# FS – Call Manager

## Content

History

| Version | Date | Adjustments |
|---|---|---|
| A | 2006-10-03 | First revision (MMAWI) |
| B | 2006-12-07 | Added requirement for far end connections.<br>Minor changes in O&M and Call interfaces.<br>Minor changes in statistics.<br>Handle SIP Reason header field with Q.850 cause/location for ViG support. |
| C | 2007-06-11 | GTD processing is supported. CallManager no longer waits a short time after "close(unforced)" before unregistering from SSP; now unregistering is done immediately. Updates on: service loading, PRACK, load regulation, CallManager interface (ermkese, mmath) |
| D | 2008-08-05 | Chapter 4.5.13.3 updated to reflect that ptime configuration was moved to Stream.<br>Updated for RTCP video fast update, FE27 Orange (ematha/estberg). |

# 1 Introduction

This document specifies the functionality of the Call Manager (**CM**) component.

## 1.1 How to read this document

This document draws heavily on SIP call scenarios. The amount of call scenarios and variations of those are many and this function specification defines the most common and important ones. However **CM** must follow the guidelines for a UA in the SIP specification, see [5], where the complete set of scenarios are defined. Those scenarios deal mainly with how to respond, what to do when a certain

response arrives and deals also with timeout of requests etc. **CM** must follow them all and handle all SIP requests and respond in a known way.

All sequence diagrams in this document are on a conceptual level meaning that the exact order of events, especially those dealing with the timings of call control and media stream creations are implementation dependant.

# 2 Definitions

This chapter defines some of the terms used in this document to help understand the functional specification and create a common view of the terms used. All figures unless specified otherwise uses the UML 2.0 notation.

## 2.1 Sessions and Calls

A session is associated with zero or more calls. Typically there is only one call associated with a session but one session can have more than one call as in the case when dealing with call transfer.

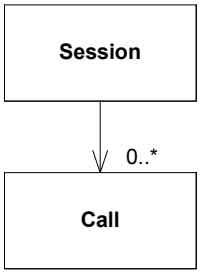


**Figure 1 Session and call relationship**

The **Call** is the actual SIP connection between UA:s and in the case of a call transfer (when two calls exist); between the originating UA to **CM** and from **CM** to the originated UA. All calls are equivalent to **CM**. If any call should have precedence over other calls, it is up to the call client (**CC**).

A **Call** is referred to as a dialog (or call leg) in the SIP standard, see [5].

## 2.2 User Agents (UA)

As defined by the SIP standard, see [5], user agents (**UA**) are the endpoints communicating with each other. A **UA** can be of two types, a user agent client (**UAC**) or a user agent server (**UAS**).

### 2.2.1 User Agent Client (UAC)

A UAC is a UA that creates and sends a SIP request. The user agent has the role of a user agent client for the duration of that initiated transaction. If the user agent receives a SIP request later, it assumes the role of a user agent server for that transaction.

### 2.2.2 User Agent Server (UAS)

A UAS is a UA that generates a response to a SIP request. The response accepts, rejects or redirects the request. This role lasts only for the duration of that transaction (initiated by the UAC). If the user agent generates a SIP request later, it assumes the role of a user agent client for that transaction.

## 2.3 SIP Serving Proxy (SSP)

A SIP Serving Proxy works in this context as a SIP registrar and as a SIP redirecting server as defined in the SIP standard, see [5].

The registrar service handles registrations of active **UA**:s and maintains a list of those **UA**:s used by the redirection service of the SSP.

The redirection service redirects incoming SIP INVITES to the correct **UAS** by examining the incoming INVITE and selecting the **UAS** from the list maintained by the registrar service.

For a full function specification of the SIP Serving Proxy, see [1].

The **SSP** is also used by **CM** as redirecting server when initiating outbound calls.

## 2.4 Call Client (CC)

A Call Client (**CC**) is a component using **CM** in order to receive inbound calls and initiate outbound calls.

## 2.5 Service Enabler

A Service Enabler is a part of a system that implements a protocol server to be used to access a service. **CM** is a service enabler that allows services in the system it is used in to be access through the SIP protocol.

# 3 Function Requirements (Commercial)

The following commercial requirements have been identified.

1. The **Call Manager component** must support the SIP protocol for session initiation, see [5].
2. The **Call Manager component** must support the SDP protocol for media negotiation, see [5].
3. The **Call Manager component** must support the offer/answer model in SDP, see [5].
4. The **Call Manager component** must be able to handle the tel URL, see [5].
5. The **Call Manager component** must support the Cisco Gateway (as used in the MTG-C).
6. The **Call Manager component** must support the Radvision Video Gateway.

# 4 Function Specification (Design Related)

## 4.1 Introduction

The **Call Manager Component, CM,** provides the functionality to handle in- and outbound calls using SIP. It is also responsible for maintaining call state, media negotiation and to initiate media stream creation.

The **CM** is considered to be both a **UAC** and a **UAS**.

The CM is also considered to be a Service Enabler as defined in

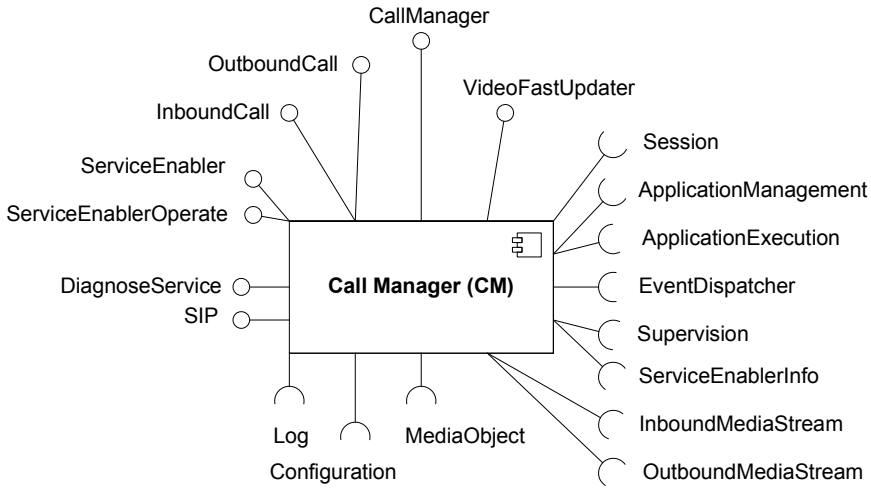


**Figure 2 The Call Manager component**

## 4.2 Exported Interfaces

**CM** has the following exported interfaces:

- The *SIP* interface is the interface towards the phone network via the SIP Serving Proxy (**SSP**), and the SIP gateway.
- The *CallManager, InboundCall,* and *OutboundCall* interfaces where a call is controlled with for example accept or disconnect.
- The *VideoFastUpdater* interface is used to request CM to send a video fast update request over SIP or to request Stream to send it over RTCP. This interface is defined by [3].
- The *ServiceEnabler* interface is for controlling the service enabler aspect of **CM**, e.g. initialize it when it should be used. This interface is defined by [2].
- The *ServiceEnablerOperate* interface is for operational administration of a service enabler, for example open, close and load regulation. This interface is defined by [4].
- The *DiagnoseService* interface is for diagnosing the service enabled by **CM**. This interface is defined by [4] and is completely separate from the other

**CM** functionality in order to not affect normal **CM** behavior. Thus, the implementation of this interface is distributed separate from all other **CM** functionality.

### 4.2.1 SIP Interface (SIP)

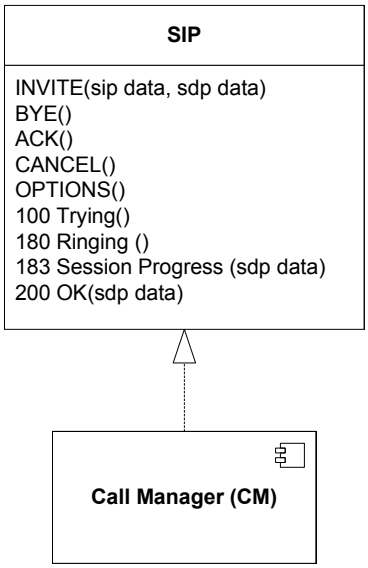


**Figure 3 The SIP interface**

Through this interface external **UA**:s can use functions in **CM** and **CM** can use functions in external **UA**:s and an **SSP**. The figure above illustrates some of the functions in this interface (especially those used in the sequence diagrams in section 4.5.1). For a full description of the SIP protocol see [5].

#### *4.2.1.1 Functions*

For a list of all possible SIP requests and responses, see the SIP Protocol described in [5].

#### *4.2.1.2 Parameter type description*

##### *4.2.1.2.1 Sip data*

This is the data contained in the SIP header of any request or response. For a full description see the SIP Protocol described in [5].

Necessary parts of this data are made available to the **CC**.

##### *4.2.1.2.2 Sdp data*

This is the data contained in the SDP attachment in a SIP request or response. It contains media related information.

For a full description of the SDP see [5].

##### *4.2.1.2.3 Gtd data*

This is a proprietary data used to convert ISDN data to SIP. It is optional. It contains information about the involved parties. See ref. [13]

### 4.2.2 The Call Manager Interface

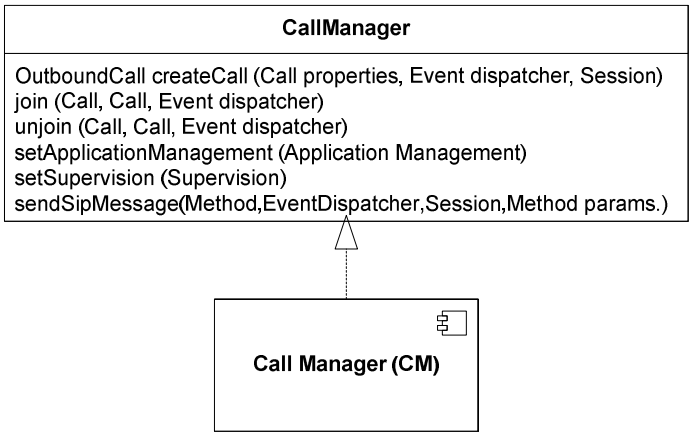


**Figure 4 The Call Manager interface**

This interface provides call management functionality not related to an active call.

#### *4.2.2.1 Functions*

##### *4.2.2.1.1 createCall*

**OutboundCall createCall (Call properties, Event dispatcher, Session)**

This function is used by **CC** to make an outbound call using the given call properties. This method creates an outbound call and initiates the call setup. The new outbound call is returned.

Zero or more Progressing events (see section 4.4.1.11) and eventually also a Connected event (see section 4.4.1.2) is generated as a result of the call setup. If the call setup cannot be performed due to the load situation, a Failed event is generated. The first event may be generated before this function returns and the **CC** must be able to handle that.

The media properties to use for the outbound call are retrieved from the given session as described in section 4.5.7.

The given event dispatcher indicates which event dispatcher to use when issuing call related events to the **CC**.

##### *4.2.2.1.2 join*

**join (Call, Call, Event dispatcher)**

This function is used by **CC** to join two calls together. The media streams of the two calls are joined full duplex. If the calls were joined ok a *Joined* event (see section 4.4.1.8) is generated, otherwise a *Join Error* event (see section 4.4.1.9) is generated. The events are sent using the event dispatcher.

##### *4.2.2.1.3 unjoin*

**unjoin (Call, Call, Event dispatcher)**

This function is used by **CC** to unjoin two calls that was previously joined. The media streams of the two calls are unjoined full duplex. If the calls were unjoined ok an *Unjoined* event (see section 4.4.1.15) is generated, otherwise an *Unjoin Error* event (see section 4.4.1.16) is generated. The events are sent using the event dispatcher.

#### *4.2.2.1.4 setApplicationManagement*

**setApplicationManagement (Application management)**

This function shall be used by **CC** before initializing the service enabler, see section 4.2.5. It sets the application management that is used by **CM** when a service instance shall be loaded due to a new inbound call.

#### *4.2.2.1.5 setSupervision*

**setSupervision (Supervision)**

This function shall be used by **CC** before initializing the service enabler, see section 4.2.5. It sets the supervision object that is used by **CM** to report statistics.

#### *4.2.2.1.6 sendSipMessage*

**sendSipMessage (Method, EventDispatcher, Session, Method parameters)**

This method is used by **CC** to send a SIP message. This method creates a SIP request using the provided method and method parameters, sends it to the configured sip proxy and waits for a response. Currently only a method for message waiting indication (MWI) is supported.

One Send Sip Message Response Event is generated as a result of the sent request. A SIP OK response, SIP Error response or SIP timeout will all generate the same kind of event. The event may be generated before this function returns and the **CC** must be able to handle that.

The session is used for logging purposes.

The given event dispatcher indicates which event dispatcher to use when issuing call related events to the **CC**.

### *4.2.2.2 Parameter type description*

#### *4.2.2.2.1 Application management*

The Application Management interface is specified in [2] and is used to load a new service instance due to an inbound call.

#### *4.2.2.2.2 Call*

The call parameter contains another call, either inbound or outbound.

#### *4.2.2.2.3 Call Properties*

Call properties contain calling party and called party in one of the following formats:

- Telephone number (format as specified in reference [8])
- SIP user (format as the user info part of a SIP or SIPS URI, see [5])
- URI (format as specified in [9])

For the calling party, the call data may also contain a presentation indication which can be Unknown, *Allowed* or *Restricted*.

The call properties also contains the max call duration, max waiting time before the call is connected and whether or not loop back prevention shall be done (see section 4.5.6).

Finally, the call properties contain the call type, i.e. whether it should be a voice or video call.

#### *4.2.2.2.4 Event dispatcher*

During the call, **CM** will send events to the **CC** in certain scenarios (see section 4.5.1). The events are sent using this event dispatcher. The event dispatcher is defined in [2].

#### *4.2.2.2.5 Session*

During the call, CM will retrieve and set data in the relating session. An example of this is when the media properties to use for the call are retrieved from the given session as described in section 4.5.7. The session is defined in [2].

#### *4.2.2.2.6 Supervision*

The Supervision interface is specified in [2] and is among other things used to report statistics.

#### *4.2.2.2.7 Method*

The method controls what kind of SIP message that is sent by the sendSipMessage method. Currently only a method for message waiting indication (MWI) is supported. The MWI method implies sending a SIP NOTIFY containing information for SIP MWI.

#### *4.2.2.2.8 Method parameters (for the MWI method)*

Method parameters are controlling the behaviour of the sendSipMessage method. The parameters are specific to a given method. The only method currently supported is message waiting indication (MWI). The method parameters for MWI contain information about:

- The receiver of the SIP MWI request.
- The mailbox that the MWI is for.
- The current MWI status flag (on/off).
- The number of new and old messages for different media types.

### 4.2.3 The Inbound Call Interface

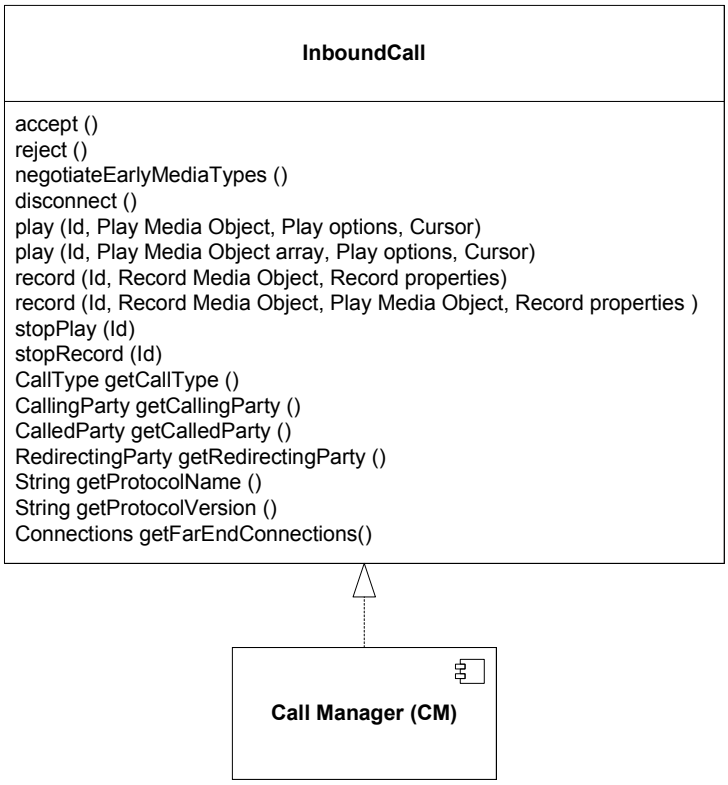


**Figure 5 The Inbound Call interface**

This interface provides call management related to an active inbound call.

#### *4.2.3.1 Functions*

##### *4.2.3.1.1 accept*

**accept ()**

This function is used to accept the inbound call. The media properties to use for the inbound call are retrieved from the given session as described in section 4.5.7.

##### *4.2.3.1.2 reject*

**reject ()**

This function is used to reject the inbound call before it is accepted.

##### *4.2.3.1.3 negotiateEarlyMediaTypes*

**negotiateEarlyMediaTypes ()**

This function is used to indicate to the inbound call that early media will be played and that which media types to use for the call must be negotiated now. The media properties to use in the negotiation for the inbound call are retrieved from the given session as described in section 4.5.7.

An *Early Media Available* event (see section 4.4.1.4) is generated when the negotiation has completed.

#### *4.2.3.1.4 disconnect*

**disconnect ()**

This function is used to disconnect a connected inbound call.

A Disconnected event (see section 4.4.1.3) is generated when the disconnect has completed.

#### *4.2.3.1.5 play*

**play (Id, Play media object, Play option, Cursor)**

**play (Id, Play media object array, Play option, Cursor)**

Plays the media object or an array of media objects on the call's outbound stream using the play option and starts from the cursor. For further details on the play function, see [3].

#### *4.2.3.1.6 record*

**record (Id, Record media object, Record properties)**

**record (Id, Record media object, Play media object, Record properties)**

Records media received on the inbound stream and stores it in the provided record media object. The recording properties indicate how the recording is to be done with regards to for example silence detection.

If a play media object is included, it is played until end and then the audio recording starts. A video recording starts when a video fast update is received.

For further details on the record function, see [3].

#### *4.2.3.1.7 stopPlay*

**stopPlay (Id)**

The play matching the received Id is stopped.

#### *4.2.3.1.8 stopRecord*

**stopRecord (Id)**

The record matching the received Id is stopped.

#### *4.2.3.1.9 getCallType*

**CallType getCallType ()**

Returns the type of call, i.e. whether the call is a voice or a video call. The call type is not determined until the media is negotiated. This means that the call type shall not be retrieved until an Early Media Available or Connected event is received.

#### *4.2.3.1.10 getCallingParty*

**CallingParty getCallingParty ()**

Returns the calling party, i.e. the party that initiated the call.

*4.2.3.1.11 getCalledParty*

**CalledParty getCalledParty ()**

Returns the called party, i.e. the party that received the call.

*4.2.3.1.12 getRedirectingParty*

**RedirectingParty getRedirectingParty ()**

Returns the redirecting party, i.e. the party that redirected the call.

*4.2.3.1.13 getProtocolName*

**String getProtocolName ()**

Returns the name of the protocol used for call setup.

*4.2.3.1.14 getProtocolVersion*

**String getProtocolVersion ()**

Returns the version of the protocol used for call setup.

*4.2.3.1.15 getFarEndConnections*

**Connections getFarEndConnections ()**

Returns the far end connections of a call, i.e. the IP addresses and ports for all SIP and RTP connections. All far end connections are not determined until the outbound streams has been created. This means that the far end connections should not be retrieved until an Early Media Available or Connected event is received.

### *4.2.3.2 Parameter type description*

*4.2.3.2.1 Cursor*

The cursor is described in [3].

*4.2.3.2.2 Id*

Id identifies a specific play or record request.

*4.2.3.2.3 Play Media Object*

A Play Media Object contains the actual media that will be played on the outbound stream (see [10] for details on the media object).

*4.2.3.2.4 Play Options*

The play options are described in [3].

*4.2.3.2.5 Record Media Object*

A Record Media Object contains the actual media that has been recorded from the inbound stream (see [10] for details on the media object).

#### *4.2.3.2.6 Record Properties*

The record properties are described in [3].

### 4.2.4 The Outbound Call Interface

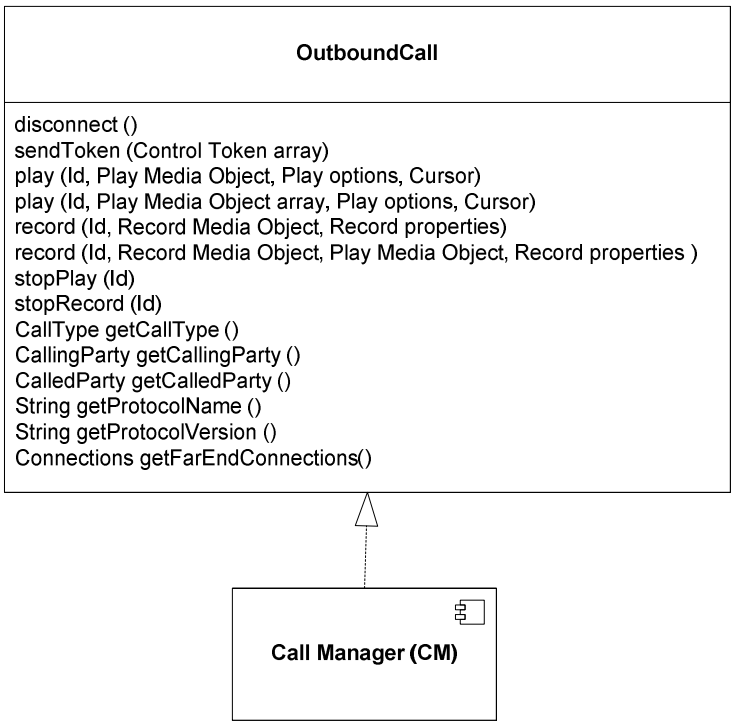


**Figure 6 The Outbound Call interface**

This interface provides call management related to an active outbound call.

#### ***4.2.4.1 Functions***

##### *4.2.4.1.1 disconnect*

**disconnect ()**

This function is used to disconnect a connected outbound call.

A Disconnected event (see section 4.4.1.3) is generated when the disconnect has completed.

##### *4.2.4.1.2 sendToken*

**sendToken (Control token array)**

This function takes the control token and sends it according to the negotiated method, see section 4.5.1.11.

##### *4.2.4.1.3 play*

**play (Id, Play media object, Play option, Cursor)**

**play (Id, Play media object array, Play option, Cursor)**

Plays the media object or an array of media objects on the call's outbound stream using the play option and starts from the cursor. For further details on the play function, see [3].

#### *4.2.4.1.4 record*

**record (Id, Record media object, Record properties)**

**record (Id, Record media object, Play media object, Record properties)**

Records media received on the inbound stream and stores it in the provided record media object. The recording properties indicate how the recording is to be done with regards to for example silence detection.

If a play media object is included, it is played until end and then the audio recording starts. A video recording starts when a video fast update is received.

For further details on the record function, see [3].

#### *4.2.4.1.5 stopPlay*

**stopPlay (Id)**

The play matching the received Id is stopped.

#### *4.2.4.1.6 stopRecord*

**stopRecord (Id)**

The record matching the received Id is stopped.

#### *4.2.4.1.7 getCallType*

**CallType getCallType ()**

Returns the type of call, i.e. whether the call is a voice or a video call. The call type is not determined until the media is negotiated. This means that the call type shall not be retrieved until a Connected event is received.

#### *4.2.4.1.8 getCallingParty*

**CallingParty getCallingParty ()**

Returns the calling party, i.e. the party that initiated the call.

#### *4.2.4.1.9 getCalledParty*

**CalledParty getCalledParty ()**

Returns the called party, i.e. the party that received the call.

#### *4.2.4.1.10 getProtocolName*

**String getProtocolName ()**

Returns the name of the protocol used for call setup.

#### *4.2.4.1.11 getProtocolVersion*

**String getProtocolVersion ()**

Returns the version of the protocol used for call setup.

#### *4.2.4.1.12 getFarEndConnections*

**Connections getFarEndConnections ()**

Returns the far end connections of a call, i.e. the IP addresses and ports for all SIP and RTP connections. All far end connections are not determined until the outbound streams has been created. This means that the far end connections should not be retrieved until a Progressing event indicating Early Media or a Connected event is received.

### *4.2.4.2 Parameter type description*

#### *4.2.4.2.1 Control Token array*

The control token is specified in [3].

#### *4.2.4.2.2 Cursor*

The cursor is described in [3].

#### *4.2.4.2.3 Id*

Id identifies a specific play or record request.

#### *4.2.4.2.4 Play Media Object*

A Play Media Object contains the actual media that is played on the outbound stream (see [10] for details on the media object).

#### *4.2.4.2.5 Play Options*

The play options are described in [3].

#### *4.2.4.2.6 Record Media Object*

A Record Media Object contains the actual media that has been recorded from the inbound stream (see [10] for details on the media object).

#### *4.2.4.2.7 Record Properties*

The record properties are described in [3].

#### *4.2.4.2.8 Transfer Type*

The transfer type specifies if the transfer is blind or bridged (bridged is the only supported transfer so far).

## 4.2.5 The Video Fast Updater Interface

**CM** exports the *VideoFastUpdater* interface as defined in [3].

The *VideoFastUpdater* interface is used to instruct **CM** to request a Video Fast Update over SIP, or to instruct Stream to request it over RTCP.

### 4.2.6 The Service Enabler Interface

**CM** exports the *ServiceEnabler* interface as defined in [2].

The *ServiceEnabler* interface is for controlling the service enabler aspect of **CM**, e.g. initialize it when it should be used.

### 4.2.7 The Service Enabler Operate Interface

**CM** exports the *ServiceEnablerOperate* interface as defined in [4].

The *ServiceEnablerOperate* interface is for operational administration of a service enabler, for example open, close and load regulation. It is for example used to control the amount of active calls using a threshold indicator (see [4]). **CM** uses this threshold to regulate its load according to the method described in section 4.5.1.11.

### 4.2.8 The Diagnose Service

**CM** exports the *DiagnoseService* interface as defined in [4].

The *DiagnoseService* interface is for diagnosing the service enabled by **CM**. This interface is completely separate from the other **CM** functionality in order to not affect normal **CM** behavior. Thus, the implementation of this interface is distributed separate from all other **CM** functionality. How this interface is used is illustrated in section 4.5.1.16.

## 4.3 Imported interfaces

**CM** imports the following interfaces:

- The *ApplicationManagement* and *ApplicationExecution* interfaces (see [2]) are used by **CM** to load a service when an inbound call arrives.
- The *Session* interface (see [2]) is used to set/retrieve session information for a call.
- The *EventDispatcher* interface (see [2]) is used to generate call related events, e.g. that a call was disconnected.
- The *InboundMediaStream and OutboundMediaStream* interfaces (see [3]) are used to create media streams and to play and record media.
- The *MediaObject* interface (see [10]) is used when playing and recording media on the streams.
- The *Supervision* and *ServiceEnablerInfo* interfaces (see [4]) are used to report statistics.
- The *Configuration* interface (see [7]) is used to retrieve configuration.
- The *Log* interface (see [6]) is used to generate logging information.

Note that not all interfaces are used during the scenarios and sequences in this document. Configuration, logging and event dispatching are understood to be used whenever needed and not shown in the scenarios here.

## 4.4 Events

Any events generated or consumed by this component is sent or received through the event mechanism described in [2].

### 4.4.1 Generated

#### *4.4.1.1 Alerting*

This event is generated when a new inbound call has been created. It contains the new call.

#### *4.4.1.2 Connected*

This event is generated when a call has been connected and all media streams are available. It contains the affected call.

#### *4.4.1.3 Disconnected*

This event is generated when either party disconnects the call. It contains the affected call.

The event also contains the reason for the disconnect:

- Far end disconnect
- Near end disconnect
- Call abandoned by far end

The event also indicates if the call already was disconnected when the event occurred. This is the case if **CC** disconnects a call that already was disconnected by remote party.

#### *4.4.1.4 Early Media Available*

This event is generated when early media types have been negotiated for an inbound call and the call is now ready to be used to play media. It contains the affected call.

How **CM** selects the call media types to use is described in section 4.5.7.

#### *4.4.1.5 Early Media Failed*

This event is generated when early media cannot be negotiated for an inbound call due to the fact that the inbound INVITE contained no SDP offer. It contains the affected call. The call is still active and the **CC** can choose between continuing the call setup (although early media cannot be played) or to reject the call.

#### *4.4.1.6 Error*

This event is generated when an error occurs that causes the call to immediately disconnect. It contains the affected call, the direction of the call

(inbound/outbound), whether the call already was considered disconnected when the event occurred, and an error message.

#### *4.4.1.7 Failed*

This event is generated when an outbound call has been rejected (i.e. disconnected before connected) by either the **UAS** or the **UAC**. It contains the affected call, the direction of the call (inbound/outbound) and an error message.

The event also contains the reason for the reject:

- Far end reject
- Near end reject
- Call abandoned by far end
- Call abandoned by near end
- Media negotiation failed

If the reason indicates a far end reject, the event also contains a Network Status Code (see [11]). The reason for the reject (indicated either by a SIP Reason header or by a SIP response code) that caused the reject is mapped to a Network Status Code through configuration as described in section 4.5.3

#### *4.4.1.8 Joined*

This event is generated when joining two calls was done successfully. It contains the two calls that were joined.

#### *4.4.1.9 Join Error*

This event is generated when joining two calls fails. It contains the two calls that should have been joined and an error message describing the error.

#### *4.4.1.10 Not Allowed*

This event is generated when **CC** makes a call requests that is illegal in the current call state, e.g. if the **CC** tries to accept an inbound call that already is connected. It contains the affected call and an error message.

#### *4.4.1.11 Play Failed*

This event is generated when playing media on an outbound stream fails, e.g. if the call was disconnected. The event is defined in [3].

#### *4.4.1.12 Progressing*

This event is generated when a progressing response (such as SIP 180 Ringing or SIP 183 Session Progress) is received for an outbound call. It contains the affected call and whether or not early media is available for the call.

#### *4.4.1.13 Record Failed*

This event is generated when recording media from an inbound stream fails, e.g. if the call was disconnected. The event is defined in [3].

#### *4.4.1.14 Send Token Error*

This event is generated when sending tokens on an outbound call fails. It contains the affected call.

#### *4.4.1.15 Unjoined*

This event is generated when unjoining two calls was done successfully. It contains the two calls that were unjoined.

#### *4.4.1.16 Unjoin Error*

This event is generated when unjoining two calls fails, e.g. if the calls are already unjoined. It contains the two calls that should have been unjoined and an error message describing the error. If unjoin fails, the two calls are treated as unjoined henceforth.

#### *4.4.1.17 Send SIP message response*

This event is generated as a result of the sendSipMessage method. The event corresponds to a successful SIP response, a SIP error response or a SIP timeout. The event will contain the received SIP response code. If the sent SIP request resulted in a timeout it is regarded as if a 408 Request Timeout was received.

### 4.4.2 Consumed

#### *4.4.2.1 Configuration has changed*

This event is consumed by the **CM** and signals that **CM** must reread its configuration. This event is defined in [7].

#### *4.4.2.2 Abandoned Stream Detected*

This event is consumed by **CM** and signals that the inbound RTP stream of a call has been abandoned. **CM** will disconnect the call. This event is defined in [3].

## 4.5 Function Specification

### 4.5.1 Scenarios

The following scenarios are described:

1. Initialization of the service enabler
2. Inbound call with and without early media
3. Outbound call with and without early media
4. Join/unjoin of two calls
5. Reject inbound call
6. Rejected outbound call
7. Inbound call with unsupported media
8. Outbound call with unsupported media

9. Media renegotiation
10. Disconnect call
11. Send outbound DTMF
12. Play outbound media
13. Record inbound media
14. Register/Unregister
15. Operational administration
16. Load regulation
17. Collecting statistics
18. Error handling
19. Diagnose service

To be able to describe the scenarios a **SIP gateway**, a **SIP Serving Proxy** (**SSP**), a **Call Client** (**CC**), an **Operate & Maintain Client (OMC)**, and a **Stream component** (**SC**), are introduced as endpoints to the **CM** interfaces.

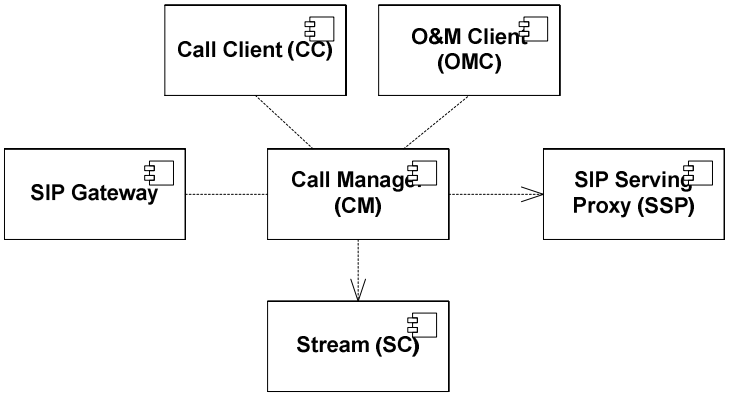


**Figure 7 Call Manager dependencies**

**CM** needs a **SIP Gateway**, **CC, OMC, SSP** and a **SC** as shown in the figure above. The **CC** uses most of **CM**'s interfaces and provides the *ApplicationManagement, ApplicationExecution*, *Session* and *EventDispatcher* interfaces. The **OMC** uses the *ServiceEnablerOperate,* and *DiagnoseService* interfaces and provides the *Supervision* interface. The **SC** provides the *InboundMediaStream* and *OutboundMediaStream* interfaces. The **SSP** provides the *SIP* interface for load balancing and to redirect inbound and outbound calls. Finally, the **SIP Gateway** provides and consumes the *SIP* interface for call control.

**Note!** that PRACK is not included in the described scenarios. For PRACK scenarios, refer to IWD-SIP [5].

#### *4.5.1.1 Initialization of the service enabler*

Before CM opens its SIP service it needs to be initialized. In the initialization, CM is given the host and port to use for the SIP service. It is also given the name of the service to load when a new inbound call is received.

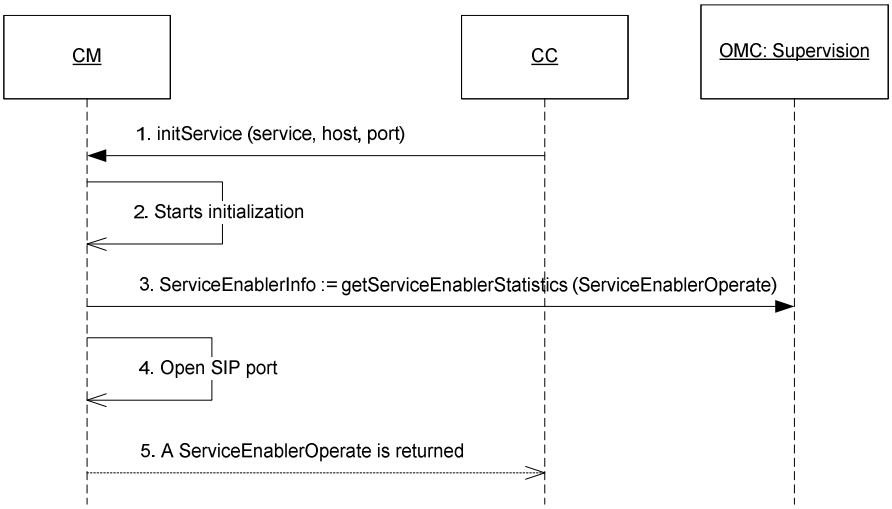


**Figure 8 Initiation of Service Enabler**

The initialization is performed in the following steps:

1. The **CC** initializes the **CM** service enabler. **CM** is given the name of the service to load and the host and port to use for the service. This method is synchronous and when it returns in step 5, **CM** is initialized.
2. **CM** starts it initialization, e.g. the configuration is read at this point.
3. **CM** asks **OMC** for an object to use when reporting call statistics.
4. When **CM** is at the last phase of initialization, the host and port to use for the SIP service is opened.
5. **CM** is now completely initialized and returns a ServiceEnablerOperate instance that can be used by **OMC** to operate **CM**.

#### *4.5.1.2 Inbound call*

An inbound call is a call that is originating outside of **CM** and ends up in **CC**. Usually it originates from a **SIP gateway.** In this scenario the **CM** acts as a **UAS**.

An inbound call both with and without early media is illustrated.

##### *4.5.1.2.1 Without early media*

The scenario for an inbound call without early media is illustrated in the figure below.

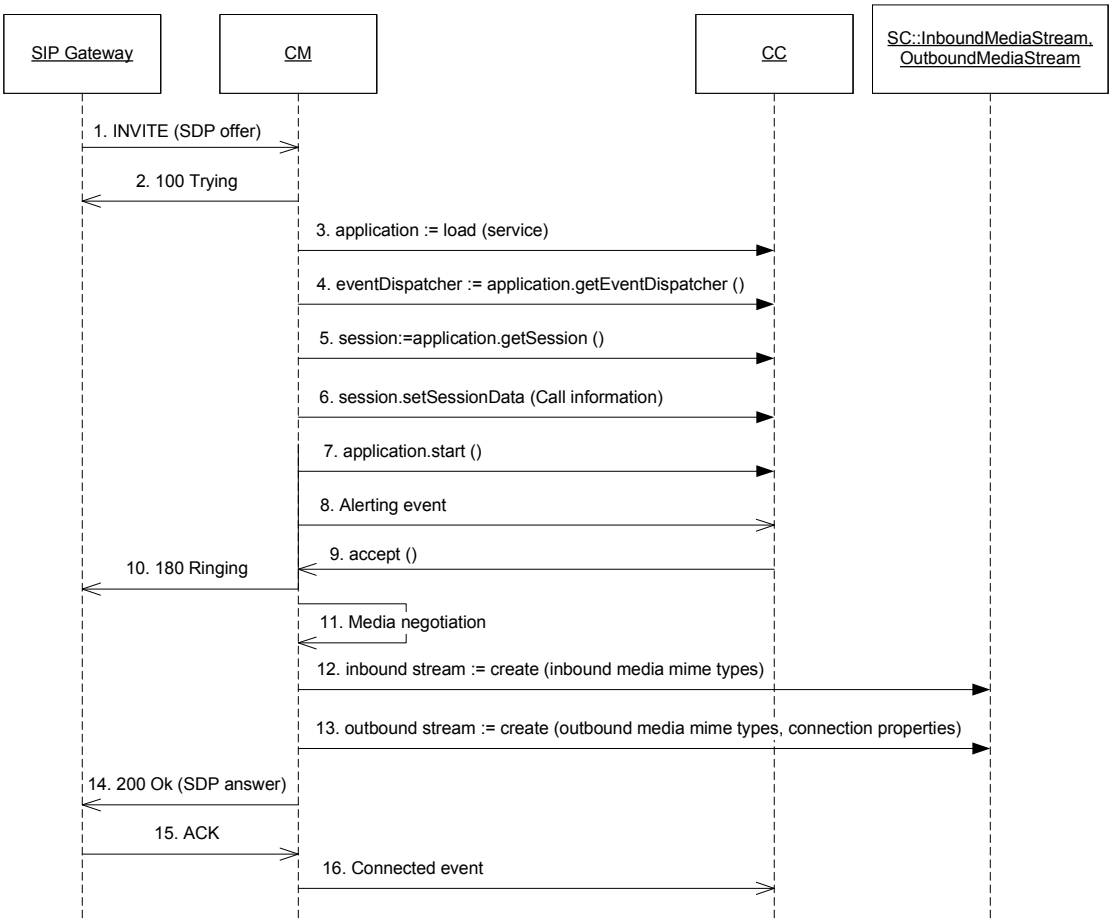


**Figure 9 Inbound call without early media**

In the figure above the call is answered with no early media by the **CC** and the media streams are established after the final ACK from the **SIP Gateway**. The 100 Trying is needed to inform the SIP Gateway that the INVITE has reached **CM** and that the SIP Gateway does not need to resend it in case of timeout.

1. The **UAC** sends an INVITE with the **UAC**:s SDP offer (containing inbound media codecs and protocols) as attachments in the INVITE.
2. **CM** answers the INVITE with a 100 "Trying" response to signal to the **UAC** that the INVITE has arrived.
3. **CM** tells the **CC** to load the service, it returns with an application instance. Refer to chapter 4.5.10 for details on service loading.
4. **CM** retrieves the event dispatcher from **CC**.
5. **CM** retrieves the session from **CC**.
6. **CM** sets call related data into the session. The information that is set is described in section 4.5.1.19.
7. **CM** starts the application.
8. **CM** indicates that a new inbound call is received by sending an Alerting event containing the new call.

9. **CC** accepts the call.
10. **CM** sends a 180 "Ringing" response to signal that the **UAS** has been located and is processing the call.
11. **CM** performs a media negotiation as described in section 4.5.7 which results in an SDP intersection that is used to create an SDP answer and to retrieve connection properties for inbound and outbound media streams.
12. **CM** creates the inbound stream based on the intersected SDP.
13. **CM** creates the outbound stream based on the intersected SDP.
14. **CM** sends a 200 "OK" response containing an SDP answer that was the result of the media negotiation.
15. The **UAC** responds with an ACK to signal that the 200 OK has arrived.
16. **CM** signals to the **CC** that the call is connected by sending a Connected event. It is now possible to play media.

If there is no SDP offer in the inbound INVITE, media negotiation and stream creation is not performed in step 11-13. Instead **CM** creates an SDP offer (as described in 4.5.7) which is sent in the 200 "OK" response. An SDP answer is then received in the ACK which results in a media negotiation and stream creation at that point instead.

#### *4.5.1.2.2 With early media*

The scenario for an inbound call with early media is illustrated in the figure below.

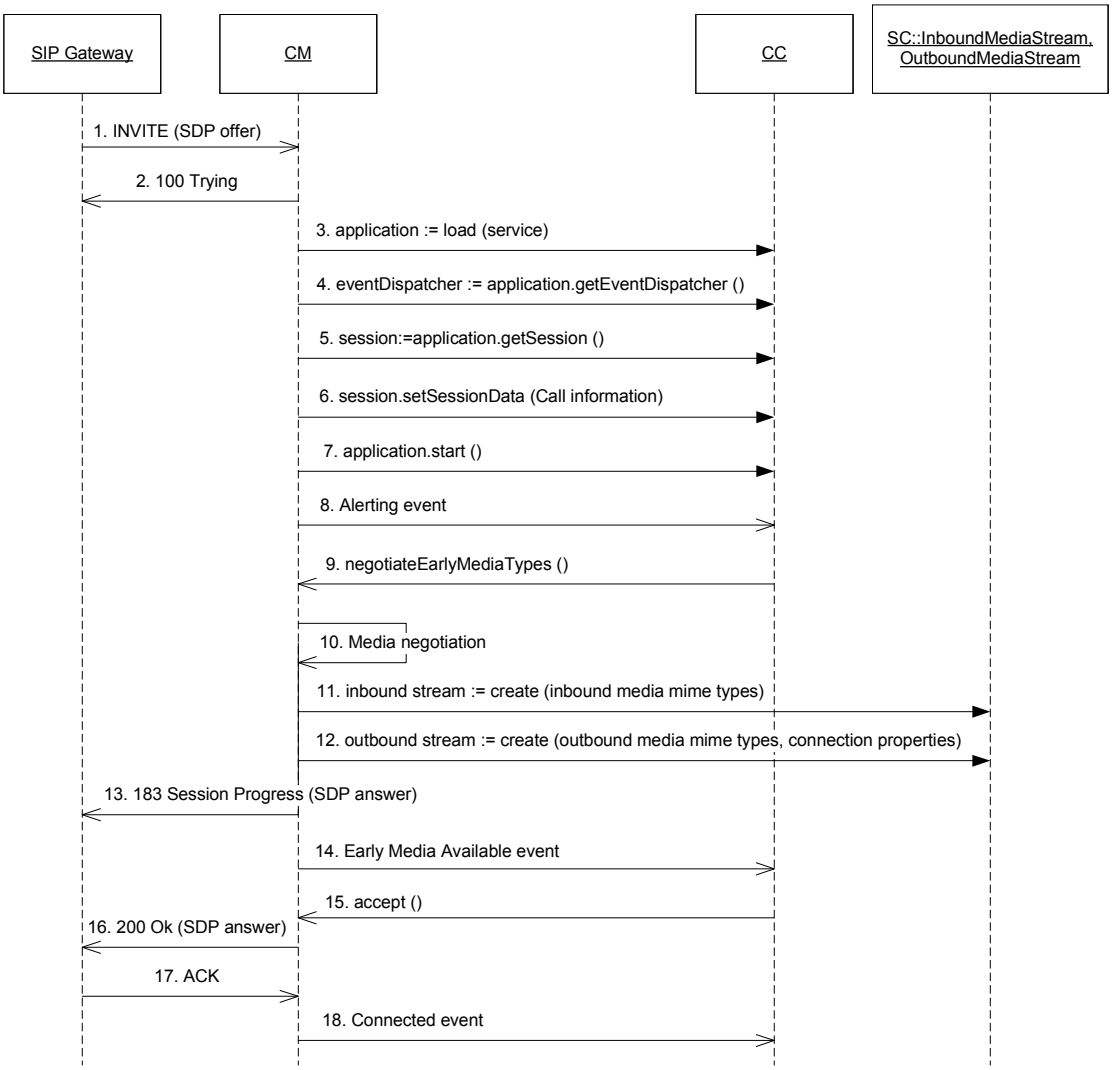


**Figure 10 Inbound call with early media**

In the figure above the **CC** indicates that early media should be used. The media negotiation and stream creation is performed at this point and a 183 "Session Progress" response is sent instead of the 180 "Ringing".

1. The **UAC** sends an INVITE with the **UAC**:s SDP offer (containing inbound media codecs and protocols) as attachments in the INVITE.
2. **CM** answers the INVITE with a 100 "Trying" response to signal to the **UAC** that the INVITE has arrived.
3. **CM** tells the **CC** to load the service, it returns with an application instance.
4. **CM** retrieves the event dispatcher from **CC**.
5. **CM** retrieves the session from **CC**.
6. **CM** sets call related data into the session. The information that is set is described in section 4.5.2.

7. **CM** starts the application.
8. **CM** indicates that a new inbound call is received by sending an Alerting event containing the new call.
9. **CC** indicates to **CM** that it whishes to send early media and that the media types to use for early media must be negotiated.
10. **CM** performs a media negotiation as described in section 4.5.7 which results in an SDP intersection that is used to create an SDP answer and to retrieve connection properties for inbound and outbound media streams.
11. **CM** creates the inbound stream based on the intersected SDP.
12. **CM** creates the outbound stream based on the intersected SDP.
13. **CM** sends a 183 "Session Progress" response containing an SDP answer that was the result of the media negotiation.
14. **CM** sends an Early Media Available event to signal to **CC** that **CM** is available for playing early media.
15. When the **CC** has finished playing the early media it accepts the call.
16. **CM** sends a 200 "OK" response containing the same SDP answer as was sent in the 183 "Session Progress" response.
17. The **UAC** responds with an ACK to signal the arrival of the 200 OK.
18. **CM** signals to the **CC** that the call is connected by sending a Connected event.

If there is no SDP offer in the inbound INVITE, early media cannot be supported. In that case, an Early Media Failed event is sent immediately after step 9 above and the **CC** can choose between continuing the call setup without early media or to reject the call.

#### *4.5.1.3 Outbound calls with or without early media*

Outbound calls are originated from the **CC**. **CC** uses the **CM** to make the calls. In this case **CM** acts as a **UAC.**

All outbound calls are sent from **CM** to an **SSP** which redirects **CM** to the proper **SIP Gateway**. The SSP responds with a SIP redirection response (3xx) containing the contact information for the SIP gateway, see the figure below.

If **CM** is registered towards multiple **SSP**:s (see section 4.5.1.11) one **SSP** is randomly selected for the outbound call.

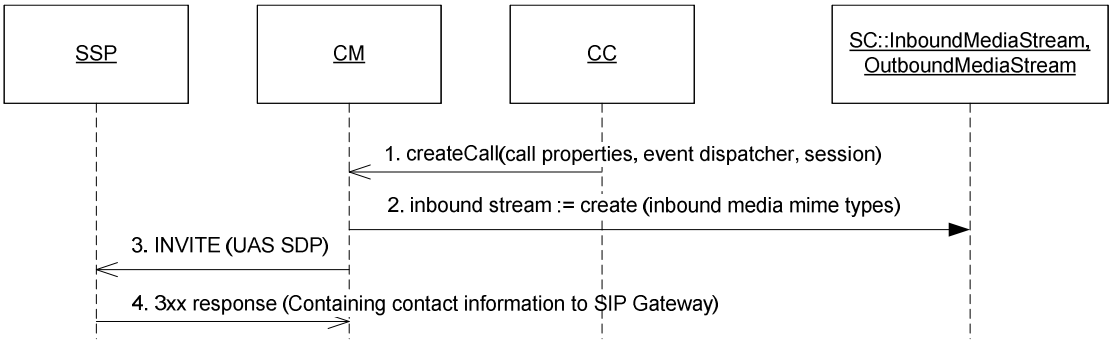


**Figure 11 SSP redirection of outbound calls**

The above create call request and call redirection is excluded in the call scenarios below which starts by sending a redirected INVITE to the **SIP Gateway**.

An outbound call both with and without early media is illustrated.

#### *4.5.1.3.1 Without early media*

The scenario for an outbound call without early media is illustrated in the figure below.

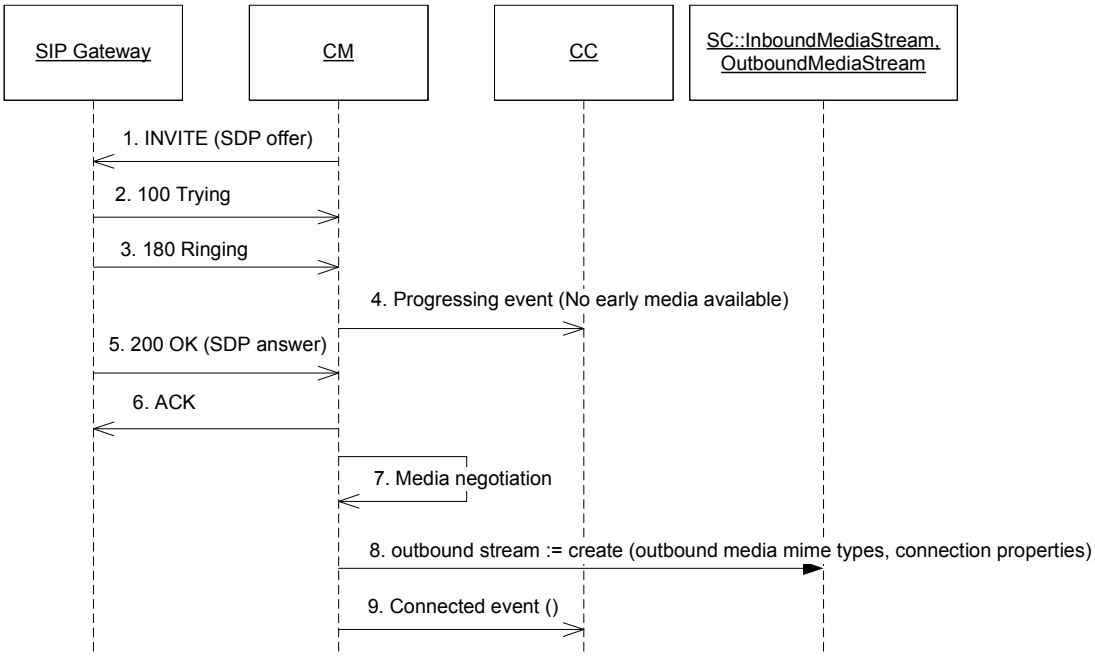


**Figure 12 Outbound call with no early media**

An outbound call without inbound early media is performed in the following steps:

1. **CM** sends an INVITE with media information in an SDP offer.
2. The **UAS** sends a 100 "Trying" to signal the arrival of the INVITE.
3. The **UAS** sends a 180 "Ringing" to inform the **CM** that the endpoint has been located and that there is no early media available.

4. **CM** sends a Progressing event to the **CC** indicating no early media is present.
5. The **UAS** accepts the call by sending a 200 "OK" with an SDP answer.
6. **CM** sends an ACK to the UAS to signal the arrival of the 200 "OK" message.
7. **CM** performs a media negotiation as described in section 4.5.7 which results in an SDP intersection between the SDP offer and the SDP answer.
8. **CM** creates the outbound stream based on the intersected SDP.
9. **CM** signals to the **CC** that the call is connected by sending a Connected event.

#### *4.5.1.3.2 With early media*

The scenario for an outbound call with early media is illustrated in the figure below.

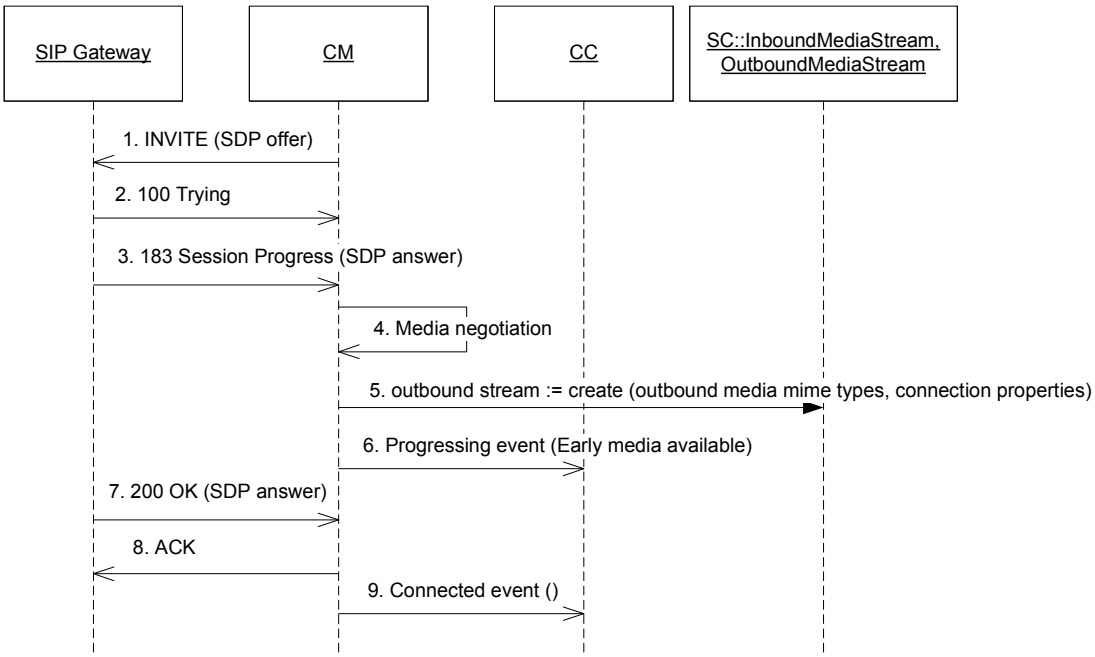


**Figure 13 Outbound call with incoming early media**

An outbound call with inbound early media is performed in the following steps:

1. **CM** sends an INVITE with media information in an SDP offer.
2. The **UAS** sends a 100 "Trying" to signal the arrival of the INVITE.
3. The **UAS** sends a 183 "Session Progress" with an SDP answer indicating that early media is available.

4. **CM** performs a media negotiation as described in section 4.5.7 which results in an SDP intersection between the SDP offer and the SDP answer.
5. **CM** creates the outbound stream based on the intersected SDP.
6. **CM** sends a Progressing event to the **CC** indicating that early media is available. It is now ok for the **CC** to retrieve media from the call or to join the media streams of this call to another.
7. The **UAS** accepts the call by sending a 200 "OK" with the same SDP answer as in the "Session Progress".
8. **CM** sends an ACK to the UAS to signal the arrival of the 200 "OK" message.
9. **CM** signals to the **CC** that the call is connected by sending a Connected event.

#### *4.5.1.4 Join/Unjoin of two calls*

For a call transfer, the media streams of one call shall be joined with the media streams of another call. **CM** supports a full duplex join (and corresponding unjoin) between two calls.

An inbound call can be joined with another call when it has been connected, i.e. when a Connected event has been generated for the call.

An outbound call can be joined with another call when it is connected or when there is early media available, i.e. when a Connected event or a Progressing event indicating early media has been generated for the call.

A join results in the inbound media stream of one call being connected to the outbound media stream of the other call and vice versa.

When a join is completed a Joined event is sent. If the join failed, a Join Error event is sent.

An unjoin has the opposite effect and shall be used when the two calls no longer should be joined together.

When an unjoin is completed an Unjoined event is sent. If the unjoin failed, an Unjoin Error event is sent.

#### *4.5.1.5 Reject an inbound call*

Whenever the **CC** does not want to accept an inbound call it must reject it. The call reject is done instead of accepting the call.

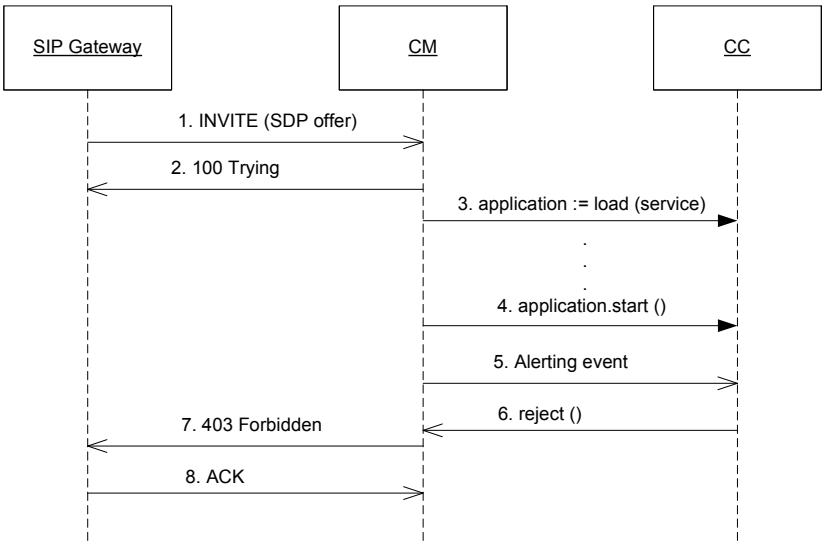


**Figure 14 Reject an inbound call**

Rejecting an inbound call is performed in the following steps:

1. The **UAC** sends an INVITE to **CM**.
2. **CM** answers the INVITE with a 100 "Trying" response to signal to the **UAC** that the INVITE has arrived.
3. **CM** loads the application and sets session data as described earlier for an inbound call.
4. When session data has been set, **CM** starts the application instance.
5. **CM** indicates that a new inbound call is received by sending an Alerting event containing the new call.
6. **CC** decides to reject the call.
7. **CM** sends a 403 "Forbidden" response indicating to the **UAC** that the call failed.
8. The **UAC** responds with an ACK to signal the arrival of the SIP final response.

#### *4.5.1.6 Rejected outbound call*

Any outbound call can be rejected by the **UAS** (e.g. by the end user) and the **UAS** will in that case answer with a SIP final response. Which final responses that are handled as a call reject is described in section 4.5.4.

The create call request and SSP call redirection illustrated in Figure 11 is excluded in the call scenario below which starts by sending a redirected INVITE to the **SIP Gateway**.

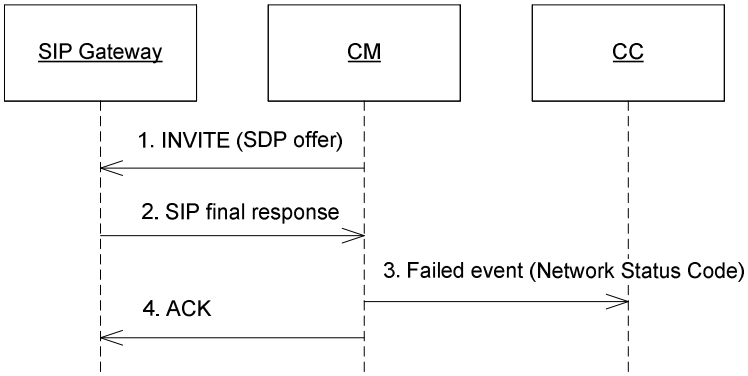


**Figure 15 Rejected outbound call**

The outbound call is rejected in this scenario immediately after the INVITE. The reason for the reject is then mapped to a Network Status Code based on configuration as described in section 4.5.3.

1. **CM** sends an INVITE with media information in an SDP offer.
2. The **UAS** declines the call with a SIP final response. The content of the response is then mapped to a Network Status Code using the **CM** configuration.
3. **CM** sends a Failed event together with the Network Status Code to the **CC**.
4. **CM** acknowledges the arrival of the response.

#### *4.5.1.7 Inbound call with unsupported media*

For an inbound call the SDP answer created by **CM** must be the intersection between the desired media types given by **CC** and the **UAS** SDP offer as described in section 4.5.7. If this intersection is empty the call cannot be completed.

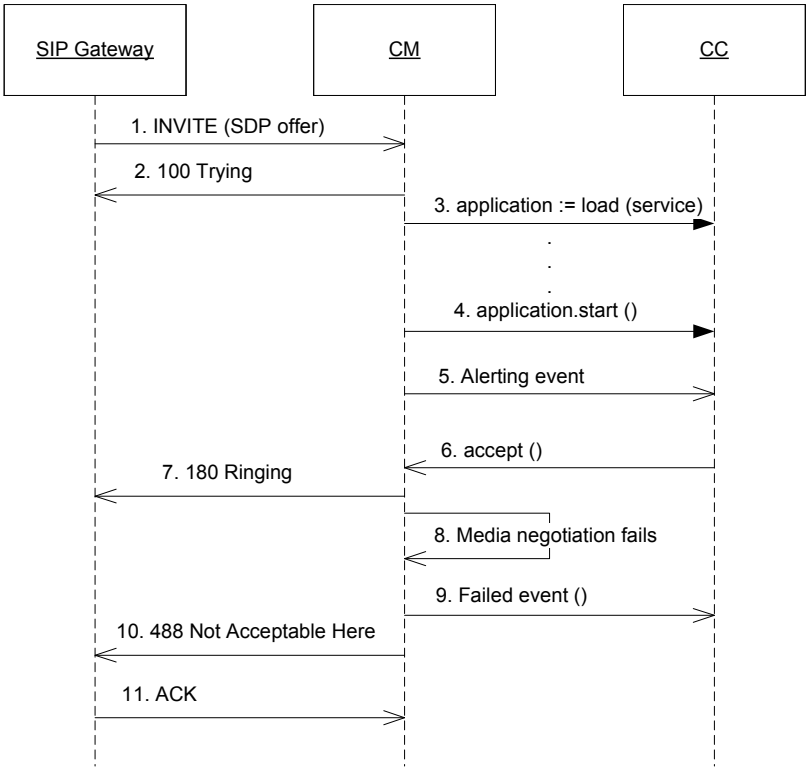


**Figure 16 Unsupported media for inbound call**

In this scenario the intersection of the **UAC** SDP offer and the desired media types from the **CC** is empty and **CM** must respond with a 488 "Not Acceptable Here" response.

**1.** The **UAC** sends an INVITE to **CM**.

**2.** **CM** answers the INVITE with a 100 "Trying" response to signal to the **UAC** that the INVITE has arrived.

**3.** **CM** loads the application and sets session data as described earlier for an inbound call.

**4.** When session data has been set, **CM** starts the application instance.

**5.** **CM** indicates that a new inbound call is received by sending an Alerting event containing the new call.

**6.** **CC** accepts the call.

**7.** **CM** sends a 180 "Ringing" response to signal that the **UAS** has been located and is processing the call.

**8.** **CM** performs a media negotiation as described in section 4.5.7 which results in no SDP intersection found.

**9.** **CM** signals to **CC** that the call failed due to media negotiation failure by sending a Failed event.

**10.** **CM** sends a 488 "Not Acceptable Here" response indicating to the **UAC** that the media suggestion was unacceptable.

**11.** The **UAC** responds with an ACK to signal the arrival of the SIP final response.

#### *4.5.1.8 Outbound call with unsupported media*

An outbound call can also fail because the **UAS** does not support the media requested by **CM**. This is the same scenario as illustrated in section 4.5.1.6. In this situation the **SIP Gateway** send a 488 "Not Acceptable Here" response.

#### *4.5.1.9 Media renegotiation*

In SIP media renegotiation is possible and done by sending a new INVITE after a call has been entirely established. In SIP media renegotiation does not have to be supported and may be rejected.

Media renegotiation is not supported by **CM**. The new INVITE is rejected by **CM** as illustrated in the figure below.

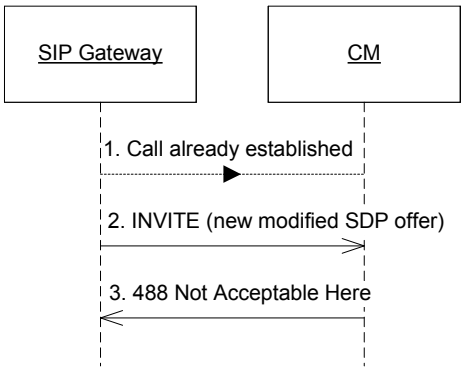


**Figure 17 Media renegotiation**

**1.** A call has already been established according to earlier scenarios for call setup.

**2.** The **UAC** sends an INVITE containing new inbound media codecs and protocols to use in an SDP offer.

**3.** **CM** rejects the INVITE with a 488 Not Acceptable Here response to signal to the **UAC** that media renegotiation not is allowed.

#### *4.5.1.10 Disconnect a call*

A call disconnect can be initiated from both sides, i.e. both from the terminal or from the **CC**. The scenarios below apply to both inbound and outbound calls.

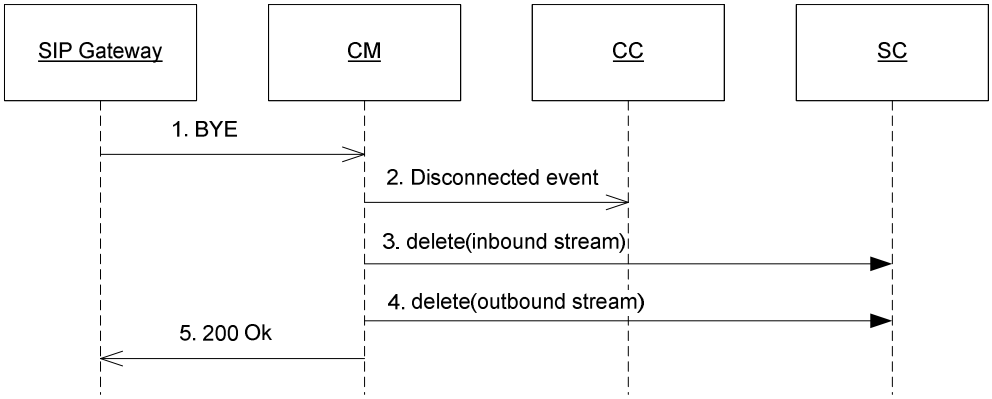


**Figure 18 Terminal initiated disconnect of inbound or outbound call**

The above sequence diagram shows the terminal initiated disconnect procedure for both inbound and outbound calls.

1. The **UA** sends a BYE request to **CM**.
2. **CM** sends a Disconnected event to the **CC**.
3. **CM** deletes the inbound stream.
4. **CM** deletes the outbound stream.
5. **CM** sends a 200 "OK" message back to the **UA** to signal that the call has been released.

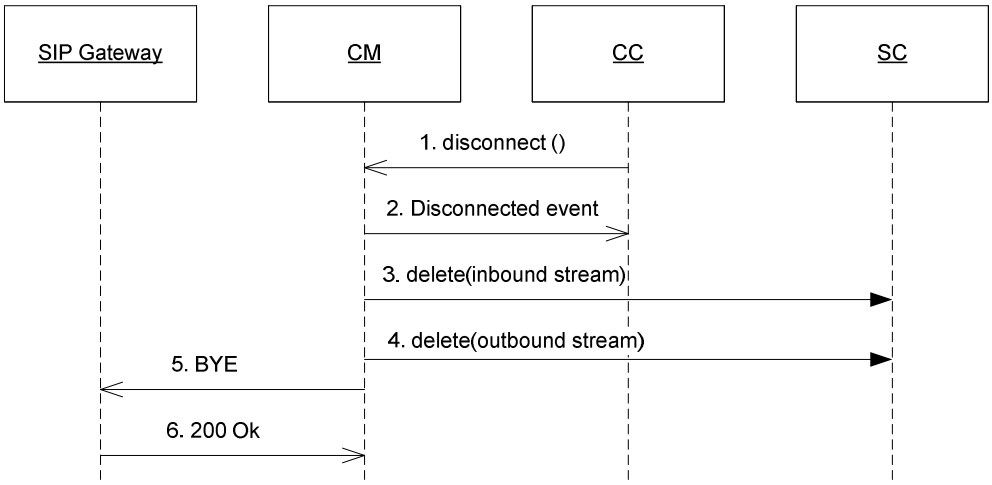


**Figure 19 CC initiated hang-up of inbound or outbound call**

The above sequence diagram shows the **CC** initiated disconnect procedure for both inbound and outbound calls.

1. **CC** calls the disconnect function in **CM** to disconnect the call.
2. **CM** sends a Disconnected event to the **CC**.
3. **CM** deletes the inbound stream.
4. **CM** deletes the outbound stream.

5. **CM** sends the BYE request to the **UA**.
6. The **UA** sends a 200 "OK" message back to **CM** to signal that the call has been released.

#### *4.5.1.11 Send outbound DTMF*

Outbound DTMF can currently only be sent in one way by **CM**; as a packet with a specific payload type in the outbound RTP stream. If the media type indicating outbound DTMF using this mechanism (i.e. the media type "audio/telephone-event") is configured mandatory in **CM**, **CM** will check if the peer **UA** supports this media type or not during the media negotiation. If this method is not supported by the peer **UA** but configured mandatory in **CM** the call will be rejected due to unsupported media type as described in sections 4.5.1.7 and 4.5.1.8.

Outbound DTMF can only be sent for outbound calls.

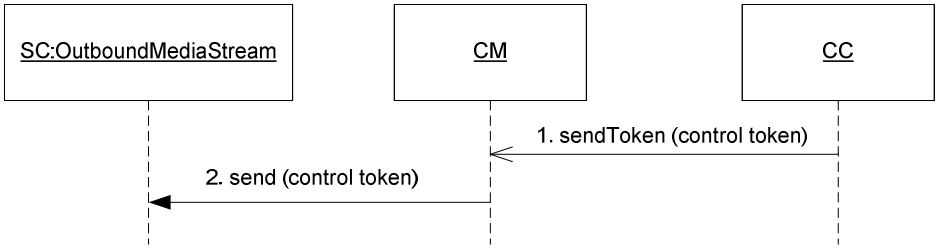


**Figure 20 Outbound DTMF using RTP**

**CC** sends a token to **CM** which uses **SC** to send DTMF over RTP:

1. **CC** wants to send an outbound control token on an outbound call.
2. **CM** sends the control token to **SC** which is responsible for sending it over the outbound stream.

If **CM** fails to send the tokens, a Send Token Error event is sent.

#### *4.5.1.12 Play outbound media*

When media has been negotiated for a call (inbound or outbound) it is possible to play media on the outbound stream.

For an inbound call media can be played when the call is connected or if early media types have been negotiated, i.e. when a Connected event or an Early Media Available event has been generated for the call.

For an outbound call media can be played when the call is connected, i.e. when a Connected event has been generated for the call.

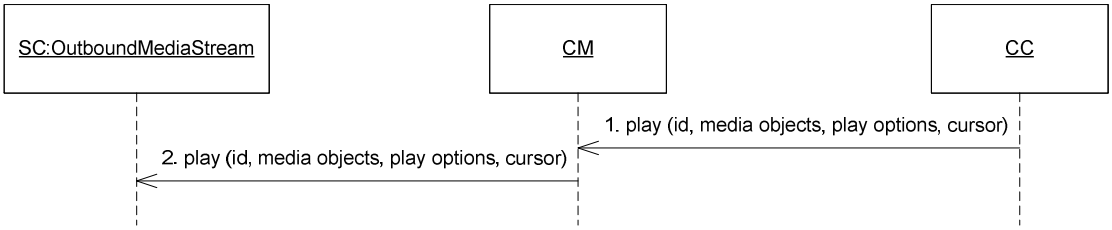


**Figure 21 Play media**

**CC** requests media to be played on the outbound media stream:

1. **CC** wants to play outbound media on a call.
2. **CM** requests the OutboundMediaStream in **SC** to play media.

If **CM** fails to play media, a Play Failed event is sent.

### *4.5.1.13 Record inbound media*

When media has been negotiated for a call (inbound or outbound) it is possible to record media from the inbound stream.

For an inbound call media can be recorded when the call is connected, i.e. when a Connected event has been generated for the call.

For an outbound call media can be recorded when the call is connected or when there is early media available, i.e. when a Connected event or a Progressing event indicating early media has been generated for the call.

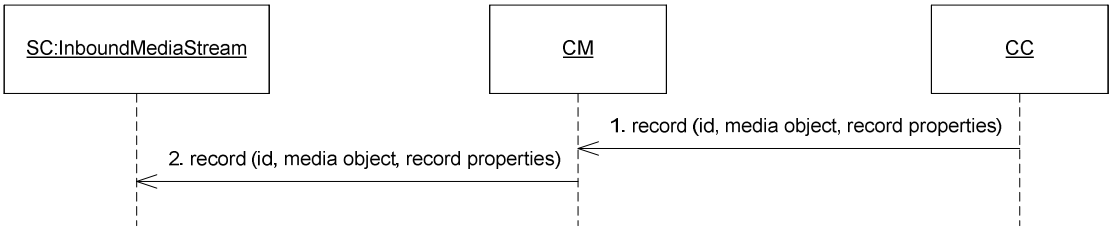


**Figure 22 Record media**

**CC** requests media to be recorded from the inbound media stream:

1. **CC** wants to record inbound media on a call.
2. **CM** requests the InboundMediaStream in **SC** to record media.

If **CM** fails to record media, a Record Failed event is sent.

### *4.5.1.14 Register/Unregister*

**CM** registers its presence to an **SSP** by sending a SIP REGISTER request to the **SSP**. This means that the **SSP** can start redirecting calls to the **CM**. **CM** only sends outbound calls to **SSP**:s to which it is registered.

When **CM** registers itself in the **SSP** it includes no expiration time indicating that it is up to the **SSP** to decide how long the registration shall be valid.

Which **SSP** to register to is configurable and **CM** supports multiple **SSP**:s.

The scenario for the register procedure:

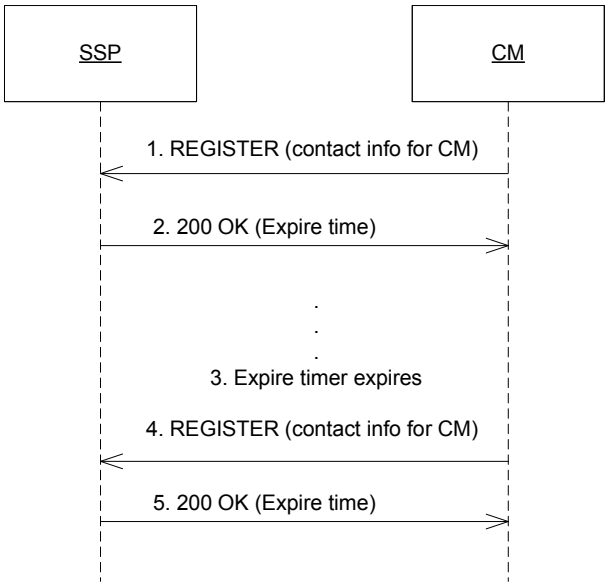


**Figure 23 CM registers in SSP**

1. **CM** sends a SIP REGISTER request to the **SSP** with its contact address.
2. The **SSP** sends back a 200 OK when the registration has been completed containing the expire time after which the registration will be removed. CM schedules a timer with the expire time.
3. The expire timer expires.
4. **CM** resends the same SIP REGISTER request to the **SSP** to renew its registration.
5. The **SSP** sends back a 200 OK when the registration has been completed with the expire time for the current registration.

**CM** unregisters from an **SSP** when it no longer whishes to establish calls. This is the method for telling the **SSP** that the **CM** can take no more calls. Both receiving inbound calls and initiating outbound calls will be impossible if **CM** is not registered in any **SSP**. This mechanism is used for load regulation, see section 4.5.1.15.

The scenario for the unregister procedure:

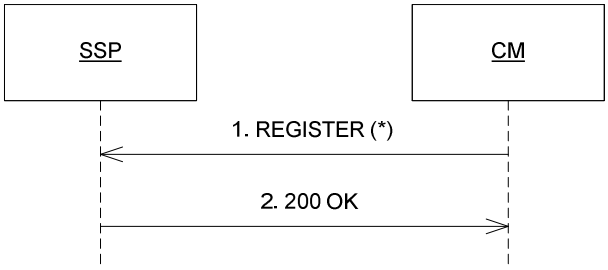


**Figure 24 CM unregisters from SSP**

1. **CM** sends a SIP REGISTER request to the **SSP** with its contact address set to "*" which indicates an unregister.
2. The **SSP** sends back a 200 OK when the unregister has been completed.

### *4.5.1.15 Operational administration*

Using the ServiceEnablerOperate interface of **CM** it is possible to administer **CM** with respect to O&M.

When to register/unregister towards an SSP depends upon the administrative state of **CM**. It is possible to modify the administrative state using the methods *close* and *open*. **CM** always starts up in closed state. When **CM** is in opened state it will try to register towards the configured SSP:s, otherwise it will make sure it is unregistered.

#### *4.5.1.15.1 Open*

When **CM** shall be able to accept inbound calls and send outbound calls, **OMC** can use the open method in the ServiceEnablerOperate interface. **CM** immediately registers towards the configured SSP:s.

When the open is complete, this is reported to **OMC** using the method *opened* on the ServiceEnablerInfo instance received at initialization (see section 4.5.1.1).

#### *4.5.1.15.2 Forced Close*

When all current calls shall be immediately discarded and no more calls shall be accepted, **OMC** can use the close method in the ServiceEnablerOperate interface with the *forced* parameter set to *true*. All current calls are immediately disconnected by **CM** using the scenario described in 4.5.1.10 (except for the disconnect request from **CC** which does not occur in this scenario) and **CM** unregisters from **SSP** to avoid new calls. New incoming calls that are received after the forced close has been requested are rejected.

When the forced close is complete, this is reported to **OMC** using the method *closed* on the ServiceEnablerInfo instance received at initialization (see section 4.5.1.1).

#### *4.5.1.15.3 Unforced Close*

When no more calls shall be accepted, but all current calls are allowed to complete, **OMC** can use the close method in the ServiceEnablerOperate interface with the *forced* parameter set to *false*. **CM** unregisters from **SSP** to avoid receiving new calls, but all current calls are allowed to run until completed.

When the unforced close is complete, this is reported to **OMC** using the method *closed* on the ServiceEnablerInfo instance received at initialization (see section 4.5.1.1).

#### *4.5.1.16 Load regulation*

Using the ServiceEnablerOperate interface of **CM** it is possible to regulate the amount of concurrent calls.

When to register/unregister towards an SSP depends upon the administrative state of **CM** (as described in 4.5.1.15) but also on the maximum amount of concurrent calls that can be supported.

It is possible to modify the amount of concurrent calls using the method *updateThreshold*.

If the amount of active calls is above the threshold, **CM** unregisters from the **SSP** so the **SSP** will stop redirecting calls to this **CM**. When the amount of active calls falls below the threshold, **CM** will register its location to the **SSP** again so the **SSP** can start redirecting calls to this **CM** assuming of course that the administrative state of **CM** is opened. Otherwise no registration is performed until the administrative state becomes opened.

To obtain a more stable load regulation, there is not just one threshold but also a high watermark (HWM) and a low watermark (LWM) where the following rule applies:

LWM <= HWM <= max threshold

When the amount of concurrent calls rises to HWM, CM unregisters from SSP:s. When the amount of concurrent calls falls to the LWM, CM registers in SSP:s.

When the amount of concurrent calls is larger or equal to HWM, CM will reject incoming OPTIONS requests but still accept incoming INVITE requests.

When the amount of concurrent calls is larger or equal to max threshold, CM will reject all incoming requests.

At startup of **CM** HWM, LWM and max threshold is set to a low initial value and after time ramped up to the configured values.

#### *4.5.1.17 Collecting Statistics*

**CM** updates status and counters available to the **OMC** using the ServiceEnablerInfo instance received at initialization (see section 4.5.1.1).

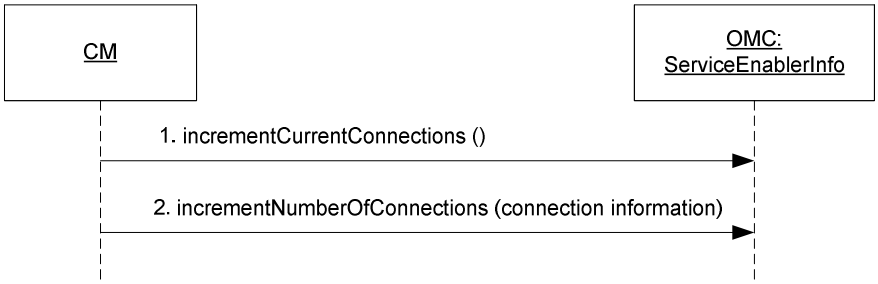


**Figure 25 Update statistics**

The scenario shows how **CM** updates the ServiceEnablerInfo instance with information regarding statistics. The updates are done when any of the counters changes to be able to show real time values for the SNMP supervision and ISP calculations.

Whenever statistical information changes (e.g. a new call is received or a call is disconnected) the service enabler statistics is updated for the amount of current connections. Statistics is also updated for the current event, e.g. statistics is incremented each time a call becomes connected.

The current calls statistics shall be increased for each new call. Since the call type is not determined until the media negotiation has been performed, this statistics is first incremented with the call type *unknown*. When the call becomes connected the statistics is updated with the correct call type. If the call is rejected or fails due to an error prior to connected, current calls are decremented for the call type *unknown*. If the error occurred after connect was complete, current calls are decremented for the known call type.

Statistics is collected for the following types of calls/events:

- **Current calls**:
  Incremented as soon as a call is initiated. When a call becomes connected the call type is updated.
  Decremented when a call becomes rejected, disconnected or an error occurs while not already disconnected, rejected or in error state.
- **Connected calls**:
  Incremented when a call becomes connected and a Connected event is sent.
- **Rejected calls**:
  Incremented when the call is rejected prior to connect (as described in sections 4.5.1.5 and 4.5.1.6) and a Failed event is generated.
- **Abandoned calls**:
  Incremented when the Abandoned Stream Detected event is received for a connected call. At this point the call is disconnected (generating a Disconnected event). The event contains information indicating if the call was abandoned.
- **Abandoned Rejected calls**:
  Incremented when the Abandoned Stream Detected event is received during early media of an outbound call or when an inbound call has been abandoned by the **CC** (i.e. the call has not been accepted in time). At this point the call is cancelled or rejected (generating a Failed event). The event contains information indicating if the call was abandoned.

- **Far end disconnected calls**:
  Incremented when the peer **UA** disconnects a call using a SIP BYE request. A Disconnected event containing information indicating that the call was disconnected by far end is generated.
- **Near end disconnected calls**:
  Incremented when the **CC** disconnects a call. A Disconnected event containing information indicating that the call was disconnected by near end is generated.
- **Errors**:
  Incremented when an error occurs for the call and an Error event is generated.
- **Dropped packets**:
  At the end of a call before streams are deleted, **CM** asks **SC** for the amount of dropped RTP packets on the inbound media stream and this statistics is updated.

For inbound calls, the sum of connected calls, rejected calls and abandoned rejected calls shall be equal to the total amount of received inbound calls.

The sum of far end disconnected calls, near end disconnected calls and abandoned calls shall be equal to the amount of connected calls.

The above describe statistics is collected separately for each call type and call direction. This is done in order to be able to separate e.g. the amount of rejected inbound video calls from the amount of rejected outbound voice calls.

#### *4.5.1.18 Failure*

In a failure situation one of two types of events is generated by **CM**: Not Allowed event and Error event.

A Not Allowed event is generated when **CC** requests some action that is not allowed in the current call state, e.g. if **CC** tries to accept a call that already is disconnected.

An Error event is generated due to one of the following situations and results in the call being rejected or disconnected:

- **Internal Error**:
  This type of error is non-recoverable.
- **Timeout Error**:
  The **CM** experienced a non-recoverable timeout or transaction error.

#### *4.5.1.19 Diagnose Service*

**CM** provides means to survey its operational status with the *DiagnoseService* interface. The **OMC** can request to send a diagnosing request and observe the result of that request as illustrated in the figure below.

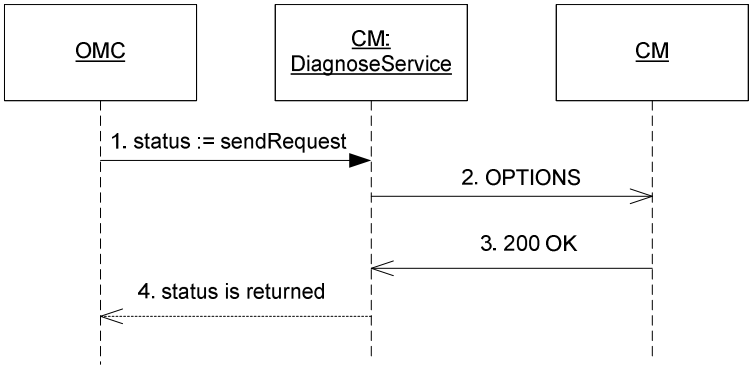


**Figure 26 Diagnose CM service**

1. The **OMC** sends a diagnose request to **CM** using the *DiagnoseService* interface. The implementation of this interface is completely separate from the other **CM** functionality. The call to sendRequest is blocking, and does not return until the status is retrieved or a timeout occurred.
2. A SIP OPTIONS request is sent to **CM** using the *SIP* interface.
3. A 200 OK response is returned containing the header Experienced-Operational-Status with the value set to up, down or impaired.
4. When the diagnose request has completed, the status of the request is returned.

The 200 "OK" response from **CM** contains the Experienced-Operational-Status header indicating the status as one of up or down. If no response is received in time, a timeout occurs and the status is considered down.

#### *4.5.1.20 Send SIP Message Waiting Indicator (SIP MWI)*

The following scenario describes the usage of the send SIP message method with the message waiting indication (MWI) method. A SIP NOTIFY is sent to a remote SIP user agent. The remote user agent responds with a SIP final response, either an OK response or an error response.

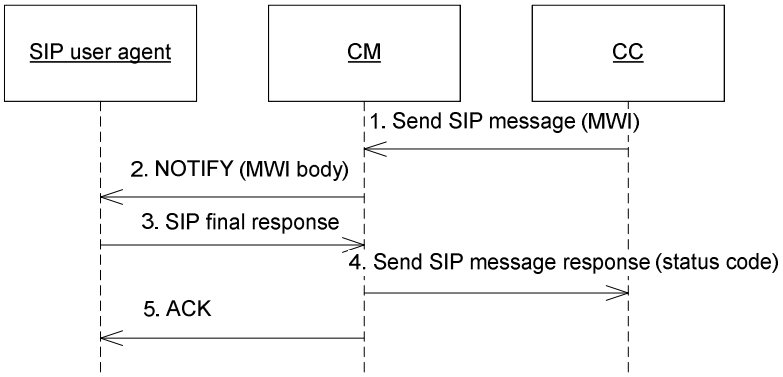


**Figure 27 Send SIP MWI**

The SIP MWI (sent as a SIP NOTIFY request) is responded to with either an OK response or an error response. The result of the NOTIFY is sent back as a status code in the Send SIP message response event.

1. **CC** request that a SIP MWI shall be sent.
2. **CM** sends a SIP NOTIFY request containing MWI parameters.
3. The SIP **UA** responds to the request.
4. **CM** sends a Send SIP message response event containing the SIP response code from the response. If the SIP response contain a Retry-After header, the retry-after value is also included in the event.
5. **CM** acknowledges the arrival of the response.

#### *4.5.1.21 Send SIP Message Waiting Indicator (SIP MWI) with timeout*

The send SIP message method sends a SIP NOTIFY to a remote SIP user agent. The remote user agent does not respond. The CM timeout and generates a Send SIP message response event containing a SIP Request timeout error code.

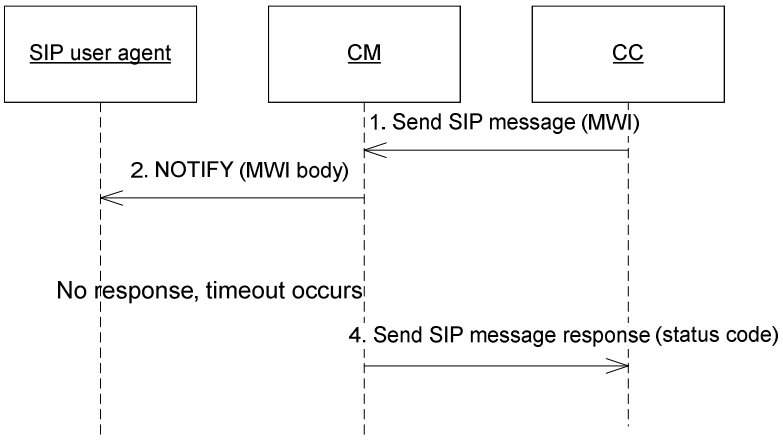


**Figure 28 Send SIP MWI message**

The SIP MWI (sent as a SIP NOTIFY request) times out. The timeout is handled as if a response with a 408 Request timeout was received. The 408 Request timeout is sent back in the Send SIP message response event.

1. **CC** request that a SIP MWI shall be sent.
2. **CM** sends a SIP NOTIFY request containing MWI parameters.
3. The SIP **UA** does not respond.
4. After a timeout, **CM** sends a Send SIP message response event with the status code for a Request timeout.

### 4.5.2 Session data variables

#### *4.5.2.1 Session creation*

The following session data variables are set by the Call Manager when starting a service instance due to a new inbound call:

- Session *Id*:
  When a new inbound call is created, a new session id is generated by Call Manager and set in the session when loading a new service instance. The type of this variable is *Id<ISession>* and is exported by [2].
- Session *MdcItems*:
  Contains call related information that can be used to filter the session logging further. The information contained in this item is telephone numbers for *calling*, *called* and *redirecting* parties if present. The type of this variable is *SessionMdcItems* and is exported by [2].
- Variable *sessioninitiator*:
  Contains the name of the protocol that initiated the session and is set to "*sip*". The type if this variable is String.

#### *4.5.2.2 Media negotiation*

CM reads the following parameters; *callmediatypesarray* and *selectedcallmediatypes,* and writes the following parameters; *selectedcallmediatypes* during media negotiation (see section 4.5.7).

### 4.5.3 Mapping of release cause to Network Status Code

When a call is rejected by the remote party prior to connect, the reason for the call reject is mapped to a Network Status Code (see [11]) using configuration. The call reject can be done with a SIP BYE, a SIP CANCEL or a SIP error response depending on the call direction.

The reason for the call reject is retrieved in the following order:

1. If the SIP message causing the reject contains a SIP Reason header field with Q.850 cause/location information, this information is used as release cause.
2. Otherwise, if the SIP message causing the reject is a SIP error response, the response code is used as release cause.

The default configuration is illustrated in the table below.

**Table 1 Default mapping of release cause to Network Status Code**

| Scenario | Network Status Code | SIP response codes | Q.850 cause values | Q.850 location values |
|---|---|---|---|---|
| User Busy | 603 | 486 | 17 | 1-15 or unknown |
| No Reply | 610 | 408, 480 | 18-19 | 0-15 or unknown |
| Not Reachable | 613 | 301, 403-404, 410, 484, 501-502, 603 | 1-9, 20-23, 25-31 | 0-15 or unknown |
| User Suppressed | 614 | 600 | 17 | 0 |

| | | | | |
|---|---|---|---|---|
| Congestion | 620 | 503 | 39-44, 46 | 0-15 or unknown |
| Default | 621 | The rest. | The rest. | 0-15 or unknown |

### 4.5.4 Final response handling for outbound calls

The table below lists how CM handles different final responses to an outbound call.

| **Response** | **Condition** | **Action** |
|---|---|---|
| 2xx | - | The call is connected. |
| 3xx | No previous redirection. | A new INVITE is sent to the first contact in the 3xx response. |
| 3xx | There is a pending redirection. | The call is considered rejected. |
| 4xx | - | The call is considered rejected. |
| 5xx | No previous redirection. | The call is considered rejected. |
| 5xx | There is a pending redirection and there is another contact to try from the previous 3xx response. | A new INVITE is sent to that contact. |
| 5xx | There is a pending redirection but there is no more contacts left from the previous 3xx response to try. | The call is considered rejected. |
| 6xx | - | The call is considered rejected. |

**Note**: **CM** only supports one level of redirection, i.e. the one from the **SSP**.

### 4.5.5 Compliance to SIP

How we comply with the SIP specification is described in detail in an IWD, see [5].

### 4.5.6 Loop Back Prevention

The **CC** is able to indicate if loop back prevention shall be done when creating a new outbound call. If loopback shall be prevented, Call Manager adds a SIP Diversion header (as described in [5]) to the outbound SIP INVITE.

### 4.5.7 Media Negotiation

All media negotiation is handled transparent to **CM** with regards to supported codecs. The codec negotiation is done based on the input from the **CC**, configured values (see section 4.5.13.3) and the peer **UA**. Thus, **CM** has no limitation regarding which codecs to support.

If **CC** has any requirements on the media types to use, it will give a possible list of media types in the session parameter *callmediatypesarray*.

**CM** will then perform media negotiation and choose one of the given media types and store it in the session parameter *selectedcallmediatypes*.

When **CM** is about to start media negotiation, is first checks the session parameter *selectedcallmediatypes* to see if one specific call media type already has been selected for the session. If that parameter is not set, it then checks the session parameter *callmediatypesarray* to see if **CC** has any requirements on the media types to use. If that parameter is not set **CC** has no requirements on the media types to use and **CM** will base the media negotiation on the configured media types only (see section 4.5.13.3).

When a remote SDP is received, the configured required outbound media types and the selected **CC** media type (if one exists) or the list of media types from **CC** (if one exists) will be matched to find an SDP intersection. The configured required outbound media types and the selected **CC** media type (if one exists) must be supported by the remote SDP. If there is no selected (see section 4.5.13.3) but there is a list of media types from **CC** at least one media type in that list must be supported by the remote SDP. The first match in the list will be set as the *selectedcallmediatypes* in the session.

First after the media negotiation has been performed, the call type is determined. This means that the call type is determined when the call has been connected for non-early media calls. For early media calls, the call type is determined when the early media has been established.

#### *4.5.7.1 Local SDP creation*

The local SDP is created using the inbound media types given in the configuration (see section 4.5.13.3). For voice calls only the audio media type is included, for video calls both the audio and video media types are included.

The type of call is determined as described below.

##### *4.5.7.1.1 Inbound call type*

An inbound SIP INVITE that contains an SDP offer is considered a video call if the media type video is contained in the SDP, otherwise it is considered a voice call.

If the inbound SIP INVITE lacks an SDP offer, Call Manager checks to see if there is a SIP header *Call-Info* present that describes the call type (as described in [5]). If the call type still cannot be determined, the configured default call type is used (see section 4.5.13.1).

##### *4.5.7.1.2 Outbound call type*

For an outbound call, the call type can be given by **CC** at call creation. If no call type is given, the configured default call type is used (see section 4.5.13.1).

#### *4.5.7.2 Media negotiation failure for inbound calls*

If the media negotiation fails, i.e. the required outbound media types were not supported by the remote SDP, a *Failed* event is generated and the call is rejected (for the early media scenario) or disconnected (for the non-early media scenario).

Note that even if the SDP offer is sent in the SIP 200 OK response and the SDP answer is received in the SIP ACK, the call is considered failed and a *Failed* event is generated. According to SIP the call actually has been connected although it is immediately disconnected, but the Call Manager considers the media negotiation as part of the call setup and therefore the call is considered failed rather than disconnected.

#### *4.5.7.3 Media negotiation failure for outbound calls*

If the media negotiation fails, i.e. the required outbound media types were not supported by the remote SDP, a Failed event is generated and the call is canceled (for the early media scenario) or acknowledged and disconnected (for the non-early media scenario).

### 4.5.8 Video Fast Update

Full picture video fast update as described in [12] is supported by **CM**.

When **CM** is requested (using the VideoFastUpdater interface, see section 4.2.5) a Video Fast Update request is sent over SIP as described in [12] – unless it is offered/accepted in SDP that Video Fast Update is to be done as RTCP feedback (see [14]), in that case Stream sends the request over RTCP.

### 4.5.9 Detection of Outbound Media Connection Already in Use

When an SDP is received from a peer **UA**, connection properties are given for the outbound media streams. **CM** checks if the ports suggested for the outbound media streams already are used by another call. If there is a collision, i.e. the host + port combination already is used by an existing call; this call is rejected.

### 4.5.10 Service loading

When an inbound call is received the **CM** loads a service to be used for that call. The following procedure is used when loading a service:

1. Try to load the service based on the called party number or PSI (Public Service Identity).
2. If the above fails, a default service is loaded instead.

### 4.5.11 Logging

#### *4.5.11.1 Session logging*

When a new thread is selected from a thread pool, the first thing that is done is that the active session (if any) is registered in the logger to enable session logging. There is an active session available for every function that is call related. Examples of functionality that are not call related is registration in an SSP or answering a SIP OPTIONS request.

#### *4.5.11.2 Info logging*

Informational logging is used by Call Manager in the following situations:

- When **CC** creates a new outbound call
- When creation of a new outbound call is rejected due to one of the following situations:
    - Mandatory parameters to create call is null, e.g. the event dispatcher, the session or the called party.
    - Internal error occurs when creating the call.
    - The current amount of calls exceeds the maximum threshold.
    - The administrative state of **CM** is not opened.
- When one of the following timers expires:
    - Call not accepted timer (see 4.5.13.1).
    - SIP INVITE expire timer.
    - Max call duration timer (see 4.2.2.2.3).
- To inform of which remote party (host and port) that has been selected for an outbound SIP INVITE.
- When **CC** requests a join or unjoin.
- When CC accepts or rejects a call or negotiates early media types for a call.
- When a SIP request, response or timeout event is received.
- When a SIP request is validated not ok and rejected as described in [5].
- How a SIP message received within a call is handled, e.g. if it is rejected.
- How a SIP message received out-of-dialog is handled, e.g. if it is rejected.
- When a SIP request or response is sent by **CM**.
- When **OMC** requests open, close or update threshold.
- The result of a requested open, close or update threshold, e.g. if registration or un-registration towards SSP:s is initiated.
- When **CM** is considered registered towards an SSP.
- When **CM** is considered un-registered from an SSP.
- When a new inbound call is created due to a session creating SIP INVITE.
- When creation of a new inbound call is rejected (with SIP INVITE) is rejected due to one of the following situations:
    - The administrative state of **CM** is not opened.
    - Internal error occurs when creating call.
- When a SIP OPTIONS request is rejected due to the administrative state of **CM** (i.e. the state is not opened).
- When an event is fired to an event dispatcher.

- When a service is loaded or cannot be loaded.

### 4.5.12 Protocol Encapsulation

The implementation of the protocols SIP and SDP is encapsulated. This is done to easily be able to exchange the current protocol stacks with others. For SIP this is done using the API JAIN SIP which could be kept even if the stack implementation was changed.

### 4.5.13 Configuration

When configuration has changed, the configuration is re-read as follows:

- For a call, the active configuration at call creation time is used and never re-read during the lifetime of the call.
- For a register/un-register procedure that is initiated, the active configuration is selected and not re-read until the next transaction.
- The active configuration is read directly at events not related to registration or call handling, e.g. when rejecting an inbound SIP request.
- The part of CM responsible for keeping track of all configured SSP:s and their status is updated as soon as the configuration is re-read.

The configuration need of **CM** is listed below.

#### *4.5.13.1 Call Related*

Call related configuration:

- Call not accepted timer:
  The time within which **CC** must have accepted an inbound call. If no accept is made, the call is considered abandoned.
- Default call type:
  The call type to use for a call when none is given in the inbound INVITE or by **CC** for an outbound call.

#### *4.5.13.2 Remote Party*

A remote party is mandatory in the configuration. The remote party could either be a phone or a list of **SSP**:s. A host and port is mandatory configuration for a phone or an **SSP**.

Optional configuration used if the remote party is an **SSP** list:

- Register backoff timer:
  The time to wait before trying again if a register request failed.
- Register before expiration time:
  The time before the registration will expire that a re-registration should be started.
- Registered name:
  The name with which **CM** registers itself in **SSP**.

#### *4.5.13.3 Media Related*

Media related configuration:

- Inbound audio media:
  The media type for recorded audio.
- Inbound video media:
  The media type for recorded video.
- Required outbound audio media:
  A list of audio media types that must be possible to play on an outbound stream.
- Required outbound video media:
  A list of video media types that must be possible to play on an outbound stream.

#### *4.5.13.4 Release Cause Mapping*

Mapping of release cause to Network Status Code (see [11]) can be configured. The configuration is optional and the default mapping is described in section 4.5.3.

#### *4.5.13.5 Diagnose Service*

The host and port to open for service diagnosing over SIP can be configured.

#### *4.5.13.6 Load Regulation*

Configuration related to load regulation during start up:

- Initial HighWaterMark
- Number of channels for ramp factor
- Number of increments for ramp factor

The ramp factor is then calculated by **CM** as Number of channels / Number of increments.

# 5 References

**[1]** FS - SIP Serving Proxy
1/FS-SWU0045 Uen

**[2]** FS - Execution Engine
3/FS-MAS0001 Uen

**[3]** FS - Stream
7/FS-MAS0001 Uen

**[4]** FS - Operate and Maintain Manager
17/FS-MAS0001 Uen

**[5]** IWD – SIP
7/IWD-1/HDB10102 Uen

**[6]** FS - Log Manager
2/FS-MAS0001 Uen

**[7]** FS - Configuration Manager
16/FS-MAS0001 Uen

**[8]** ITU-T Recommendation E.164
Assigned Country Codes

**[9]** Uniform Resource Identifiers (URI): Generic Syntax
RFC 2396

**[10]** FS - Media Object
13/FS-MAS0001 Uen

**[11]** UCD – Outdial Notification
11/155 53 – HDB 101 02 Uen

**[12]** Internet Draft: XML Schema for Media Control
draft-levin-mmusic-xml-media-control-03

**[13]** ITU-T Recommendation Q1980.1
http://www.itu.int/itudoc/itu-t/aap/sg11aap/history/q1980.1/index.html

**[14]** IWD – SIP Telephony
1/IWD-MAS0001 Uen