tts-support>: Describes the available languages for TTS. The <tts-support> element has no attributes. The <tts-support> element has one child element. The child element, <language>, reports that the Media Server supports TTS for a specific language. The <language> element has a single attribute, 'xml:lang'. The attribute 'xml:lang' contains the ISO 639-1 [ISO.639.2002] code of the supported language.

5.1.5.17. <vxml-support>

The <vxml-support> element specifies if the Media Server supports VoiceXML (VXML) and, if it does, through which protocols the support is exposed (e.g., via the control framework, RFC 4240 [RFC4240], or RFC 5552 [RFC5552]). The element MAY be present.

The <vxml-support> element has no attributes.

The <vxml-support> element has a single child element:

<vxml-mode>: Has two attributes: 'package' and 'support'. The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the VXML support applies. The 'support' attribute provides the type of VXML support provided by the Media Server (e.g., RFC 5552 [RFC5552], RFC 4240 [RFC4240], or the IVR Package [RFC6231]), and valid values are case-insensitive RFC references (e.g., "rfc6231" to specify that the Media Server supports VoiceXML as provided by the IVR Package [RFC6231]).

The presence of at least one <vxml-mode> child element would indicate that the Media Server does support VXML as specified by the child element itself. An empty <vxml> element would otherwise indicate that the Media Server does not support VXML at all.

5.1.5.18. <media-server-location>

The <media-server-location> element provides information about the civic location of a Media Server. Its description makes use of the Civic Address Schema standardized in RFC 5139 [RFC5139]. The element MAY be present. More precisely, this section is entirely optional, and it's implementation specific to fill it with just the details each implementer deems necessary for any optimization that may be needed.

The <media-server-location> element has no attributes.

The <media-server-location> element has a single child element:

<civicAddress>: Describes the civic address location of the Media Server, whose representation refers to Section 4 of RFC 5139 [RFC5139].

5.1.5.19. <label>

The <label> element allows a Media Server to declare a piece of information that will be understood by the MRB. For example, the Media Server can declare if it's a blue or green one. It's a string to allow arbitrary values to be returned to allow arbitrary classification. The element MAY be present.

The <label> element has no attributes.

The <label> element has no child elements.

5.1.5.20. <media-server-address>

The <media-server-address> element allows a Media Server to provide a direct SIP URI where it can be reached (e.g., the URI that the Application Server would call in order to set up a Control Channel and relay SIP media dialogs). The element MAY be present.

The <media-server-address> element has no attributes.

The <media-server-address> element has no child elements.

5.1.5.21. <encryption>

The <encryption> element allows a Media Server to declare support for encrypting RTP media streams using RFC 3711 [RFC3711]. The element MAY be present. If the element is present, then the Media Server supports DTLS-SRTP (a Secure Real-time Transport Protocol (SRTP) extension for Datagram Transport Layer Security (DTLS)) [RFC5763].

The <encryption> element has no attributes.

The <encryption> element has no child elements.

5.2. Media Service Resource Consumer Interface

The Media Server Consumer interface provides the ability for clients of an MRB, such as Application Servers, to request an appropriate Media Server to satisfy specific criteria. This interface allows a client to pass detailed meta-information to the MRB to help select an appropriate Media Server. The MRB is then able to make an informed

decision and provide the client with an appropriate Media Server resource. The MRB Consumer interface includes both 1) the In-line Aware MRB Mode (IAMM), which uses the Session Initiation Protocol (SIP) and 2) the Query mode, which uses the Hypertext Transfer Protocol (HTTP) [RFC2616]. The MRB Consumer interface does not include the In-line Unaware Mode (IUMM), which is further explained in Section 5.3. The following sub-sections provide guidance on using the Consumer interface, which is represented by the 'application/mrb-consumer+xml' media type in Section 11, with HTTP and SIP.

5.2.1. Query Mode/HTTP Consumer Interface Usage

An appropriate interface for such a 'query' style interface is in fact an HTTP usage. Using HTTP and XML combined reduces complexity and encourages the use of common tools that are widely available in the industry today. The following information explains the primary operations required to request and then receive information from an MRB, by making use of HTTP [RFC2616] and HTTPS [RFC2818] as transport for a query for a media resource, and the appropriate response.

The media resource query, as defined by the <mediaResourceRequest> element from Section 11, MUST be carried in the body of an HTTP/HTTPS POST request. The media type contained in the HTTP/HTTPS request/response MUST be 'application/mrb-consumer+xml'. This value MUST be reflected in the appropriate HTTP headers, such as 'Content-Type' and 'Accept'. The body of the HTTP/HTTPS POST request MUST only contain an <mrbconsumer> root element with only one child <mediaResourceRequest> element as defined in Section 11.

The media resource response to a query, as defined by the <mediaResourceResponse> element from Section 11, MUST be carried in the body of an HTTP/HTTPS 200 response to the original HTTP/HTTPS POST request. The media type contained in the HTTP/HTTPS request/response MUST be 'application/mrb-consumer+xml'. This value MUST be reflected in the appropriate HTTP headers, such as 'Content-Type' and 'Accept'. The body of the HTTP/HTTPS 200 response MUST only contain an <mrbconsumer> root element with only one child <mediaResourceResponse> element as defined in Section 11.

When an Application Server wants to release previously awarded media resources granted through a prior request/response exchange with an MRB, it will send a new request with an <action> element with value 'remove', as described in Section 5.2.3 ("Consumer Interface Lease Mechanism").

5.2.2. In-Line Aware Mode/SIP Consumer Interface Usage

This document provides a complete toolkit for MRB deployment that includes the ability to interact with an MRB using SIP for the Consumer interface. The following information explains the primary operations required to request and then receive information from an MRB, by making use of SIP [RFC3261] as transport for a request for media resources, and the appropriate response when using IAMM as the mode of operation (as discussed in Section 5.2.2.1).

The use of IAMM, besides having the MRB select appropriate media resources on behalf of a client application, includes setting up either a Control Framework Control Channel between an Application Server and one of the Media Servers (Section 5.2.2.1) or a media dialog session between an Application Server and one of the Media Servers (Section 5.2.2.2). Note that in either case the SIP URIs of the selected Media Servers are made known to the requesting Application Server in the SIP 200 OK response by means of one or more <media-server-address> child elements in the <response-session-info> element (Section 5.2.6).

5.2.2.1. IAMM and Setting Up a Control Framework Control Channel

The media resource request information, as defined by the <mediaResourceRequest> element from Section 11, is carried in a SIP INVITE request. The INVITE request will be constructed as it would have been to connect to a Media Server, as defined by the Media Control Channel Framework [RFC6230]. It should be noted that this specification does not exclude the use of an offerless INVITE as defined in RFC 3261 [RFC3261]. Using offerless INVITE messages to an MRB can potentially cause confusion when applying resource selection algorithms, and an MRB, like any other SIP device, can choose to reject with a 4xx response. For an offerless INVITE to be treated appropriately, additional contextual information would need to be provided with the request; this is out of scope for this document. The following additional steps MUST be followed when using the Consumer interface:

- The Consumer client will include a payload in the SIP INVITE request of type 'multipart/mixed' [RFC2046]. One of the parts to be included in the 'multipart/mixed' payload MUST be the 'application/sdp' format, which is constructed as specified in the Media Control Channel Framework [RFC6230].

- Another part of the 'multipart/mixed' payload MUST be of type 'application/mrb-consumer+xml', as specified in this document and defined in Section 11. The body part MUST be an XML document without prolog and whose root element is <mediaResourceRequest>.

- The INVITE request will then be dispatched to the MRB, as defined by [RFC6230].

On receiving a SIP INVITE request containing the multipart/mixed payload as specified previously, the MRB will complete a number of steps to fulfill the request. It will:

- Extract the multipart MIME payload from the SIP INVITE request. It will then use the contextual information provided by the client in the 'application/mrb-consumer+xml' part to determine which Media Server (or Media Servers, if more than one is deemed to be needed) should be selected to service the request.

- Extract the 'application/sdp' part from the payload and use it as the body of a new SIP INVITE request for connecting the client to one of the selected Media Servers, as defined in the Media Channel Control Framework [RFC6230]. The policy the MRB follows to pick a specific Media Server out of the Media Servers it selects is implementation specific and out of scope for this document. It is important to configure the SIP elements between the MRB and the Media Server in such a way that the INVITE will not fork. In the case of a failure in reaching the chosen Media Server, the MRB SHOULD proceed to the next one, if available.

If none of the available Media Servers can be reached, the MRB MUST reply with a SIP 503 error message that includes a Retry-After header with a non-zero value. The Application Server MUST NOT attempt to set up a new session before the time that the MRB asked it to wait has passed.

If at least one Media Server is reachable, the MRB acts as a Back-to-Back User Agent (B2BUA) that extracts the 'application/mrb-consumer+xml' information from the SIP INVITE request and then sends a corresponding SIP INVITE request to the Media Server it has selected, to negotiate a Control Channel as defined in the Media Channel Control Framework [RFC6230].

In the case of a failure in negotiating the Control Channel with the Media Server, the MRB SHOULD proceed to the next one, if available, as explained above. If none of the available Media Servers can be reached, or the negotiations of the Control Channel with all of them fail, the MRB MUST reply with a SIP 503 error message that includes a Retry-After header with a non-zero value. The Application Server MUST NOT attempt to set up a new session before the time that the MRB asked it to wait has expired.

Once the MRB receives the SIP response from the selected media resource (i.e., Media Server), it will in turn respond to the requesting client (i.e., Application Server).

The media resource response generated by an MRB to a request, as defined by the <mediaResourceResponse> element from Section 11, MUST be carried in the payload of a SIP 200 OK response to the original SIP INVITE request. The SIP 200 OK response will be constructed as it would have been to connect from a Media Server, as defined by the Media Control Channel Framework [RFC6230]. The following additional steps MUST be followed when using the Consumer interface:

- Include a payload in the SIP 200 response of type 'multipart/mixed' as per RFC 2046 [RFC2046]. One of the parts to be included in the 'multipart/mixed' payload MUST be the 'application/sdp' format, which is constructed as specified in the Media Control Channel Framework [RFC6230] and based on the incoming response from the selected media resource.

- Another part of the 'multipart/mixed' payload MUST be of type 'application/mrb-consumer+xml', as specified in this document and defined in Section 11. Only the <mediaResourceResponse> and its child elements can be included in the payload.

- The SIP 200 response will then be dispatched from the MRB.

- A SIP ACK to the 200 response will then be sent back to the MRB.

Considering that the use of SIP as a transport for Consumer transactions may result in failure, the IAMM relies on a successful INVITE transaction to address the previously discussed sequence (using the 'seq' XML element) increment mechanism. This means that if the INVITE is unsuccessful for any reason, the Application Server MUST use the same 'seq' value as previously used for the next Consumer request that it may want to send to the MRB for the same session.

An MRB implementation may be programmed to conclude that the requested resources are no longer needed when it receives a SIP BYE from the Application Server or Media Server that concludes the SIP dialog that initiated the request, or when the lease (Section 5.2.3) interval expires.

5.2.2.2. IAMM and Setting Up a Media Dialog

This scenario is identical to the description in the previous section for setting up a Control Framework Control Channel, with the exception that the application/sdp payload conveys content appropriate for setting up the media dialog to the media resource, as per RFC 3261 [RFC3261], instead of setting up a Control Channel.

5.2.3. Consumer Interface Lease Mechanism

The Consumer interface defined in Sections 5.2 and 11 allows a client to request an appropriate media resource based on information included in the request (either an HTTP POST or SIP INVITE message). In the case of success, the response that is returned to the client MUST contain a <response-session-info> element in either the SIP 200 or HTTP 200 response. The success response contains the description of certain resources that have been reserved to a specific Consumer client in a (new or revised) "resource session", which is identified in the <response-session-info>. The resource session is a "lease", in that the reservation is scheduled to expire at a particular time in the future, releasing the resources to be assigned for other uses. The lease may be extended or terminated earlier by future Consumer client requests that identify and reference a specific resource session.

Before delving into the details of such a lease mechanism, it is worth clarifying its role within the context of the Consumer interface. As explained in Section 5.1, the knowledge the MRB has of the resources of all the Media Servers it is provisioned to manage is not real-time. How an MRB actually manages such resources is implementation specific -- for example, an implementation may choose to have the MRB keeping track and state of the allocated resources, or simply rely on the Media Servers themselves to provide the information using the Publish interface. Further information may also be inferred by the signaling, in the case where an MRB is in the path of media dialogs.

The <mediaResourceResponse> element returned from the MRB contains a <response-session-info> element if the request is successful. The <response-session-info> element has zero or more of the following child elements, which provide the appropriate resource session information:

- <session-id> is a unique identifier that enables a Consumer client and MRB to correlate future media resource requests related to an initial media resource request. The <session-id> MUST be included in all future related requests (see the <session-id> paragraph later in this section, where constructing a subsequent request is discussed).

- <seq> is a numeric value returned to the Consumer client. On issuing any future requests related to the media resource session (as determined by the <session-id> element), the Consumer client MUST increment the value returned in the <seq> element and include it in the request (see the <seq> paragraph later in this section, where constructing a subsequent request is discussed). Its value is a non-negative integer that MUST be limited within the 0..2^31-1 range.

- <expires> provides a value indicating the number of seconds that the request for media resources is deemed alive. The Consumer client should issue a refresh of the request, as discussed later in this section, if the expiry is due to fire and the media resources are still required.

- <media-server-address> provides information representing an assigned Media Server. More instances of this element may appear should the MRB assign more Media Servers to a Consumer request.

The <mediaResourceRequest> element is used in subsequent Consumer interface requests if the client wishes to manipulate the session. The Consumer client MUST include the <session-info> element, which enables the receiving MRB to determine an existing media resource allocation session. The <session-info> element has the following child elements, which provide the appropriate resource session information to the MRB:

- <session-id> is a unique identifier that allows a Consumer client to indicate the appropriate existing media resource session to be manipulated by the MRB for this request. The value was provided by the MRB in the initial request for media resources, as discussed earlier in this section (<session-id> element included as part of the <session-info> element in the initial <mediaResourceResponse>).

- <seq> is a numeric value returned to the Consumer client in the initial request for media resources, as discussed earlier in this section (<seq> element included as part of the <session-info> element in the initial <mediaResourceResponse>). On issuing any future requests related to the specific media resource session (as determined by the <session-id> element), the Consumer client MUST increment the value returned in the <seq> element from the initial response (contained in the <mediaResourceResponse>) for every new request. The value of the <seq> element in requests acts as a counter and when used in conjunction with the unique <session-id> allows for unique identification of a request. As anticipated before, the <seq> value is limited to the 0..2^31-1 range: in the unlikely case that the counter increases to reach the highest allowed value, the <seq> value MUST be set to 0. The first numeric value for the <seq> element is not meant to be '1' but SHOULD be generated randomly by the MRB: this is to reduce the chances of a malicious MRB disrupting the session created by this MRB, as explained in Section 12.

- <action> provides the operation to be carried out by the MRB on receiving the request:

  * The value of 'update' is a request by the Consumer client to update the existing session on the MRB with alternate media resource requirements. If the requested resource information is identical to the existing MRB session, the MRB will attempt a session refresh. If the information has changed, the MRB will attempt to update the existing session with the new information. If the operation is successful, the 200 status code in the response is returned in the status attribute of the <mediaResourceResponseType> element. If the operation is not successful, a 409 status code in the response is returned in the status attribute of the <mediaResourceResponseType> element.

  * The value of 'remove' is a request by the Consumer client to remove the session on the MRB. This provides a mechanism for Consumer clients to release unwanted resources before they expire. If the operation is successful, a 200 status code in the response is returned in the status attribute of the <mediaResourceResponseType> element. If the operation is not successful, a 410 status code in the response is returned in the status attribute of the <mediaResourceResponseType> element.

Omitting the 'action' attribute means requesting a new set of resources.

When used with HTTP, the <session-info> element MUST be included in an HTTP POST message (as defined in [RFC2616]). When used with SIP, the <session-info> element MUST instead be included in either a SIP INVITE or a SIP re-INVITE (as defined in [RFC3261]), or in a SIP UPDATE (as defined in [RFC3311]) request: in fact, any SIP dialog, be it a new or an existing one, can be exploited to carry leasing information, and as such new SIP INVITE messages can update other leases as well as request a new one.

With IAMM, the Application Server or Media Server will eventually send a SIP BYE to end the SIP session, whether it was for a Control Channel or a media dialog. That BYE contains no Consumer interface lease information.

5.2.4. <mrbconsumer>

This section defines the XML elements for the Consumer interface. The formal XML schema definition for the Consumer interface can be found in Section 11.

The root element is <mrbconsumer>. All other XML elements (requests, responses) are contained within it. The MRB Consumer interface request element is detailed in Section 5.2.5.1. The MRB Consumer interface response element is detailed in Section 5.2.6.1.

The <mrbconsumer> element has the following attributes:

version: a token specifying the mrb-consumer package version. The value is fixed as '1.0' for this version of the package. The attribute MUST be present.

The <mrbconsumer> element may have zero or more children of one of the following child element types:

<mediaResourceRequest> for sending a Consumer request. See Section 5.2.5.1.

<mediaResourceResponse> for sending a Consumer response. See Section 5.2.6.1.

5.2.5. Media Service Resource Request

This section provides the element definitions for use in Consumer interface requests. The requests are carried in the <mediaResourceRequest> element.

5.2.5.1. <mediaResourceRequest>

The <mediaResourceRequest> element provides information for clients wishing to query an external MRB entity. The <mediaResourceRequest> element has a single mandatory attribute, 'id': this attribute contains a random identifier, generated by the client, that will be included in the response in order to map it to a specific request. The <mediaResourceRequest> element has <generalInfo>, <ivrInfo>, and <mixerInfo> as child elements. These three elements are used to describe the requirements of a client requesting a Media Server and are covered in Sections 5.2.5.1.1, 5.2.5.1.2, and 5.2.5.1.3, respectively.

5.2.5.1.1. <generalInfo>

The <generalInfo> element provides general Consumer request information that is neither IVR specific nor mixer specific. This includes session information that can be used for subsequent requests as part of the leasing mechanism described in Section 5.2.3. The following sub-sections describe the <session-info> and <packages> elements, as used by the <generalInfo> element.

5.2.5.1.1.1. <session-info>

The <session-info> element is included in Consumer requests when an update is being made to an existing media resource session. The ability to change and remove an existing media resource session is described in more detail in Section 5.2.3. The element MAY be present.

The <session-info> element has no attributes.

The <session-info> element has zero or more of the following child elements:

<session-id>: A unique identifier that explicitly references an existing media resource session on the MRB. The identifier is included to update the existing session and is described in more detail in Section 5.2.3.

<seq>: Used in association with the <session-id> element in a subsequent request to update an existing media resource session on an MRB. The <seq> number is incremented from its original value returned in response to the initial request for media resources. Its value is a non-negative integer that MUST be limited within the 0..2^31-1 range. In the unlikely case that the counter increases to reach the highest allowed value, the <seq> value MUST be set to 0. More information about its use is provided in Section 5.2.3.

<action>: Provides the operation that should be carried out on an existing media resource session on an MRB:

* The value of 'update' instructs the MRB to attempt to update the existing media resource session with the information contained in the <ivrInfo> and <mixerInfo> elements.

* The value of 'remove' instructs the MRB to attempt to remove the existing media resource session. More information on its use is provided in Section 5.2.3.

5.2.5.1.1.2. <packages>

The <packages> element provides a list of Media Control Channel Framework compliant packages that are required by the Consumer client. The element MAY be present.

The <packages> element has no attributes.

The <packages> element has a single child element:

<package>: Contains a string representing the Media Control Channel Framework package required by the Consumer client. The <package> element can appear multiple times. A valid value is a Control Package name compliant with Section 13.1.1 of [RFC6230].

5.2.5.1.2. <ivrInfo>

The <ivrInfo> element provides information for general Consumer request information that is IVR specific. The following sub-sections describe the elements of the <ivrInfo> element: <ivr-sessions>, <file-formats>, <dtmf>, <tones>, <asr-tts>, <vxml>, <location>, <encryption>, <application-data>, <max-prepared-duration>, and <file-transfer-modes>.

5.2.5.1.2.1. <ivr-sessions>

The <ivr-sessions> element indicates the number of IVR sessions that a Consumer client requires from a media resource. The element MAY be present.

The <ivr-sessions> element has no attributes.

The <ivr-sessions> element has a single child element:

<rtp-codec>: Describes a required codec and the number of sessions using that codec. The <rtp-codec> element has one attribute. The value of the attribute, 'name', is a media type (which can include parameters per [RFC6381]). The <rtp-codec> element has two child elements. The child element <decoding> contains the number of RTP sessions required for decoding using the specified codec, and the child element <encoding> contains the number of RTP sessions required for encoding using the specified codec.

5.2.5.1.2.2. <file-formats>

The <file-formats> element provides a list of file formats required for the purpose of playing media. It should be noted that this element describes media types and might better have been named "media-formats", but due to existing implementations the name "file-formats" is being used. The element MAY be present.

The <file-formats> element has no attributes.

The <file-formats> element has a single child element:

<required-format>: Has a single attribute, 'name', which provides the type of file format that is required. A valid value is a media type that, depending on its definition, can include additional parameters (e.g., [RFC6381]). The <required-format> element then has a further child element, <required-file-package>. The <required-file-package> element has a single attribute, 'required-file-package-name', which contains the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the file format support applies.

5.2.5.1.2.3. <dtmf>

The <dtmf> element specifies the required methods to detect DTMF tones and to generate them. The element MAY be present.

The <dtmf> element has no attributes.

The <dtmf> element has zero or more of the following child elements:

<detect>: Indicates the required support for DTMF detection. The <detect> element has no attributes. The <detect> element has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes: 'name' and 'package'. The 'name' attribute provides the type of DTMF required and is a case-insensitive string containing either 'RFC4733' [RFC4733] or 'Media' (detecting tones as signals from the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the DTMF type applies.

<generate>: Indicates the required support for DTMF generation. The <generate> element has no attributes. The <generate> element has a single child element, <dtmf-type>. The <dtmf-type> element has two attributes: 'name' and 'package'. The 'name' attribute provides the type of DTMF required and is a case-insensitive string containing either 'RFC4733' [RFC4733] or 'Media' (generating tones as signals in the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the DTMF type applies.

<passthrough>: Indicates the required support for passing DTMF through without re-encoding. The <passthrough> element has no attributes. The <passthrough> element then has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes: 'name' and 'package'. The 'name' attribute provides the type of DTMF required and is a case-insensitive string containing either 'RFC4733' [RFC4733] or 'Media' (passing tones as signals through the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the DTMF type applies.

5.2.5.1.2.4. <tones>

The <tones> element provides requested tones that a Media Server must support for IVR. In particular, the request refers to both support for country codes (ISO 3166-1 [ISO.3166-1]) and requested functionality (ITU-T Recommendation Q.1950 [ITU-T.Q.1950]). The element MAY be present.

The <tones> element has no attributes.

The <tones> element has zero or more of the following child elements:

<country-codes>: Describes the requested country codes in relation to tones. The <country-codes> element has no attributes. The <country-codes> element has one child element. The child element, <country-code>, requests a specific country code, compliant with the ISO 3166-1 [ISO.3166-1] specification. The <country-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], in which the tones from the specified country code are requested.

<h248-codes>: Describes the requested H.248 codes in relation to tones. The <h248-codes> element has no attributes. The <h248-codes> element has one child element. The child element, <h248-code>, requests a specific H.248 code, compliant with the ITU-T Recommendation Q.1950 [ITU-T.Q.1950] specification. The codes can be either specific (e.g., cg/dt to only report the Dial Tone from the Call Progress Tones package) or generic (e.g., cg/* to report all the tones from the Call Progress Tones package), using wildcards. The <h248-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], in which the specified codes are requested.

5.2.5.1.2.5. <asr-tts>

The <asr-tts> element requests information about the support for Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) functionality in a Media Server. The functionality is requested by referring to the supported languages (using ISO 639-1 [ISO.639.2002] codes) in relation to both ASR and TTS. The <asr-tts> element has no attributes. The <asr-tts> element has zero or more of the following child elements:

<asr-support>: Describes the available languages for ASR. The <asr-support> element has no attributes. The <asr-support> element has one child element. The child element, <language>,

requests that the Media Server supports ASR for a specific language. The <language> element has a single attribute, 'xml:lang'. The attribute 'xml:lang' contains the ISO 639-1 [ISO.639.2002] code of the supported language.

<tts-support>: Describes the available languages for TTS. The <tts-support> element has no attributes. The <tts-support> element has one child element. The child element, <language>, requests that the Media Server supports TTS for a specific language. The <language> element has a single attribute, 'xml:lang'. The attribute 'xml:lang' contains the ISO 639-1 [ISO.639.2002] code of the supported language.

5.2.5.1.2.6. <vxml>

The <vxml> element specifies if the Consumer client requires VoiceXML and, if so, which protocols are supported (e.g., via the control framework, RFC 4240 [RFC4240], or RFC 5552 [RFC5552]). The element MAY be present.

The <vxml> element has a single child element:

<vxml-mode>: Has two attributes: 'package' and 'require'. The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the VXML support applies. The 'require' attribute specifies the type of VXML support required by the Consumer client (e.g., RFC 5552 [RFC5552], RFC 4240 [RFC4240], or IVR Package [RFC6231]), and valid values are case-insensitive RFC references (e.g., "rfc6231" to specify that the client requests support for VoiceXML as provided by the IVR Package [RFC6231]).

The presence of at least one <vxml> child element would indicate that the Consumer client requires VXML support as specified by the child element itself. An empty <vxml> element would otherwise indicate that the Consumer client does not require VXML support.

5.2.5.1.2.7. <location>

The <location> element requests a civic location for an IVR Media Server. The request makes use of the Civic Address Schema standardized in RFC 5139 [RFC5139]. The element MAY be present. More precisely, this section is entirely optional and is implementation specific in its level of population.

The <location> element has no attributes.

The <location> element has a single child element:

<civicAddress>: Describes the civic address location of the requested Media Server, whose representation refers to Section 4 of RFC 5139 [RFC5139].

5.2.5.1.2.8. <encryption>

The <encryption> element allows a Consumer client to request support for encrypting RTP media streams using RFC 3711 [RFC3711]. The element MAY be present. If the element is present, then the Media Server supports DTLS-SRTP [RFC5763].

The <encryption> element has no attributes.

The <encryption> element has no child elements.

5.2.5.1.2.9. <application-data>

The <application-data> element provides an arbitrary string of characters as IVR application-level data. This data is meant to only have meaning at the application-level logic and as such is not otherwise restricted by this specification. The set of allowed characters is the same as those in XML (viz., tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646 [ISO.10646.2012] (see also Section 2.2 of <http://www.w3.org/TR/xml/>)). The element MAY be present.

The <application-data> element has no attributes.

The <application-data> element has no child elements.

5.2.5.1.2.10. <max-prepared-duration>

The <max-prepared-duration> element indicates the amount of time required by the Consumer client representing media dialog preparation in the system before it is executed. The element MAY be present.

The <max-prepared-duration> element has no attributes.

The <max-prepared-duration> element has a single child element:

<max-time>: Has a single attribute, 'max-time-seconds', which provides the amount of time in seconds that a media dialog can be in the prepared state. The <max-time> element then has a further child element, <max-time-package>. The <max-time-package> element

provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the time period applies.

5.2.5.1.2.11. <file-transfer-modes>

The <file-transfer-modes> element allows the Consumer client to specify which scheme names are required for file transfer to a Media Server for each Media Control Channel Framework package type. For example, does the Media Server support fetching media resources via HTTP, HTTPS, NFS, etc.? The element MAY be present.

The <file-transfer-modes> element has no attributes.

The <file-transfer-modes> element has a single child element:

<file-transfer-mode>: Has two attributes: 'name' and 'package'. The 'name' attribute provides the scheme name of the protocol required for fetching resources: valid values are case-insensitive scheme names (e.g., HTTP, HTTPS, NFS, etc.). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the scheme name applies.

The same considerations relating to file transfer and live streaming are explained further in Section 5.1.5.15 and apply here as well.

5.2.5.1.3. <mixerInfo>

The <mixerInfo> element provides information for general Consumer request information that is mixer specific. The following sub-sections describe the elements of the <mixerInfo> element: <mixers>, <file-formats>, <dtmf>, <tones>, <mixing-modes>, <application-data>, <location>, and <encryption>.

5.2.5.1.3.1. <mixers>

The <mixers> element provides information detailing the required mixed RTP sessions. The element MAY be present.

The <mixers> element has no attributes.

The <mixers> element has a single child element:

<mix>: Describes the required mixed RTP sessions. The <mix> element has one attribute. The value of the attribute, 'users', is the number of participants required in the mix. The <mix> element has

one child element. The child element, <rtp-codec>, contains the same information relating to RTP sessions as that defined in Section 5.1.5.3. The element MAY be present.

5.2.5.1.3.2. <file-formats>

The <file-formats> element provides a list of file formats required by the Consumer client for the purpose of playing media to a mix. The element MAY be present.

The <file-formats> element has no attributes.

The <file-formats> element has a single child element:

<required-format>: Has a single attribute, 'name', which provides the type of file format supported. A valid value is a media type that, depending on its definition, can include additional parameters (e.g., [RFC6381]). The <required-format> element has a child element, <required-file-package>. The <required-file-package> element contains a single attribute, 'required-file-package-name', which contains the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the file format support applies.

5.2.5.1.3.3. <dtmf>

The <dtmf> element specifies the required methods to detect DTMF tones and to generate them in a mix. The element MAY be present.

The <dtmf> element has no attributes.

The <dtmf> element has zero or more of the following child elements:

<detect>: Indicates the required support for DTMF detection. The <detect> element has no attributes. The <detect> element then has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes: 'name' and 'package'. The 'name' attribute provides the type of DTMF being used and is a case-insensitive string containing either 'RFC4733' [RFC4733] or 'Media' (detecting tones as signals from the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the DTMF type applies.

<generate>: Indicates the required support for DTMF generation. The <generate> element has no attributes. The <generate> element has a single child element, <dtmf-type>. The <dtmf-type> element has two attributes: 'name' and 'package'. The 'name' attribute

provides the type of DTMF being used and is a case-insensitive string containing either 'RFC4733' [RFC4733] or 'Media' (generating tones as signals in the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the DTMF type applies.

<passthrough>: Indicates the required support for passing DTMF through without re-encoding. The <passthrough> element has no attributes. The <passthrough> element has a single child element, <dtmf-type>. The <dtmf-type> element has two attributes: 'name' and 'package'. The 'name' attribute provides the type of DTMF being used and is a case-insensitive string containing either 'RFC4733' [RFC4733] or 'Media' (passing tones as signals through the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the DTMF type applies.

5.2.5.1.3.4. <tones>

The <tones> element provides requested tones that a Media Server must support for a mix. In particular, the request refers to both support for country codes (ISO 3166-1 [ISO.3166-1]) and requested functionality (ITU-T Recommendation Q.1950 [ITU-T.Q.1950]). The element MAY be present.

The <tones> element has no attributes.

The <tones> element has zero or more of the following child elements:

<country-codes>: Describes the requested country codes in relation to tones. The <country-codes> element has no attributes. The <country-codes> element has a single child element. The child element, <country-code>, requests a specific country code, compliant with the ISO 3166-1 [ISO.3166-1] specification. The <country-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the specification in the related IANA registry (e.g., "msc-ivr/1.0"), in which the tones from the specified country code are requested.

<h248-codes>: Describes the requested H.248 codes with respect to tones. The <h248-codes> element has no attributes. The <h248-codes> element has a single child element. The child element, <h248-code>, requests a specific H.248 code, compliant with the ITU-T Recommendation Q.1950 [ITU-T.Q.1950] specification. The codes can be either specific (e.g., cg/dt to only report the Dial Tone from the Call Progress Tones package) or generic (e.g.,

cg/* to report all the tones from the Call Progress Tones package), using wildcards. The <h248-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], in which the specified codes are requested.

5.2.5.1.3.5. <mixing-modes>

The <mixing-modes> element requests information relating to support for audio and video mixing, more specifically a list of supported algorithms to mix audio and a list of supported video presentation layouts. The element MAY be present.

The <mixing-modes> element has no attributes.

The <mixing-modes> element has zero or more of the following child elements:

<audio-mixing-modes>: Describes the requested algorithms for audio mixing. The <audio-mixing-modes> element has no attributes. The <audio-mixing-modes> element has one child element. The child element, <audio-mixing-mode>, contains a requested mixing algorithm. Valid values for the <audio-mixing-mode> element are algorithm names, e.g., 'nbest' and 'controller' as defined in [RFC6505]. The element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the algorithm support is requested.

<video-mixing-modes>: Describes the requested video presentation layouts for video mixing. The <video-mixing-modes> element has two attributes: 'vas' and 'activespeakermix'. The 'vas' attribute is of type boolean with a value of 'true' indicating that the Consumer client requires automatic Voice Activated Switching. The 'activespeakermix' attribute is of type boolean with a value of 'true' indicating that the Consumer client requires an additional video stream for the loudest speaker participant without its contribution. The <video-mixing-modes> element has one child element. The child element, <video-mixing-mode>, contains the name of a specific video presentation layout. The name may refer to one of the predefined video layouts defined in the XCON conference information data model, or to non-XCON layouts as well, as long as they are appropriately prefixed. The <video-mixing-mode> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with Section 13.1.1 of [RFC6230], for which the algorithm support is requested.

5.2.5.1.3.6. <application-data>

The <application-data> element provides an arbitrary string of characters as mixer application-level data. This data is meant to only have meaning at the application-level logic and as such is not otherwise restricted by this specification. The set of allowed characters is the same as those in XML (viz., tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646 [ISO.10646.2012] (see also Section 2.2 of <http://www.w3.org/TR/xml/>)). The element MAY be present.

The <application-data> element has no attributes.

The <application-data> element has no child elements.

5.2.5.1.3.7. <location>

The <location> element requests a civic location for a mixer Media Server. The request makes use of the Civic Address Schema standardized in RFC 5139 [RFC5139]. The element MAY be present. More precisely, this section is entirely optional, and it's implementation specific to fill it with just the details each implementer deems necessary for any optimization that may be needed.

The <location> element has no attributes.

The <location> element has a single child element:

<civicAddress>: Describes the civic address location of the requested Media Server, whose representation refers to Section 4 of RFC 5139 [RFC5139].

5.2.5.1.3.8. <encryption>

The <encryption> element allows a Consumer client to request support for encrypting mixed RTP media streams using RFC 3711 [RFC3711]. The element MAY be present. If the element is present, then the Media Server supports DTLS-SRTP [RFC5763].

The <encryption> element has no attributes.

The <encryption> element has no child elements.

5.2.6. Media Service Resource Response

This section provides the element definitions for use in Consumer interface responses. The responses are carried in the <mediaResourceResponse> element.

5.2.6.1. <mediaResourceResponse>

The <mediaResourceResponse> element provides information for clients receiving response information from an external MRB entity.

The <mediaResourceResponse> element has two mandatory attributes: 'id' and 'status'. The 'id' attribute must contain the same value that the client provided in the 'id' attribute in the <mediaResourceRequest> to which the response is mapped. The 'status' attribute indicates the status code of the operation. The following status codes are defined for 'status':

code	description
200	OK
400	Syntax error
405	Wrong sequence number
408	Unable to find Resource
409	Unable to update Resource
410	Unable to remove Resource
420	Unsupported attribute or element

Table 2: <mediaResourceResponse> Status Codes

If a new media resource request made by a client application has been accepted, the MRB MUST reply with a <mediaResourceResponse> with status code 200. The same rule applies whenever a request to update (action='update') or remove (action='remove') an existing transaction can be fulfilled by the MRB.

A media resource request, nevertheless, may fail for several reasons. In such a case, the status codes defined in Table 2 must be used instead. Specifically, if the MRB fails to handle a request due to a syntax error in the request itself (e.g., incorrect XML, violation of

the schema constraints, or invalid values in any of the attributes/elements), the MRB MUST reply with a <mediaResourceResponse> with status code 400. If a syntactically correct request fails because the request also includes any attribute/element the MRB doesn't understand, the MRB MUST reply with a <mediaResourceResponse> with status code 420. If a syntactically correct request fails because it contains a wrong sequence number, that is, a 'seq' value not consistent with the increment the MRB expects according to Section 5.2.3, the MRB MUST reply with a <mediaResourceResponse> with status code 405. If a syntactically correct request fails because the MRB couldn't find any Media Server able to fulfill the requirements presented by the Application Server in its request, the MRB MUST reply with a <mediaResourceResponse> with status code 408. If a syntactically correct request fails because the MRB couldn't update an existing request according to the new requirements presented by the Application Server in its request, the MRB MUST reply with a <mediaResourceResponse> with status code 409. If a syntactically correct request fails because the MRB couldn't remove an existing request and release the related resources as requested by the Application Server, the MRB MUST reply with a <mediaResourceResponse> with status code 410.

Further details on status codes 409 and 410 are included in Section 5.2.3, where the leasing mechanism, along with its related scenarios, is described in more detail.

The <mediaResourceResponse> element has <response-session-info> as a child element. This element is used to describe the response of a Consumer interface query and is covered in the following sub-section.

5.2.6.1.1. <response-session-info>

The <response-session-info> element is included in Consumer responses. This applies to responses to both requests for new resources and requests to update an existing media resource session. The ability to change and remove an existing media resource session is described in more detail in Section 5.2.3. If the request was successful, the <mediaResourceResponse> MUST have one <response-session-info> child, which describes the media resource session addressed by the request. If the request was not successful, the <mediaResourceResponse> MUST NOT have a <response-session-info> child.

The <response-session-info> element has no attributes.

The <response-session-info> element has zero or more of the following child elements:

<session-id>: A unique identifier that explicitly references an existing media resource session on the MRB. The identifier is included to update the existing session and is described in more detail in Section 5.2.3.

<seq>: Used in association with the <session-id> element in a subsequent request to update an existing media resource session on an MRB. The <seq> number is incremented from its original value returned in response to the initial request for media resources. More information on its use is provided in Section 5.2.3.

<expires>: Includes the number of seconds that the media resources are reserved as part of this interaction. If the lease is not refreshed before expiry, the MRB will reclaim the resources and they will no longer be guaranteed. It is RECOMMENDED that a minimum value of 300 seconds be used for the value of the 'expires' attribute. It is also RECOMMENDED that a Consumer client refresh the lease at an interval that is not too close to the expiry time. A value of 80% of the timeout period could be used. For example, if the timeout period is 300 seconds, the Consumer client would refresh the transaction at 240 seconds. More information on its use is provided in Section 5.2.3.

<media-server-address>: Provides information to reach the Media Server handling the requested media resource. One or more instances of these elements may appear. The <media-server-address> element has a single attribute named 'uri', which supplies a SIP URI that reaches the specified Media Server. It also has three optional elements: <connection-id>, <ivr-sessions>, and <mixers>. The <ivr-sessions> and <mixers> elements are defined in Sections 5.2.5.1.2.1 and 5.2.5.1.3.1, respectively, and have the same meaning but are applied to individual Media Server instances as a subset of the overall resources reported in the <connection-id> element. If multiple Media Servers are assigned in an IAMM operation, exactly one <media-server-address> element, more specifically the Media Server that provided the media dialog or CFW response, will have a <connection-id> element. Additional information relating to the use of the <connection-id> element for media dialogs is included in Section 6.

5.3. In-Line Unaware MRB Interface

An entity acting as an In-line MRB can act in one of two roles for a request, as introduced in Section 4.2: the In-line Unaware MRB Mode (IUMM) of operation and the In-line Aware MRB Mode (IAMM) of operation. This section further describes IUMM.

It should be noted that the introduction of an MRB entity into the network, as specified in this document, requires interfaces to be implemented by those requesting Media Server resources (for example, an Application Server). This applies when using the Consumer interface as discussed in Sections 5.2.1 (Query mode) and 5.2.2 (IAMM). An MRB entity can also act in a client-unaware mode when deployed into the network. This allows any SIP-compliant client entity, as defined by RFC 3261 [RFC3261] and its extensions, to send requests to an MRB that in turn will select an appropriate Media Server based on knowledge of Media Server resources it currently has available transparently to the client entity. Using an MRB in this mode allows for easy migration of current applications and services that are unaware of the MRB concept and would simply require a configuration change resulting in the MRB being set as a SIP outbound proxy for clients requiring media services.

With IUMM, the MRB may conclude that an assigned media resource is no longer needed when it receives a SIP BYE from the Application Server or Media Server that ends the SIP dialog that initiated the request.

As with IAMM, in IUMM the SIP INVITE from the Application Server could convey the application/sdp payload to either set up a media dialog or a Control Framework Control Channel. In either case, in order to permit the Application Server to associate a media dialog with a Control Channel to the same Media Server, using the procedures of [RFC6230] Section 6, the MRB should be acting as a SIP proxy (and not a B2BUA). This allows the SIP URI of the targeted Media Server to be transparently passed back to the Application Server in the SIP response, resulting in a direct SIP dialog between the Application Server and the Media Server.

While IUMM has the least impact on legacy Application Servers, it also provides the least versatility. See Section 8.

6. MRB Acting as a B2BUA

An MRB entity can act as a SIP Back-to-Back User Agent (B2BUA) or a SIP Proxy Server as defined in RFC 3261 [RFC3261]. When an MRB acts as a B2BUA, issues can arise when using Media Control Channel packages such as the IVR [RFC6231] and mixer [RFC6505] packages. Specifically, the framework attribute 'connectionid' as provided in

Appendix A ("Common Package Components") of [RFC6230] uses a concatenation of the SIP dialog identifiers to be used for referencing SIP dialogs within the Media Control Channel. When a request traverses an MRB acting as a B2BUA, the SIP dialog identifiers change, and so the 'connectionid' cannot be used as intended due to this change. For this reason, when an MRB wishes to act as a SIP B2BUA when handling a request from an Application Server to set up a media dialog to a Media Server, it MUST include the optional <connection-id> element in a Consumer interface response with a value that provides the equivalent for the 'connectionid' ('Local Dialog Tag' + 'Remote Dialog Tag') for the far side of the B2BUA. If present, this value MUST be used as the value for the 'connectionid' in packages where the Common Package Components are used. The <connection-id> element MUST NOT be included in an HTTP Consumer interface response.

It is important to point out that although more Media Server instances may be returned in a Consumer response (i.e., the MRB has assigned more than one Media Server to a Consumer request to fulfill the Application Server requirements), in IAMM the MRB will only act as a B2BUA with a single Media Server. In this case, exactly one <media-server-address> element, describing the media dialog or CFW response, will have a <connection-id> element that will not be included in any additional <media-server-address> elements.

7. Multimodal MRB Implementations

An MRB implementation may operate multimodally with a collection of Application Server clients all sharing the same pool of media resources. That is, an MRB may be simultaneously operating in Query mode, IAMM, and IUMM. It knows in which mode to act on any particular request from a client, depending on the context of the request:

- If the received request is an HTTP POST message with application/mrb-consumer+xml content, then the MRB processes it in Query mode.

- If the received request is a SIP INVITE with application/mrb-consumer+xml content and application/sdp content, then the MRB processes it in IAMM.

- If the received request is a SIP INVITE without application/mrb-consumer+xml content but with application/sdp content, then the MRB processes it in IUMM.

8. Relative Merits of Query Mode, IAMM, and IUMM

At a high level, the possible Application Server MRB interactions can be distinguished by the following basic types:

a. Query mode - the client is requesting the assignment by the MRB of suitable Media Server resources;

b. IAMM/media dialog - the client is requesting the assignment by the MRB of suitable Media Server resources and the establishment of a media dialog to one of the Media Servers;

c. IAMM/Control Channel - the client is requesting the assignment by the MRB of suitable Media Server resources and the establishment of a CFW Control Channel to one of the Media Servers;

d. IUMM/media dialog - the client is requesting the establishment of a media dialog to a Media Server resource;

e. IUMM/Control Channel - the client is requesting the establishment of a CFW Control Channel to a Media Server resource.

Each type of interaction has advantages and disadvantages, where such considerations relate to the versatility of what the MRB can provide, technical aspects such as efficiency in different application scenarios, complexity, delay, use with legacy Application Servers, or use with the Media Control Channel Framework. Depending on the characteristics of a particular setting that an MRB is intended to support, some of the above interaction types may be more appropriate than others. This section provides a few observations on relative merits but is not intended to be exhaustive. Some constraints of a given interaction type may be subtle.

- Operation with other types of media control: Any of the types of interactions work with the mechanisms described in RFC 4240 [RFC4240] and RFC 5552 [RFC5552] where initial control instructions are conveyed in the SIP INVITE from the Application Server for the media dialog to the Media Server and subsequent instructions may be fetched using HTTP. Query mode (a), IAMM/media dialog (b), and IUMM/media dialog (d) work with the Media Server Markup Language (MSML) as per RFC 5707 [RFC5707] or the Media Server Control Markup Language (MSCML) as per RFC 5022 [RFC5022].

- As stated previously, IUMM has no interface impacts on an Application Server. When using IUMM, the Application Server does not specify the characteristics of the type of media resource it requires, as the <mediaResourceRequest> element is not passed to

the MRB. For IUMM/media dialog (d), the MRB can deduce an appropriate media resource on a best-effort basis using information gleaned from examining information in the SIP INVITE. This includes the SDP information for the media dialog, or initial control information in the SIP Request-URI as per RFC 4240 [RFC4240]. With IUMM/Control Channel (e), there is even less information for the MRB to use.

- If using IUMM/Control Channel (e), the subsequent sending of the media dialog to the Media Server should not be done using IUMM/media dialog. That is, the SIP signaling to send the media dialog to the selected Media Server must be directly between the Application Server and that Media Server, and not through the MRB. Unless resources can be confidentially identified, the MRB could send the media dialog to a different Media Server. Likewise, if using IUMM/media dialog (d), the subsequent establishment of a Control Channel should not be done with IUMM/Control Channel (e) unless definitive information is available.

- Query mode (a) and IAMM/Control Channel (c) lend themselves to requesting a pool of media resources (e.g., a number of IVR or conferencing ports) in advance of use and retaining use over a period of time, independent of whether there are media dialogs to those resources at any given moment, whereas the other types of interactions do not. This also applies to making a subsequent request to increase or decrease the amount of resources previously awarded.

- While Query mode (a) and IAMM/Control Channel (c) are the most versatile interaction types, the former is completely decoupled from the use or non-use of a Control Channel, whereas the latter requires the use of a Control Channel.

- When Media Control Channel Framework Control Channels are to be used in conjunction with the use of an MRB, Query mode (a) would typically result in fewer such channels being established over time, as compared to IAMM/Control Channel (c). That is because the latter would involve setting up an additional Control Channel every time an Application Server has a new request for an MRB for media resources.

9. Examples

This section provides examples of both the Publish and Consumer interfaces. Both the Query mode and In-line mode are addressed.

Note that due to RFC formatting conventions, this section often splits HTTP, SIP/SDP, and CFW across lines whose content would exceed 72 characters. A backslash character marks where this line folding has taken place. This backslash, and its trailing CRLF and whitespace, would not appear in the actual protocol contents. 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.

9.1. Publish Example

The following example assumes that a Control Channel has been established and synced as described in the Media Control Channel Framework ([RFC6230]).

Figure 9 shows the subscription/notification mechanism the Publish interface is based on, as defined in Section 5.1. The MRB subscribes for information at the Media Server (message A1.), and the Media Server accepts the subscription (A2.). Notifications are triggered by the Media Server (B1.) and acknowledged by the MRB (B2.).

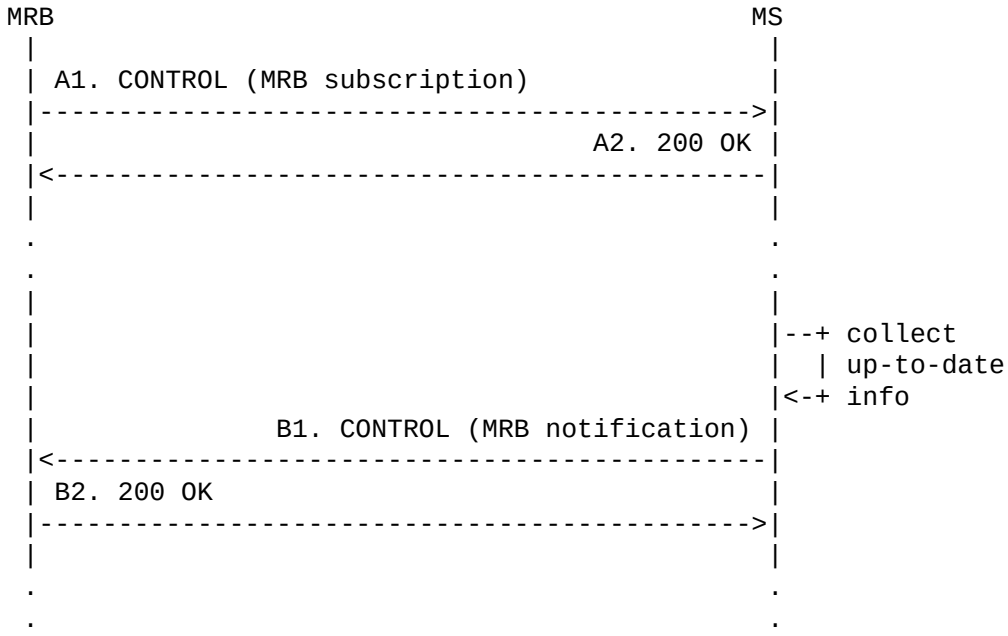


Figure 9: Publish Example: Sequence Diagram

The rest of this section includes a full dump of the messages associated with the previous sequence diagram, specifically:

1. the subscription (A1.), in an <mrbrequest> (CFW CONTROL);

2. the Media Server accepting the subscription (A2.), in an <mrbresponse> (CFW 200);

3. a notification (B1.), in an <mrbnotification> (CFW CONTROL);

4. the ack to the notification (B2.), in a framework-level 200 message (CFW 200).

```
A1. MRB -> MS (CONTROL, publish request)
----------------------------------------
CFW lidc30BZObiC CONTROL
Control-Package: mrb-publish/1.0
Content-Type: application/mrb-publish+xml
Content-Length: 337

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbpublish version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-publish">
    <mrbrequest>
        <subscription action="create" seqnumber="1" id="p0T65U">
            <expires>600</expires>
            <minfrequency>20</minfrequency>
            <maxfrequency>20</maxfrequency>
        </subscription>
    </mrbrequest>
</mrbpublish>


A2. MRB <- MS (200 to CONTROL, request accepted)
------------------------------------------------
CFW lidc30BZObiC 200
Timeout: 10
Content-Type: application/mrb-publish+xml
Content-Length: 139

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


B1. MRB <- MS (CONTROL, event notification from MS)
---------------------------------------------------
CFW 03fff52e7b7a CONTROL
Control-Package: mrb-publish/1.0
Content-Type: application/mrb-publish+xml
Content-Length: 4226

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <mrbpublish version="1.0"
              xmlns="urn:ietf:params:xml:ns:mrb-publish">
    <mrbnotification seqnumber="1" id="QQ6J3c">
        <media-server-id>a1b2c3d4</media-server-id>
        <supported-packages>
            <package name="msc-ivr/1.0"/>
            <package name="msc-mixer/1.0"/>
```

```
            <package name="mrb-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="7cfgs43">
                <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:MS1@ms.example.net</media-server-address>
     <encryption/>
    </mrbnotification>
   </mrbpublish>


B2. MRB -> MS (200 to CONTROL)
------------------------------
CFW 03fff52e7b7a 200
```

9.2. Consumer Examples

As specified in Section 5.2, the Consumer interface can be involved in two different modes: Query and In-line aware. When in Query mode, Consumer messages are transported in HTTP messages: an example of such an approach is presented in Section 9.2.1. When in In-line aware mode, messages are instead transported as part of SIP negotiations: considering that SIP negotiations may be related to either the creation of a Control Channel or to a User Agent Client (UAC) media dialog, two separate examples of such an approach are presented in Section 9.2.2.

9.2.1. Query Example

The following example assumes that the interested Application Server already knows the HTTP URL where an MRB is listening for Consumer messages.

Figure 10 shows the HTTP-based transaction between the Application Server (AS, as shown in the figure) and the MRB. The Application Server sends a Consumer request as payload of an HTTP POST message (1.), and the MRB provides an answer in an HTTP 200 OK message (2.). Specifically, as will be shown in the examples, the Application Server is interested in 100 IVR ports: the MRB finds two Media Servers that can satisfy the request (one providing 60 ports and the other providing 40 ports) and reports them to the Application Server.

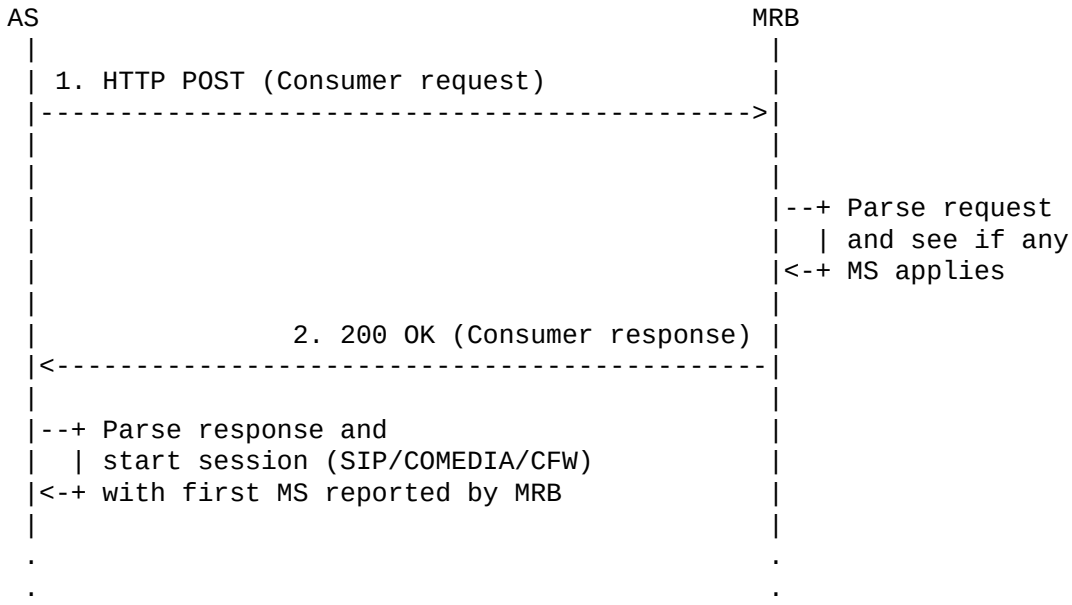


Figure 10: Consumer Example (Query): Sequence Diagram

The rest of this section includes a full dump of the messages associated with the previous sequence diagram, specifically:

1. the Consumer request (1.), in a <mediaResourceRequest> (HTTP POST, Content-Type 'application/mrb-consumer+xml');

2. the Consumer response (2.), in a <mediaResourceResponse> (HTTP 200 OK, Content-Type 'application/mrb-consumer+xml').

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.net: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="gh11x23v">
        <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>
```

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: 1133
Date: Mon, 12 Apr 2011 14:59:26 GMT

<?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="gh11x23v">
        <response-session-info>
            <session-id>5t3Y4IQ84gY1</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>
</mrbconsumer>
```

   As the example shows, the request and response are associated by means of the 'id' attribute (id="gh11x23v"). The MRB has picked '9' as the random sequence number that needs to be incremented by the Application Server for the subsequent request associated with the same session.

The rest of the scenario is omitted for brevity. After having received the 'mediaResourceResponse', the Application Server has the URIs of two Media Servers able to fulfill its media requirements and can start a control dialog with one or both of them.

9.2.2. IAMM Examples

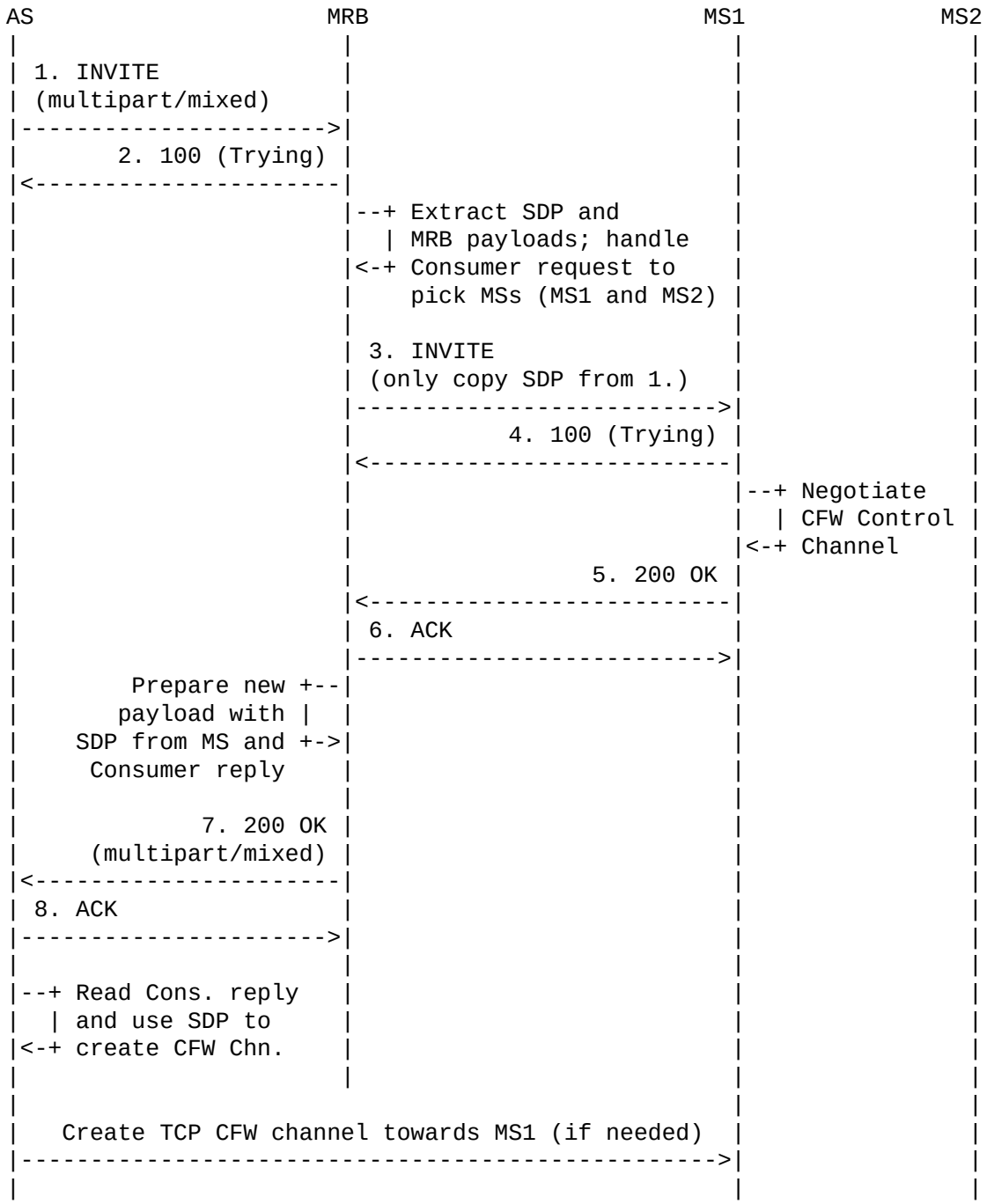
Two separate examples are presented for the IAMM case: in fact, IAMM can take advantage of two different approaches with respect to the SIP dialogs to be exploited to carry Consumer messages, i.e., i) a SIP control dialog to create a Control Channel, and ii) a UAC media dialog to attach to a Media Server. 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 behavior of the involved parties may differ.

9.2.2.1. IAMM Example: CFW-Based Approach

The following example assumes that the interested Application Server already knows the SIP URI of an MRB.

Figure 11 shows the first approach, i.e., SIP-based transactions between the Application Server, the MRB, and one Media Server that the MRB chooses from the two that are allocated to fulfill the request. The diagram is more complex than before. This is basically a scenario envisaging the MRB as a B2BUA. The Application Server sends a SIP INVITE (1.) containing both a CFW-related SDP and a Consumer request (multipart body). The MRB sends a provisional response to the Application Server (2.) and starts working on the request. First of all, it makes use of the Consumer request from the Application Server to determine which Media Servers should be exploited. Once the right Media Servers have been chosen (MS1 and MS2 in the example), the MRB sends a new SIP INVITE (3.) to one of the Media Servers (MS1 in the example) by just including the SDP part of the original request. That Media Server negotiates this INVITE as specified in [RFC6230] (4., 5., 6.), providing the MRB with its own CFW-related SDP. The MRB replies to the original Application Server INVITE preparing a SIP 200 OK with another multipart body (7.): this multipart body includes the Consumer response used by the MRB to determine the right Media Servers and the SDP returned by the Media Server (MS1) in (5.). The Application Server finally acknowledges the 200 OK (8.), and can start a CFW connection towards that Media Server (MS1). Since the MRB provided the Application Server with two Media Server instances to fulfill its requirements, the Application Server can use the URI in the <media-server-address> element in the <mediaResourceResponse> that describes the other Media Server to establish a CFW channel with that Media Server (MS2) as well.

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.



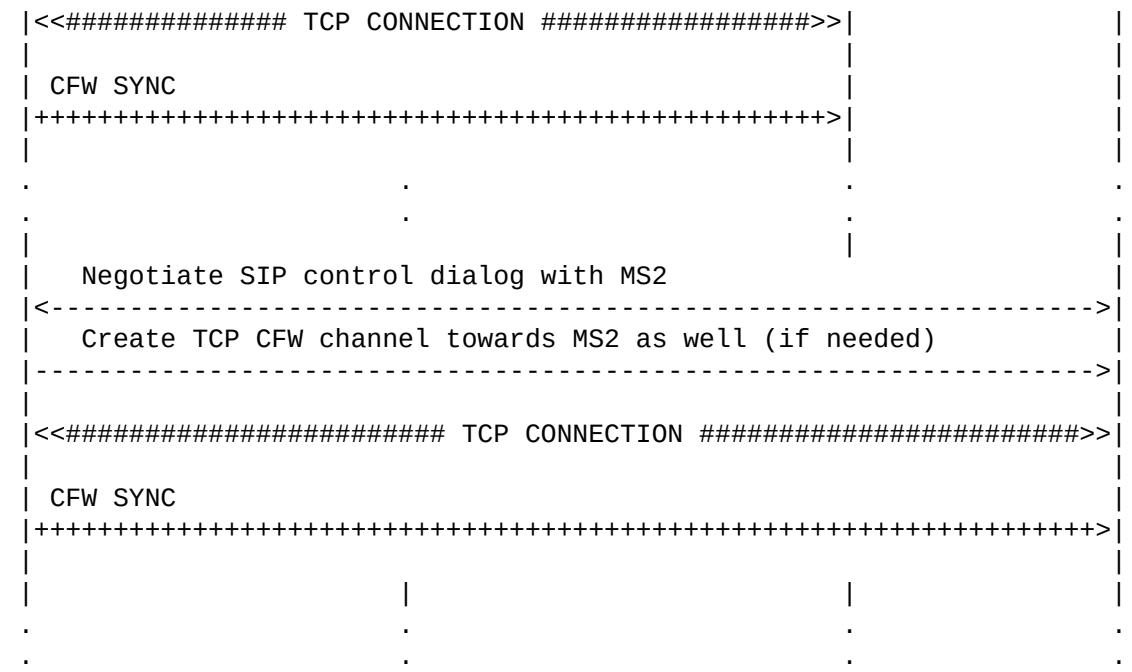


Figure 11: Consumer Example (IAMM/Control Channel): Sequence Diagram

The rest of this section includes an almost full trace of the messages associated with the previous sequence diagram. Only the relevant SIP messages are shown (both the INVITEs and the 200 OKs), and only the relevant headers are preserved for brevity (Content-Type and multipart-related information). Specifically:

1. the original INVITE (1.) containing both a CFW-related SDP (Connection-Oriented Media (COMEDIA) information to negotiate a new Control Channel) and a Consumer <mediaResourceRequest>;

2. the INVITE sent by the MRB (acting as a B2BUA) to the Media Server (3.), containing only the CFW-related SDP from the original INVITE;

3. the 200 OK sent by the Media Server back to the MRB (5.) to complete the CFW-related negotiation (SDP only);

4. the 200 OK sent by the MRB back to the Application Server in response to the original INVITE (7.), containing both the CFW-related information sent by the Media Server and a Consumer <mediaResourceRequest> documenting the MRB's decision to use that Media Server.

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
   a=ctrl-package:msc-mixer/1.0
   a=ctrl-package:msc-ivr/1.0

   =_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="pz78hnq1">
        <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
   a=ctrl-package:msc-mixer/1.0
   a=ctrl-package:msc-ivr/1.0
```

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
   a=ctrl-package:msc-mixer/1.0
   a=ctrl-package:msc-ivr/1.0
   a=ctrl-package:mrb-publish/1.0
   a=ctrl-package:msc-example-pkg/1.0
```

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
   a=ctrl-package:msc-mixer/1.0
   a=ctrl-package:msc-ivr/1.0
   a=ctrl-package:mrb-publish/1.0
   a=ctrl-package:msc-example-pkg/1.0

   =_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="pz78hnq1">
        <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
```

As the previous example illustrates, the only difference in the response that the MRB provides to the Application Server is in the 'connection-id' attribute that is added to the first allocated Media Server instance: this allows the Application Server to understand that the MRB has sent the CFW channel negotiation to that specific Media Server and that the connection-id to be used is the one provided. This will be described in more detail in the following section for the media dialog-based approach.

The continuation of the scenario (the Application Server connecting to MS1 to start the Control Channel and the related SYNC message, the Application Server connecting to MS2 as well later on, all the media dialogs being attached to either Media Server) is omitted for brevity.

9.2.2.2. IAMM Example: Media Dialog-Based Approach

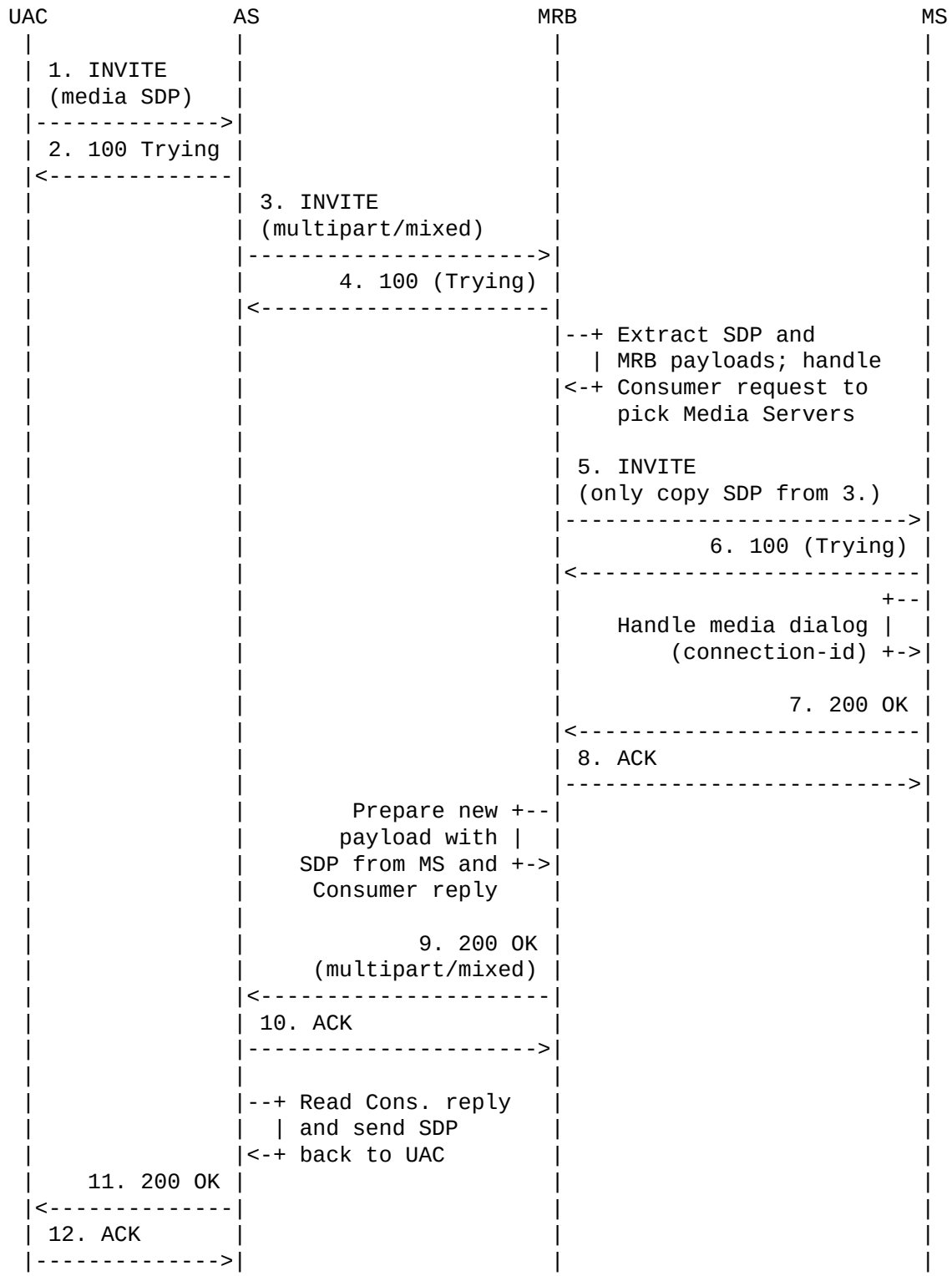
The following example assumes that the interested Application Server already knows the SIP URI of an MRB.

Figure 12 shows the second approach, i.e., SIP-based transactions between a SIP client, the Application Server, the MRB, and the Media Server that the MRB chooses. The interaction is basically the same as previous examples (e.g., contents of the multipart body), but considering that a new party is involved in the communication, the diagram is slightly more complex than before. As before, the MRB acts as a B2BUA. A UAC sends a SIP INVITE to a SIP URI handled by the Application Server, since it is interested to its services (1.). The Application Server sends a provisional response (2.) and, since it doesn't have the resources yet, sends to the MRB a new SIP INVITE (3.) containing both the UAC media-related SDP and a Consumer request (multipart body). The MRB sends a provisional response to the Application Server (4.) and starts working on the request. First of all, it makes use of the Consumer request from the Application Server to determine which Media Servers should be chosen. Once the Media Server has been chosen, the MRB sends a new SIP INVITE to one of the Media Servers by including the SDP part of the original request (5.).

The Media Server negotiates this INVITE as specified in [RFC6230] (6., 7., 8.) to allocate the needed media resources to handle the new media dialog, eventually providing the MRB with its own media-related SDP. The MRB replies to the original Application Server INVITE preparing a SIP 200 OK with a multipart body (9.): this multipart body includes the Consumer response from the MRB indicating the chosen Media Servers and the SDP returned by the Media Server in (7.). The Application Server finally acknowledges the 200 OK (10.) and ends the scenario by eventually providing the UAC with the SDP it needs to set up the RTP channels with the chosen Media Server: a separate direct SIP control dialog may be initiated by the Application Server to the same Media Server in order to set up a Control Channel to manipulate the media dialog.

As with the IAMM/Control Channel example in the prior section, this example has the MRB selecting Media Server resources across two Media Server instances. The convention could be that the MRB sent the SIP INVITE to the first Media Server in the list provided to the Application Server in the Consumer response information. For the sake of brevity, considerations related to connecting to the other Media Servers as well are omitted, since they have already been addressed in the previous section.

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.


UAC
AS
MRB
MS
1. INVITE
(media SDP)
2. 100 Trying
3. INVITE
(multipart/mixed)
4. 100 (Trying)
Extract SDP and
MRB payloads; handle
Consumer request to
pick Media Servers
5. INVITE
(only copy SDP from 3.)
6. 100 (Trying)
Handle media dialog
(connection-id)
7. 200 OK
8. ACK
Prepare new
payload with
SDP from MS and
Consumer reply
9. 200 OK
(multipart/mixed)
10. ACK
Read Cons. reply
and send SDP
back to UAC
11. 200 OK
12. ACK

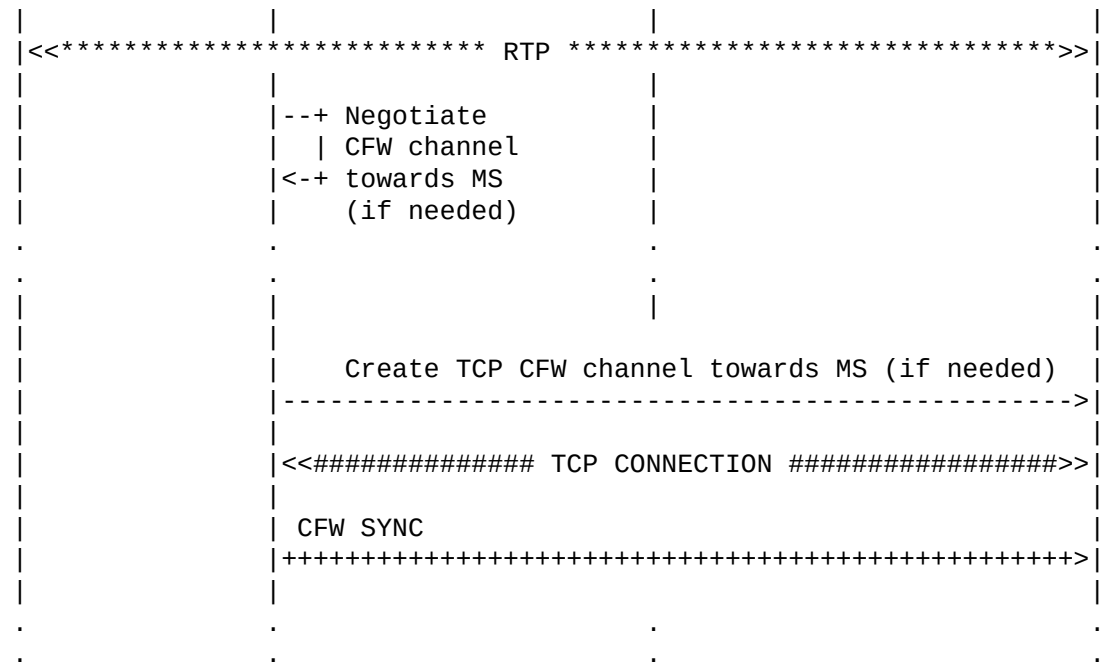


Figure 12: Consumer Example (IAMM/Media Dialog): Sequence Diagram

The rest of this section includes a trace of the messages associated with the previous sequence diagram. Only the relevant SIP messages are shown (both the INVITEs and the 200 OKs), and only the relevant headers are preserved for brevity (Content-Type, From/To, and multipart-related information). Specifically:

1. the original INVITE (1.) containing the media-related SDP sent by a UAC;

2. the INVITE sent by the AS to the MRB (3.), containing both the media-related SDP and a Consumer <mediaResourceRequest>;

3. the INVITE sent by the MRB (acting as a B2BUA) to the Media Server (5.), containing only the media-related SDP from the original INVITE;

4. the 200 OK sent by the Media Server back to the MRB (7.) to complete the media-related negotiation (SDP only);

5. the 200 OK sent by the MRB back to the Application Server in response to the original INVITE (9.), containing both the media-related information sent by the Media Server and a Consumer <mediaResourceRequest> documenting the MRB's decision to use that Media Server;

6. the 200 OK sent by the Application Server back to the UAC to have it set up the RTP channel(s) with the Media Server (11.).

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@mrb.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="ns56g1x0">
     <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
```

5. MRB -> MS (INVITE sdp only)

```
   [..]
   From: <sip:Mrb@mrb.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@mrb.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@mrb.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="ns56g1x0">
        <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
```

As the examples illustrate, as in the IAMM/Control Channel example, the MRB provides the Application Server with a <media-server-address> element in the Consumer response: the 'uri' attribute identifies the

specific Media Server to which the MRB has sent the SDP media negotiation, and the 'connection-id' enables the Application Server to identify to the Media Server the dialog between the MRB and Media Server. This attribute is needed, since according to the framework specification [RFC6230] the connection-id is built out of the From/To tags of the dialog between the MRB and Media Server; since the MRB acts as a B2BUA in this scenario, without that attribute the Application Server does not know the relevant tags, thus preventing the CFW protocol from working as expected.

The continuation of the scenario (the Application Server connecting to the Media Server to start the Control Channel, the SYNC message, etc.) is omitted for brevity.

10. Media Service Resource Publisher Interface XML Schema

This section gives the XML Schema Definition [W3C.REC-xmlschema-1-20041028] [W3C.REC-xmlschema-2-20041028] of the "application/mrb-publish+xml" format.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:ietf:params:xml:ns:mrb-publish"
 elementFormDefault="qualified" blockDefault="#all"
 xmlns="urn:ietf:params:xml:ns:mrb-publish"
 xmlns:fw="urn:ietf:params:xml:ns:control:framework-attributes"
 xmlns:ca="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation>
   IETF MediaCtrl MRB 1.0

   This is the schema of the IETF MediaCtrl MRB package.

   The schema namespace is urn:ietf:params:xml:ns:mrb-publish

  </xsd:documentation>
 </xsd:annotation>


 <!--
  ################################################################

  SCHEMA IMPORTS

  ################################################################
 -->
```

```
 <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
  schemaLocation="http://www.w3.org/2001/xml.xsd">
  <xsd:annotation>
   <xsd:documentation>
    This import brings in the XML attributes for
    xml:base, xml:lang, etc.
   </xsd:documentation>
  </xsd:annotation>
 </xsd:import>

 <xsd:import
  namespace="urn:ietf:params:xml:ns:control:framework-attributes"
  schemaLocation="framework.xsd">
  <xsd:annotation>
   <xsd:documentation>
    This import brings in the framework attributes for
    conferenceid and connectionid.
   </xsd:documentation>
  </xsd:annotation>
 </xsd:import>

 <xsd:import
  namespace="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
  schemaLocation="civicAddress.xsd">
  <xsd:annotation>
   <xsd:documentation>
    This import brings in the civicAddress specification
    from RFC 5139.
   </xsd:documentation>
  </xsd:annotation>
 </xsd:import>

<!--
  ######################################################

  Extensible core type

  ######################################################
 -->

 <xsd:complexType name="Tcore">
  <xsd:annotation>
   <xsd:documentation>
    This type is extended by other (non-mixed) component types to
    allow attributes from other namespaces.
   </xsd:documentation>
  </xsd:annotation>
```

```
  <xsd:sequence/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
 </xsd:complexType>


<!--
  ########################################################

  TOP-LEVEL ELEMENT: mrbpublish

  ########################################################
 -->

<xsd:complexType name="mrbpublishType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
     <xsd:choice>
      <xsd:element ref="mrbrequest" />
      <xsd:element ref="mrbresponse" />
      <xsd:element ref="mrbnotification" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
     </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="version" type="version.datatype"
      use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="mrbpublish" type="mrbpublishType" />

<!--
  ########################################################

  mrbrequest TYPE

  ########################################################
 -->

<!--  mrbrequest -->

 <xsd:complexType name="mrbrequestType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
```

```
     <xsd:element ref="subscription" />
     <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="mrbrequest" type="mrbrequestType" />

<!--  subscription -->

<xsd:complexType name="subscriptionType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
     <xsd:element name="expires" type="xsd:nonNegativeInteger"
      minOccurs="0" maxOccurs="1" />
     <xsd:element name="minfrequency" type="xsd:nonNegativeInteger"
      minOccurs="0" maxOccurs="1" />
     <xsd:element name="maxfrequency" type="xsd:nonNegativeInteger"
      minOccurs="0" maxOccurs="1" />
     <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="id" type="id.datatype" use="required" />
    <xsd:attribute name="seqnumber" type="xsd:nonNegativeInteger"
      use="required" />
    <xsd:attribute name="action" type="action.datatype"
      use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="subscription" type="subscriptionType" />
```

```
<!--
  ######################################################

  mrbresponse TYPE

  ######################################################
 -->

<!--  mrbresponse -->

 <xsd:complexType name="mrbresponseType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
     <xsd:element ref="subscription" minOccurs="0" maxOccurs="1" />
     <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="status" type="status.datatype"
     use="required" />
    <xsd:attribute name="reason" type="xsd:string" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>


 <xsd:element name="mrbresponse" type="mrbresponseType" />

<!--
  ######################################################

  mrbnotification TYPE

  ######################################################
 -->

<!--  mrbnotification -->

<xsd:complexType name="mrbnotificationType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
     <xsd:element name="media-server-id"
        type="subscriptionid.datatype"/>
     <xsd:element ref="supported-packages" minOccurs="0" />
     <xsd:element ref="active-rtp-sessions" minOccurs="0" />
     <xsd:element ref="active-mixer-sessions" minOccurs="0" />
```

```
     <xsd:element ref="non-active-rtp-sessions" minOccurs="0" />
     <xsd:element ref="non-active-mixer-sessions" minOccurs="0" />
     <xsd:element ref="media-server-status" minOccurs="0" />
     <xsd:element ref="supported-codecs" minOccurs="0" />
     <xsd:element ref="application-data" minOccurs="0"
      maxOccurs="unbounded" />
     <xsd:element ref="file-formats" minOccurs="0" />
     <xsd:element ref="max-prepared-duration" minOccurs="0" />
     <xsd:element ref="dtmf-support" minOccurs="0" />
     <xsd:element ref="mixing-modes" minOccurs="0" />
     <xsd:element ref="supported-tones" minOccurs="0" />
     <xsd:element ref="file-transfer-modes" minOccurs="0" />
     <xsd:element ref="asr-tts-support" minOccurs="0" />
     <xsd:element ref="vxml-support" minOccurs="0" />
     <xsd:element ref="media-server-location" minOccurs="0" />
     <xsd:element ref="label" minOccurs="0" />
     <xsd:element ref="media-server-address" minOccurs="0" />
     <xsd:element ref="encryption" minOccurs="0" />
     <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
     <xsd:attribute name="id" type="subscriptionid.datatype"
      use="required" />
     <xsd:attribute name="seqnumber" type="xsd:nonNegativeInteger"
      use="required" />
     <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="mrbnotification" type="mrbnotificationType" />


<!--  supported-packages -->

 <xsd:complexType name="supported-packagesType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="package" minOccurs="0"
        maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
```

```
<xsd:element name="supported-packages" type="supported-packagesType"/>


 <xsd:complexType name="packageType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="package" type="packageType" />


<!--  active-rtp-sessions -->

 <xsd:complexType name="active-rtp-sessionsType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="rtp-codec" minOccurs="0"
        maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="active-rtp-sessions" type="active-rtp-sessionsType"/>


 <xsd:complexType name="rtp-codecType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element name="decoding" type="xsd:nonNegativeInteger" />
      <xsd:element name="encoding" type="xsd:nonNegativeInteger" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
```

```
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="rtp-codec" type="rtp-codecType" />


<!--  active-mixer-sessions -->

<xsd:complexType name="active-mixer-sessionsType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="active-mix" minOccurs="0"
        maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="active-mixer-sessions"
  type="active-mixer-sessionsType" />


<xsd:complexType name="active-mixType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="rtp-codec" minOccurs="0"
        maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attributeGroup ref="fw:framework-attributes" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="active-mix" type="active-mixType" />
```

```
<!--  non-active-rtp-sessions -->

<xsd:complexType name="non-active-rtp-sessionsType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="rtp-codec" minOccurs="0"
        maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="non-active-rtp-sessions"
  type="non-active-rtp-sessionsType" />

<!--  non-active-mixer-sessions -->

<xsd:complexType name="non-active-mixer-sessionsType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="non-active-mix" minOccurs="0"
        maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="non-active-mixer-sessions"
  type="non-active-mixer-sessionsType" />

 <xsd:complexType name="non-active-mixType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="rtp-codec" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="available" type="xsd:nonNegativeInteger"
      use="required" />
```

```
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="non-active-mix" type="non-active-mixType" />

<!--  media-server-status -->

 <xsd:element name="media-server-status" type="msstatus.datatype" />

<!--  supported-codecs -->

<xsd:complexType name="supported-codecsType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="supported-codec"
         minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
         maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="supported-codecs" type="supported-codecsType" />

 <xsd:complexType name="supported-codecType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="supported-codec-package"
         minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="supported-codec" type="supported-codecType" />
```

```
 <xsd:complexType name="supported-codec-packageType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element name="supported-action" type="actions.datatype"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="supported-codec-package"
  type="supported-codec-packageType" />


<!--  application-data -->

<xsd:element name="application-data" type="appdata.datatype" />

<!--  file-formats -->

<xsd:complexType name="file-formatsType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="supported-format"
         minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
         maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="file-formats" type="file-formatsType" />

 <xsd:complexType name="supported-formatType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="supported-file-package"
         minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
```

```
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="supported-format" type="supported-formatType" />

 <xsd:element name="supported-file-package"
  type="xsd:string" />

<!--  max-prepared-duration -->

<xsd:complexType name="max-prepared-durationType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="max-time" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="max-prepared-duration"
  type="max-prepared-durationType" />


 <xsd:complexType name="max-timeType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element name="max-time-package" type="xsd:string" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="max-time-seconds" type="xsd:nonNegativeInteger"
     use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="max-time" type="max-timeType" />
```

```
<!--  dtmf-support -->

<xsd:complexType name="dtmf-supportType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="detect" />
       <xsd:element ref="generate" />
       <xsd:element ref="passthrough" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="dtmf-support" type="dtmf-supportType" />

 <xsd:complexType name="detectType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="dtmf-type"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="detect" type="detectType" />

 <xsd:complexType name="generateType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="dtmf-type"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
```

```
 <xsd:element name="generate" type="generateType" />

 <xsd:complexType name="passthroughType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="dtmf-type"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="passthrough" type="passthroughType" />


 <xsd:complexType name="dtmf-typeType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="dtmf.datatype" use="required" />
    <xsd:attribute name="package" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="dtmf-type" type="dtmf-typeType" />


<!--  mixing-modes -->

<xsd:complexType name="mixing-modesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
      <xsd:element ref="audio-mixing-modes"
        minOccurs="0" maxOccurs="1" />
      <xsd:element ref="video-mixing-modes"
        minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
```

```
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="mixing-modes" type="mixing-modesType" />

<xsd:complexType name="audio-mixing-modesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="audio-mixing-mode"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="audio-mixing-modes" type="audio-mixing-modesType" />

<xsd:complexType name="audio-mixing-modeType" mixed="true">
 <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="package" type="xsd:string" use="required" />
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="audio-mixing-mode" type="audio-mixing-modeType" />

<xsd:complexType name="video-mixing-modesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="video-mixing-mode"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:attribute name="vas" type="boolean.datatype"
     default="false" />
   <xsd:attribute name="activespeakermix" type="boolean.datatype"
     default="false" />
```

```
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="video-mixing-modes" type="video-mixing-modesType" />

<xsd:complexType name="video-mixing-modeType" mixed="true">
 <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="package" type="xsd:string" use="required" />
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="video-mixing-mode" type="video-mixing-modeType" />


<!--  supported-tones -->

<xsd:complexType name="supported-tonesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
      <xsd:element ref="supported-country-codes"
        minOccurs="0" maxOccurs="1" />
      <xsd:element ref="supported-h248-codes"
        minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-tones" type="supported-tonesType" />

<xsd:complexType name="supported-country-codesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="country-code"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
```

```
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-country-codes"
  type="supported-country-codesType" />

<xsd:complexType name="country-codeType" mixed="true">
 <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="package" type="xsd:string" use="required" />
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="country-code" type="country-codeType" />

<xsd:complexType name="supported-h248-codesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="h248-code"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-h248-codes"
  type="supported-h248-codesType" />

<xsd:complexType name="h248-codeType" mixed="true">
 <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="package" type="xsd:string" use="required" />
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="h248-code" type="h248-codeType" />
```

```
<!--  file-transfer-modes -->

 <xsd:complexType name="file-transfer-modesType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="file-transfer-mode"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="file-transfer-modes"
              type="file-transfer-modesType" />

 <xsd:complexType name="file-transfer-modeType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="transfermode.datatype"
     use="required" />
    <xsd:attribute name="package" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
 <xsd:element name="file-transfer-mode" type="file-transfer-modeType" />


<!--  asr-tts-support -->

<xsd:complexType name="asr-tts-supportType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
      <xsd:element ref="asr-support"
        minOccurs="0" maxOccurs="1" />
      <xsd:element ref="tts-support"
        minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
```

```
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="asr-tts-support" type="asr-tts-supportType" />

<xsd:complexType name="asr-supportType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="language"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="asr-support" type="asr-supportType" />

<xsd:complexType name="tts-supportType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="language"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="tts-support" type="tts-supportType" />

<xsd:complexType name="languageType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:attribute ref="xml:lang" />
```

```
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="language" type="languageType" />


<!--  media-server-location -->

<xsd:complexType name="media-server-locationType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element name="civicAddress" type="ca:civicAddress"
                       minOccurs="1" maxOccurs="1" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="media-server-location"
  type="media-server-locationType" />


<!--  vxml-support -->

 <xsd:complexType name="vxml-supportType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="vxml-mode"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="vxml-support" type="vxml-supportType" />
```

```
 <xsd:complexType name="vxml-modeType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="package" type="xsd:string" use="required" />
    <xsd:attribute name="support" type="vxml.datatype" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="vxml-mode" type="vxml-modeType" />


<!--  label -->

 <xsd:element name="label" type="label.datatype" />


<!-- media-server-address -->

 <xsd:element name="media-server-address" type="xsd:anyURI" />

<!--  encryption -->

 <xsd:complexType name="encryptionType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="encryption" type="encryptionType" />
```

```
<!--
  ######################################################

  DATATYPES

  ######################################################
 -->

 <xsd:simpleType name="version.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="1.0" />
  </xsd:restriction>
 </xsd:simpleType>

<xsd:simpleType name="id.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
 </xsd:simpleType>

 <xsd:simpleType name="status.datatype">
  <xsd:restriction base="xsd:positiveInteger">
   <xsd:pattern value="[0-9][0-9][0-9]" />
  </xsd:restriction>
 </xsd:simpleType>

 <xsd:simpleType name="msstatus.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="active" />
   <xsd:enumeration value="deactivated" />
   <xsd:enumeration value="unavailable" />
  </xsd:restriction>
 </xsd:simpleType>

 <xsd:simpleType name="action.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="create" />
   <xsd:enumeration value="update" />
   <xsd:enumeration value="remove" />
  </xsd:restriction>
 </xsd:simpleType>

 <xsd:simpleType name="actions.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="encoding" />
   <xsd:enumeration value="decoding" />
   <xsd:enumeration value="passthrough" />
  </xsd:restriction>
 </xsd:simpleType>
```

```
 <xsd:simpleType name="appdata.datatype">
  <xsd:restriction base="xsd:string" />
 </xsd:simpleType>

 <xsd:simpleType name="dtmf.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
 </xsd:simpleType>

 <xsd:simpleType name="transfermode.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
 </xsd:simpleType>

 <xsd:simpleType name="boolean.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="true" />
   <xsd:enumeration value="false" />
  </xsd:restriction>
 </xsd:simpleType>

 <xsd:simpleType name="vxml.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
 </xsd:simpleType>

 <xsd:simpleType name="label.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
 </xsd:simpleType>

 <xsd:simpleType name="subscriptionid.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
 </xsd:simpleType>

</xsd:schema>
```

11. Media Service Resource Consumer Interface XML Schema

This section gives the XML Schema Definition [W3C.REC-xmlschema-1-20041028] [W3C.REC-xmlschema-2-20041028] of the "application/mrb-consumer+xml" format.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:ietf:params:xml:ns:mrb-consumer"
 elementFormDefault="qualified" blockDefault="#all"
 xmlns="urn:ietf:params:xml:ns:mrb-consumer"
 xmlns:ca="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation>
   IETF MediaCtrl MRB 1.0

   This is the schema of the IETF MediaCtrl MRB Consumer interface.

   The schema namespace is urn:ietf:params:xml:ns:mrb-consumer

  </xsd:documentation>
 </xsd:annotation>


 <!--
  ################################################################

  SCHEMA IMPORTS

  ################################################################
 -->

 <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
  schemaLocation="http://www.w3.org/2001/xml.xsd">
  <xsd:annotation>
   <xsd:documentation>
    This import brings in the XML attributes for
    xml:base, xml:lang, etc.
   </xsd:documentation>
  </xsd:annotation>
 </xsd:import>

 <xsd:import
  namespace="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
  schemaLocation="civicAddress.xsd">
  <xsd:annotation>
   <xsd:documentation>
```

```
    This import brings in the civicAddress specification
    from RFC 5139.
   </xsd:documentation>
  </xsd:annotation>
 </xsd:import>

<!--
  ########################################################

  Extensible core type

  ########################################################
 -->

 <xsd:complexType name="Tcore">
  <xsd:annotation>
   <xsd:documentation>
    This type is extended by other (non-mixed) component types to
    allow attributes from other namespaces.
   </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
 </xsd:complexType>


<!--
  ########################################################

  TOP-LEVEL ELEMENT: mrbconsumer

  ########################################################
 -->

<xsd:complexType name="mrbconsumerType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
     <xsd:choice>
      <xsd:element ref="mediaResourceRequest" />
      <xsd:element ref="mediaResourceResponse" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
     </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="version" type="version.datatype"
      use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
```

```
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

 <xsd:element name="mrbconsumer" type="mrbconsumerType" />


<!--
  ######################################################

  mediaResourceRequest TYPE

  ######################################################
 -->

<!--  mediaResourceRequest -->

 <xsd:complexType name="mediaResourceRequestType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="generalInfo" minOccurs="0" />
      <xsd:element ref="ivrInfo" minOccurs="0" />
      <xsd:element ref="mixerInfo" minOccurs="0" />
     <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"
      use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="mediaResourceRequest"
        type="mediaResourceRequestType" />
```

```
<!--
  ######################################################

  generalInfo TYPE

  ######################################################
-->

<!--  generalInfo -->

<xsd:complexType name="generalInfoType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="session-info" minOccurs="0" />
      <xsd:element ref="packages" minOccurs="0" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="generalInfo" type="generalInfoType" />


<!--  session-info -->

<xsd:complexType name="session-infoType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element name="session-id" type="id.datatype"/>
       <xsd:element name="seq" type="xsd:nonNegativeInteger"/>
       <xsd:element name="action" type="action.datatype"/>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="session-info" type="session-infoType" />
```

```
<!--  packages -->

<xsd:complexType name="packagesType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element name="package" type="xsd:string" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="packages" type="packagesType"/>


<!--
  ######################################################

  ivrInfo TYPE

  ######################################################
-->

<!--  ivrInfo -->

<xsd:complexType name="ivrInfoType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="ivr-sessions" minOccurs="0" />
      <xsd:element ref="file-formats" minOccurs="0" />
      <xsd:element ref="dtmf-type" minOccurs="0" />
      <xsd:element ref="tones" minOccurs="0" />
      <xsd:element ref="asr-tts" minOccurs="0" />
      <xsd:element ref="vxml" minOccurs="0" />
      <xsd:element ref="location" minOccurs="0" />
      <xsd:element ref="encryption" minOccurs="0" />
      <xsd:element ref="application-data" minOccurs="0" />
      <xsd:element ref="max-prepared-duration" minOccurs="0" />
      <xsd:element ref="file-transfer-modes" minOccurs="0" />
      <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
```

```
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="ivrInfo" type="ivrInfoType" />


<!--
  ########################################################

  mixerInfo TYPE

  ########################################################
-->

<!--  mixerInfo -->

<xsd:complexType name="mixerInfoType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="mixers" minOccurs="0"/>
      <xsd:element ref="file-formats" minOccurs="0"/>
      <xsd:element ref="dtmf-type" minOccurs="0"/>
      <xsd:element ref="tones" minOccurs="0"/>
      <xsd:element ref="mixing-modes" minOccurs="0"/>
      <xsd:element ref="application-data" minOccurs="0"/>
      <xsd:element ref="location" minOccurs="0"/>
      <xsd:element ref="encryption" minOccurs="0"/>
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="mixerInfo" type="mixerInfoType" />
```

```
<!--
  ######################################################

  mediaResourceResponse TYPE

  ######################################################
 -->

<!--  mediaResourceResponse -->

 <xsd:complexType name="mediaResourceResponseType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="response-session-info" minOccurs="0" />
       <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"
     use="required" />
    <xsd:attribute name="status" type="status.datatype"
     use="required" />
    <xsd:attribute name="reason" type="xsd:string" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="mediaResourceResponse"
         type="mediaResourceResponseType" />


<!--
  ######################################################

  ELEMENTS

  ######################################################
 -->

<!--  response-session-info -->

<xsd:complexType name="response-session-infoType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element name="session-id" type="id.datatype"/>
       <xsd:element name="seq" type="xsd:nonNegativeInteger"/>
```

```
       <xsd:element name="expires" type="xsd:nonNegativeInteger"/>
       <xsd:element ref="media-server-address"
          minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
         maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="response-session-info"
   type="response-session-infoType" />

<!-- media-server-address -->

<xsd:complexType name="media-server-addressTYPE">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
     <xsd:element name="connection-id" type="xsd:string"
         minOccurs="0" maxOccurs="unbounded" />
     <xsd:element ref="ivr-sessions" minOccurs="0"/>
     <xsd:element ref="mixers" minOccurs="0"/>
     <xsd:any namespace="##other" minOccurs="0"
         maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="uri" type="xsd:anyURI" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="media-server-address"
    type="media-server-addressTYPE" />

<!--  ivr-sessions -->

<xsd:complexType name="ivr-sessionsType">
 <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="rtp-codec" minOccurs="0"
        maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
```

```
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="ivr-sessions" type="ivr-sessionsType" />

<xsd:complexType name="rtp-codecType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element name="decoding" type="xsd:nonNegativeInteger" />
      <xsd:element name="encoding" type="xsd:nonNegativeInteger" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="rtp-codec" type="rtp-codecType" />


<!-- file-formats -->

<xsd:complexType name="file-formatsType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="required-format"
         minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
         maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="file-formats" type="file-formatsType" />

<xsd:complexType name="required-formatType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="required-file-package"
         minOccurs="0" maxOccurs="unbounded" />
```

```
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="required-format" type="required-formatType" />

<xsd:complexType name="required-file-packageType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element name="required-file-package-name" type="xsd:string"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="required-file-package"
  type="required-file-packageType" />

<!--  dtmf-type -->

<xsd:complexType name="dtmfType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="detect" />
       <xsd:element ref="generate" />
       <xsd:element ref="passthrough" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="dtmf" type="dtmfType" />

<xsd:complexType name="detectType">
```

```
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="dtmf-type"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="detect" type="detectType" />

<xsd:complexType name="generateType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="dtmf-type"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="generate" type="generateType" />

<xsd:complexType name="passthroughType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="dtmf-type"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="passthrough" type="passthroughType" />

<xsd:complexType name="dtmf-typeType">
```

```
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="dtmf.datatype" use="required" />
    <xsd:attribute name="package" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="dtmf-type" type="dtmf-typeType" />

<!--  tones -->

<xsd:complexType name="required-tonesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
      <xsd:element ref="country-codes"
        minOccurs="0" maxOccurs="1" />
      <xsd:element ref="h248-codes"
        minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="tones" type="required-tonesType" />

<xsd:complexType name="required-country-codesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="country-code"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:element name="country-codes"
   type="required-country-codesType" />

<xsd:complexType name="country-codeType" mixed="true">
 <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="package" type="xsd:string" use="required" />
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="country-code" type="country-codeType" />

<xsd:complexType name="required-h248-codesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="h248-code"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="h248-codes"
   type="required-h248-codesType" />

<xsd:complexType name="h248-codeType" mixed="true">
 <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="package" type="xsd:string" use="required" />
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="h248-code" type="h248-codeType" />

<!--  asr-tts -->

<xsd:complexType name="asr-ttsType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
```

```
      <xsd:element ref="asr-support"
        minOccurs="0" maxOccurs="1" />
      <xsd:element ref="tts-support"
        minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="asr-tts" type="asr-ttsType" />

<xsd:complexType name="asr-supportType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="language"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="asr-support" type="asr-supportType" />

<xsd:complexType name="tts-supportType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="language"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="tts-support" type="tts-supportType" />

<xsd:complexType name="languageType">
 <xsd:complexContent>
```

```
  <xsd:extension base="Tcore">
   <xsd:sequence>
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:attribute ref="xml:lang" />
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="language" type="languageType" />


<!--  vxml -->

<xsd:complexType name="vxmlType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="vxml-mode"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="vxml" type="vxmlType" />

<xsd:complexType name="vxml-modeType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="package" type="xsd:string" use="required" />
    <xsd:attribute name="require" type="vxml.datatype" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="vxml-mode" type="vxml-modeType" />
```

```
<!--  location -->

<xsd:complexType name="locationType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:element ref="ca:civicAddress"
                       minOccurs="1" maxOccurs="1" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

<xsd:element name="location" type="locationType" />


<!--  encryption -->

 <xsd:complexType name="encryptionType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="encryption" type="encryptionType" />

<!--  application-data -->

<xsd:element name="application-data" type="appdata.datatype" />

<!--  max-prepared-duration -->

<xsd:complexType name="max-prepared-durationType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="max-time" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
```

```
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="max-prepared-duration"
  type="max-prepared-durationType" />


<xsd:complexType name="max-timeType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element name="max-time-package" type="xsd:string" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="max-time-seconds" type="xsd:nonNegativeInteger"
     use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="max-time" type="max-timeType" />


<!--  file-transfer-modes -->

<xsd:complexType name="file-transfer-modesType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="file-transfer-mode"
        minOccurs="0" maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="file-transfer-modes"
             type="file-transfer-modesType" />

<xsd:complexType name="file-transfer-modeType">
```

```
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="transfermode.datatype"
     use="required" />
    <xsd:attribute name="package" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="file-transfer-mode" type="file-transfer-modeType" />

<!--  mixers -->

<xsd:complexType name="mixerssessionsType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="mix" minOccurs="0"
        maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="mixers" type="mixerssessionsType" />

<xsd:complexType name="mixType">
  <xsd:complexContent>
   <xsd:extension base="Tcore">
    <xsd:sequence>
      <xsd:element ref="rtp-codec" minOccurs="0"
        maxOccurs="unbounded" />
       <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="users" type="xsd:nonNegativeInteger"
     use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
   </xsd:extension>
  </xsd:complexContent>
```

```
</xsd:complexType>

<xsd:element name="mix" type="mixType" />

<!--  mixing-modes -->

<xsd:complexType name="mixing-modesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
      <xsd:element ref="audio-mixing-modes"
        minOccurs="0" maxOccurs="1" />
      <xsd:element ref="video-mixing-modes"
        minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="mixing-modes" type="mixing-modesType" />

<xsd:complexType name="audio-mixing-modesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="audio-mixing-mode"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="audio-mixing-modes" type="audio-mixing-modesType" />

<xsd:complexType name="audio-mixing-modeType" mixed="true">
 <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="package" type="xsd:string" use="required" />
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

```
<xsd:element name="audio-mixing-mode" type="audio-mixing-modeType" />

<xsd:complexType name="video-mixing-modesType">
 <xsd:complexContent>
  <xsd:extension base="Tcore">
   <xsd:sequence>
     <xsd:element ref="video-mixing-mode"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
       maxOccurs="unbounded" processContents="lax" />
   </xsd:sequence>
   <xsd:attribute name="vas" type="boolean.datatype"
     default="false" />
   <xsd:attribute name="activespeakermix" type="boolean.datatype"
     default="false" />
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="video-mixing-modes" type="video-mixing-modesType" />

<xsd:complexType name="video-mixing-modeType" mixed="true">
 <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax" />
 </xsd:sequence>
 <xsd:attribute name="package" type="xsd:string" use="required" />
 <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="video-mixing-mode" type="video-mixing-modeType" />


<!--
  ######################################################

  DATATYPES

  ######################################################
 -->

<xsd:simpleType name="version.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="1.0" />
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="id.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
</xsd:simpleType>

<xsd:simpleType name="status.datatype">
  <xsd:restriction base="xsd:positiveInteger">
   <xsd:pattern value="[0-9][0-9][0-9]" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="transfermode.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
</xsd:simpleType>

<xsd:simpleType name="action.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="remove" />
   <xsd:enumeration value="update" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="dtmf.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
</xsd:simpleType>

<xsd:simpleType name="boolean.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="true" />
   <xsd:enumeration value="false" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="vxml.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
</xsd:simpleType>

<xsd:simpleType name="appdata.datatype">
  <xsd:restriction base="xsd:string" />
 </xsd:simpleType>

</xsd:schema>
```

12. Security Considerations

The MRB network entity has two primary interfaces -- Publish and Consumer -- that carry sensitive information and must therefore be appropriately protected and secured.

The Publish interface, as defined in and described in Section 5.1, uses the Media Control Channel Framework [RFC6230] as a mechanism to connect an MRB to a Media Server. It is very important that the communication between the MRB and the Media Server is secured: a malicious entity may change or even delete subscriptions to a Media Server, thus affecting the view the MRB has of the resources actually available on a Media Server, leading it to incorrect selection when media resources are being requested by an Application Server. A malicious entity may even manipulate available resources on a Media Server, for example, to make the MRB think no resources are available at all. Considering that the Publish interface is a CFW Control Package, the same security considerations included in the Media Control Channel Framework specification apply here to protect interactions between an MRB and a Media Server.

The Publish interface also allows a Media Server, as explained in Section 5.1.5.18, to provide more or less accurate information about its geographic location, should Application Servers be interested in such details when looking for services at an MRB. While the usage of this information is entirely optional and the level of detail to be provided is implementation specific, it is important to draw attention to the potential security issues that the disclosure of such addresses may introduce. As such, it is important to make sure MRB implementations don't disclose this information as is to interested Application Servers but only exploit those addresses as part of computation algorithms to pick the most adequate resources Application Servers may be looking for.

The Consumer interface, as defined in and described in Section 5.2, conceives transactions based on a session ID. These transactions may be transported either by means of HTTP messages or SIP dialogs. This means that malicious users could be able to disrupt or manipulate an MRB session should they have access to the above-mentioned session ID or replicate it somehow: for instance, a malicious entity could modify an existing session between an Application Server and the MRB, e.g., requesting less resources than originally requested to cause media dialogs to be rejected by the Application Server, or requesting many more resources instead to try and lock as many of (if not all) the resources an MRB can provide, thus making them unavailable to other legitimate Application Servers in subsequent requests. In order to prevent this, it is strongly advised that MRB implementations generate session identifiers that are very hard to

replicate, in order to minimize the chances that malicious users could gain access to valid identifiers by just guessing or by means of brute-force attacks. It is very important, of course, to also secure the way that these identifiers are transported by the involved parties, in both requests and responses, in order to prevent network attackers from intercepting Consumer messages and having access to session IDs. The Consumer interface uses either the Hypertext Transfer Protocol (HTTP) or the Session Initiation Protocol (SIP) as the mechanism for clients to connect to an MRB to request media resources. In the case where HTTP is used, any binding using the Consumer interface MUST be capable of being transacted over Transport Layer Security (TLS), as described in RFC 2818 [RFC2818]. In the case where SIP is used, the same security considerations included in the Media Control Channel Framework specification apply here to protect interactions between a client requesting media resources and an MRB.

Should a valid session ID be compromised somehow (that is, intercepted or just guessed by a malicious user), as a further means to prevent disruption the Consumer interface also prescribes the use of a sequence number in its transactions. This sequence number is to be increased after each successful transaction, starting from a first value randomly generated by the MRB when the session is first created, and it must match in every request/response. While this adds complexity to the protocol (implementations must pay attention to those sequence numbers, since wrong values will cause "Wrong sequence number" errors and the failure of the related requests), it is an important added value for security. In fact, considering that different transactions related to the same session could be transported in different, unrelated HTTP messages (or SIP INVITEs in cases where the In-line mode is being used), this sequence number protection prevents the chances of session replication or disruption, especially in cases where the session ID has been compromised: that is, it should make it harder for malicious users to manipulate or remove a session for which they have obtained the session ID. It is strongly advised that the MRB doesn't choose 1 as the first sequence number for a new session but rather picks a random value to start from. The reaction to transactions that are out of sequence is left to MRB implementations: a related error code is available, but implementations may decide to enforce further limitations or actions upon the receipt of too many failed attempts in a row or of what looks like blatant attempts to guess what the current, valid sequence number is.

It is also worth noting that in In-line mode (both IAMM and IUMM) the MRB may act as a Back-to-Back User Agent (B2BUA). This means that when acting as a B2BUA the MRB may modify SIP bodies: it is the case, for instance, for the IAMM handling multipart/mixed payloads. This

impacts the ability to use any SIP security feature that protects the body (e.g., RFC 4474 [RFC4474], S/MIME, etc.), unless the MRB acts as a mediator for the security association. This should be taken into account when implementing an MRB compliant with this specification.

Both the Publishing interface and Consumer interface may address the location of a Media Server: the Publishing interface may be used to inform the MRB where a Media Server is located (approximately or precisely), and the Consumer interface may be used to ask for a Media Server located somewhere in a particular region (e.g., a conference bridge close to San Francisco). Both Media Server and MRB implementers need to take this into account when deciding whether or not to make this location information available, and if so how many bits of information really need to be made available for brokering purposes.

It is worthwhile to cover authorization issues related to this specification. Neither the Publishing interface nor the Consumer interface provides an explicit means for implementing authentication, i.e., they do not contain specific protocol interactions to ensure that authorized Application Servers can make use of the services provided by an MRB instance. Considering that both interfaces are transported using well-established protocols (HTTP, SIP, CFW), support for such functionality can be expressed by means of the authentication mechanisms provided by the protocols themselves. Therefore, any MRB-aware entity (Application Servers, Media Servers, MRBs themselves) MUST support HTTP and SIP Digest access authentication. The usage of such Digest access authentications is recommended and not mandatory, which means MRB-aware entities MAY exploit it in deployment.

An MRB may want to enforce further constraints on the interactions between an Application Server/Media Server and an MRB. For example, it may choose to only accept requests associated with a specific session ID from the IP address that originated the first request or may just make use of pre-shared certificates to assess the identity of legitimate Application Servers and/or Media Servers.

13. IANA Considerations

There are several IANA considerations associated with this specification.

13.1. Media Control Channel Framework Package Registration

This section registers a new Media Control Channel Framework package, per the instructions in Section 13.1 of [RFC6230].

Package Name: mrb-publish/1.0

Published Specification(s): RFC 6917

Person and email address to contact for further information: IETF MediaCtrl working group (mediactrl@ietf.org), Chris Boulton (chris@ns-technologies.com).

13.2. application/mrb-publish+xml Media Type

To: application

Subject: Registration of media type application/mrb-publish+xml

Type name: application

Subtype name: mrb-publish+xml

Required parameters: none

Optional parameters: Same as charset parameter of application/xml as specified in RFC 3023 [RFC3023].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [RFC3023].

Security considerations: See Section 10 of RFC 3023 [RFC3023] and Section 12 of RFC 6917.

Interoperability considerations: none.

Published specification: Section 10 of RFC 6917.

Applications that use this media type: This media type is used to support a Media Resource Broker (MRB) entity.

Additional Information:

Magic Number: None

File Extension: .xdf

Macintosh file type code: "TEXT"

Person and email address to contact for further information: Chris Boulton (chris@ns-technologies.com).

Intended usage: COMMON

Author/Change controller: The IETF.

13.3. application/mrb-consumer+xml Media Type

To: application

Subject: Registration of media type application/mrb-consumer+xml

Type name: application

Subtype name: mrb-consumer+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter of application/xml as specified in RFC 3023 [RFC3023].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [RFC3023].

Security considerations: See Section 10 of RFC 3023 [RFC3023] and Section 12 of RFC 6917.

Interoperability considerations: none.

Published specification: Section 11 of RFC 6917.

Applications that use this media type: This media type is used to support a Media Resource Broker (MRB) entity.

Additional Information:

Magic Number: None

File Extension: .xdf

Macintosh file type code: "TEXT"

Person and email address to contact for further information: Chris Boulton (chris@ns-technologies.com).

Intended usage: COMMON

Author/Change controller: The IETF.

13.4. URN Sub-Namespace Registration for mrb-publish

IANA has registered the URN "urn:ietf:params:xml:ns:mrb-publish", with the ID of "mrb-publish". The schema of the XML namespace named urn:ietf:params:xml:ns:mrb-publish is in Section 10.

13.5. URN Sub-Namespace Registration for mrb-consumer

IANA has registered the URN "urn:ietf:params:xml:ns:mrb-consumer", with the ID of "mrb-consumer". The schema of the XML namespace named urn:ietf:params:xml:ns:mrb-consumer is in Section 11.

13.6. XML Schema Registration for mrb-publish

IANA has registered the schema for mrb-publish:

URI: urn:ietf:params:xml:schema:mrb-publish

ID: mrb-publish

Filename: mrb-publish

Registrant Contact: IETF MediaCtrl working group (mediactrl@ietf.org)

Schema: The XML for the schema is in Section 10 of this document.

13.7. XML Schema Registration for mrb-consumer

Please register the schema for mrb-consumer:

URI: urn:ietf:params:xml:schema:mrb-consumer

ID: mrb-consumer

Filename: mrb-consumer

Registrant Contact: IETF MediaCtrl working group (mediactrl@ietf.org)

Schema: The XML for the schema is in Section 11 of this document.

14. Acknowledgements

The authors would like to thank the members of the Publish Interface design team, who provided valuable input into this document. The design team consisted of Adnan Saleem, Michael Trank, Victor Paulsamy, Martin Dolly, and Scott McGlashan. The authors would also like to thank John Dally, Bob Epley, Simon Romano, Henry Lum, Christian Groves, and Jonathan Lennox for input into this specification.

Ben Campbell carried out the RAI expert review on an early version of this specification and provided a great deal of invaluable input.

15. References

15.1. Normative References

[ISO.10646.2012]
International Organization for Standardization, "Information technology -- Universal Coded Character Set (UCS)", ISO Standard 10646, 2012.

[ISO.3166-1]
International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions - Part 1: Country codes", ISO Standard 3166-1:2006, 2006.

[ISO.639.2002]
International Organization for Standardization, "Codes for the representation of names of languages -- Part 1: Alpha-2 code", ISO Standard 639, 2002.

[ITU-T.Q.1950]
International Telecommunication Union, "Bearer independent call bearer control protocol", ITU-T Recommendation Q.1950, December 2002.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3311] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC5139] Thomson, M. and J. Winterbottom, "Revised Civic Location Format for Presence Information Data Format Location Object (PIDF-LO)", RFC 5139, February 2008.

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, May 2010.

[W3C.REC-xmlschema-1-20041028]
Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[W3C.REC-xmlschema-2-20041028]
Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

15.2. Informative References

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC4240] Burger, E., Van Dyke, J., and A. Spitzer, "Basic Network Media Services with SIP", RFC 4240, December 2005.

[RFC4474] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, August 2006.

[RFC4733] Schulzrinne, H. and T. Taylor, "RTP Payload for DTMF Digits, Telephony Tones, and Telephony Signals", RFC 4733, December 2006.

[RFC5022] Van Dyke, J., Burger, E., and A. Spitzer, "Media Server Control Markup Language (MSCML) and Protocol", RFC 5022, September 2007.

[RFC5167] Dolly, M. and R. Even, "Media Server Control Protocol Requirements", RFC 5167, March 2008.

[RFC5552] Burke, D. and M. Scott, "SIP Interface to VoiceXML Media Services", RFC 5552, May 2009.

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

[RFC5707] Saleem, A., Xin, Y., and G. Sharratt, "Media Server Markup Language (MSML)", RFC 5707, February 2010.

[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.

[RFC6381] Gellens, R., Singer, D., and P. Frojdh, "The ’Codecs’ and ’Profiles’ Parameters for "Bucket" Media Types", RFC 6381, August 2011.

[RFC6501] Novo, O., Camarillo, G., Morgan, D., and J. Urpalainen, "Conference Information Data Model for Centralized Conferencing (XCON)", RFC 6501, March 2012.

[RFC6505] McGlashan, S., Melanchuk, T., and C. Boulton, "A Mixer Control Package for the Media Control Channel Framework", RFC 6505, March 2012.

Authors' Addresses

Chris Boulton
NS-Technologies

EMail: chris@ns-technologies.com

Lorenzo Miniero
Meetecho
Via Carlo Poerio 89
Napoli 80100
Italy

EMail: lorenzo@meetecho.com

Gary Munson
AT&T
200 Laurel Avenue South
Middletown, New Jersey 07748
USA

EMail: gamunson@gmail.com