Author: Kenneth Selin, EERITOR
Title: FD - Media Access Server

FD – Media Access Server

Content

1 INTRODUCTION 2

2 GLOSSARY 2

3 FUNCTION STRUCTURE 4

3.1 OVERVIEW ____ 4
3.1.1 Conceptual areas ____ 5
3.1.2 Statistical interface ____ 6
3.2 COMPONENTS ____ 6
3.2.1 Application and Media Content package manager ____ 6
3.2.2 Call Manager ____ 7
3.2.3 Configuration Manager ____ 7
3.2.4 Event Dispatcher ____ 7
3.2.5 Execution Engine ____ 8
3.2.6 External Component Register ____ 8
3.2.7 Log Manager ____ 9
3.2.8 Mailbox ____ 9
3.2.9 Media Object ____ 9
3.2.10 Media Content Manager ____ 9
3.2.11 Media Content Package ____ 9
3.2.12 Media Translation Manager ____ 10
3.2.13 Message Sender ____ 10
3.2.14 Number Analyze ____ 10
3.2.15 Operate and Maintain Manager ____ 10
3.2.16 Profile Manager ____ 11
3.2.17 Provision Manager ____ 11
3.2.18 Service Request Manager ____ 11
3.2.19 Stream ____ 11
3.2.20 Traffic Event Sender ____ 11

4 FUNCTION BEHAVIOR 12

4.1 START-UP ____ 12
4.2 STOP ____ 12
4.3 ACCESS APPLICATION VIA VOICE CALL ____ 13
4.3.1 UC Initiate Call Using Media Content without Early Media ____ 13
4.3.2 UC Initiate Call using Media Content and Early Media ____ 15
4.3.3 UC Initiate Call without Media Content ____ 16
4.3.4 UC Deposit ____ 16
4.3.5 UC Play Prompt ____ 18
4.3.6 UC Transfer Call ____ 19

4.3.7 UC TTS 22
4.3.8 UC ASR 24
4.3.9 Support of Scalable User Directory (SUD) 25
4.4 PROVISIONING 26
4.4.1 Self provisioning 26
4.5 PROVIDE SUPERVISION INFORMATION 28
4.5.1 Provided service supervision 28
4.5.2 Service Enabler statistics 33
4.5.3 UC Monitor connection 34
4.6 OPERATION 36
4.6.1 Managing administrative state 36
4.6.2 Configuration handling 37
4.7 OUT DIAL NOTIFICATION SERVICE 38
4.7.1 initiateOutdialNotification method 39
4.8 SIP MESSAGE WAITING 40
5 THREAD MODEL INFORMATION 41
5.1 EXECUTION ENGINE 41
5.2 STREAM 41
5.3 CALL MANAGER 41
6 DESIGN RULES 42
6.1 CASE SENSITIVENESS 42
7 REFERENCES 43
8 TERMINOLOGY 43

History

Version	Date	Adjustments
A	2006-10-11	Initial revision (ERMTERI).
B	2006-12-04	Removed support of MSLOCATION in section 4.4.1. Updated sections Provided service supervision and Managing administrative state. (ERMKESE)
C	2007-06-13	Describes SIP Message Waiting. CallManager starts rejecting calls immediately at shutdown (ERMKESE).
D	2008-08-05	Added information about new events. Updated with support of SUD (EERITOR).

1 Introduction

This document describes overall design of MAS.

2 Glossary

Application	An Application provides one or several Services. An Application is executed by MAS. An Application contains the application logic

	in form of a set of VoiceXML, CCXML and ECMAScripts. The Application is ***not*** included in MAS.
Call	An established or connecting communication between a user and the system. A call has a call state.
Component	A component in MAS is design object that encapsulates some particular functionality. It exports interfaces to be used by other components in MAS and imports other interfaces exported by other components.
Connection	A connection is an established or connecting communication between a user and the system. A connection has a state.
Media Content	A set of one or several Media Objects.
Media Content Resource	A set of one or several Media Content with common characteristics such as language, voice or video, female or male etc. A Media Content Resource is bundled in a Media Content Package.
Media Object	Objects that abstract the following media formats: - text - binary data: sound, video (includes sound). Possibly other binary information may be used, e.g. a Word document.
Media Type	The characteristics of the media for example its encoding.
Services	Services in this document is the same services that are registered in MCR. I.e. the service is at least used by some other components in the system. It is the Application only that defines what services are offered through MAS.
Service Enabler	A Service Enabler is a part of MAS that implements a protocol server to be used to access a service.
Session	A Session has data related to a service request instance e.g. Outdial Notification.
SSP	SIP Serving Proxy
Token	An entity that contains an end user input. For example it can be a DTMF key pressed from a regular mobile terminal or a command recognized by an ASR engine.

3 Function Structure

3.1 Overview

The MAS is divided in a number of components. A component provides its functionality through one or several exported interfaces. It may also make use of functionality in other components by importing one or several interfaces. A component is not only a logical unit; it is also a candidate for a deliverable product or a module in the Foundation.

MAS have a configuration that defines which components are included. Among other things the configuration specifies which implementation shall be used for a certain component. So it is possible to replace an existing implementation of a component with another implementation. This is very useful for testing purposes for example it is possible to replace a real component with a stub implementation.

A "Provided Service" corresponds to a specific (configurable in Execution Engine) Application. When such a Service is requested a Session is created.

Figure 1 shows the MAS components and their dependencies (all of them are "use" dependencies).

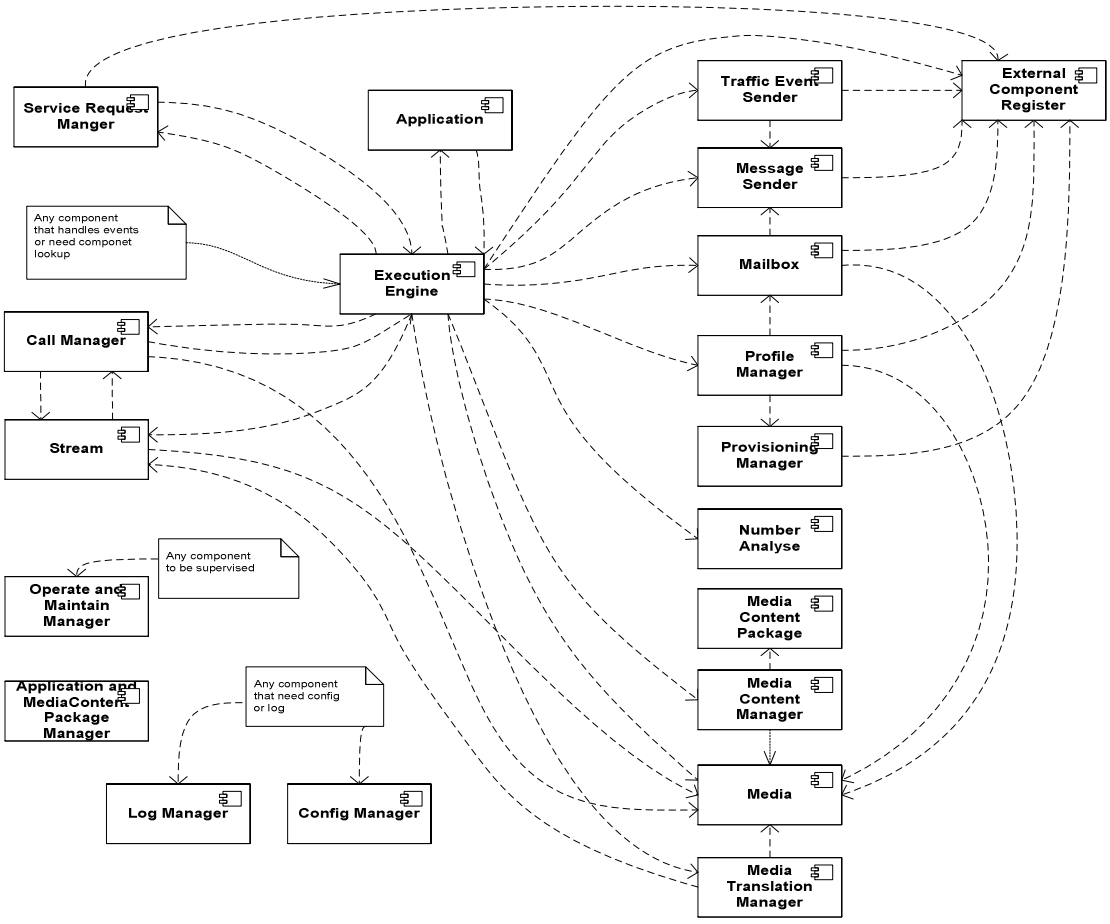


Figure 1 Components in MAS and their dependencies

A component can be a singleton or not. The user of the component is not aware that though, it just requests an instance using the components factory.

3.1.1 Conceptual areas

Even though components used in MAS are just components communicating with other components using its exported and imported interfaces, they can conceptually be divided in a few conceptual areas as shown in Figure 1

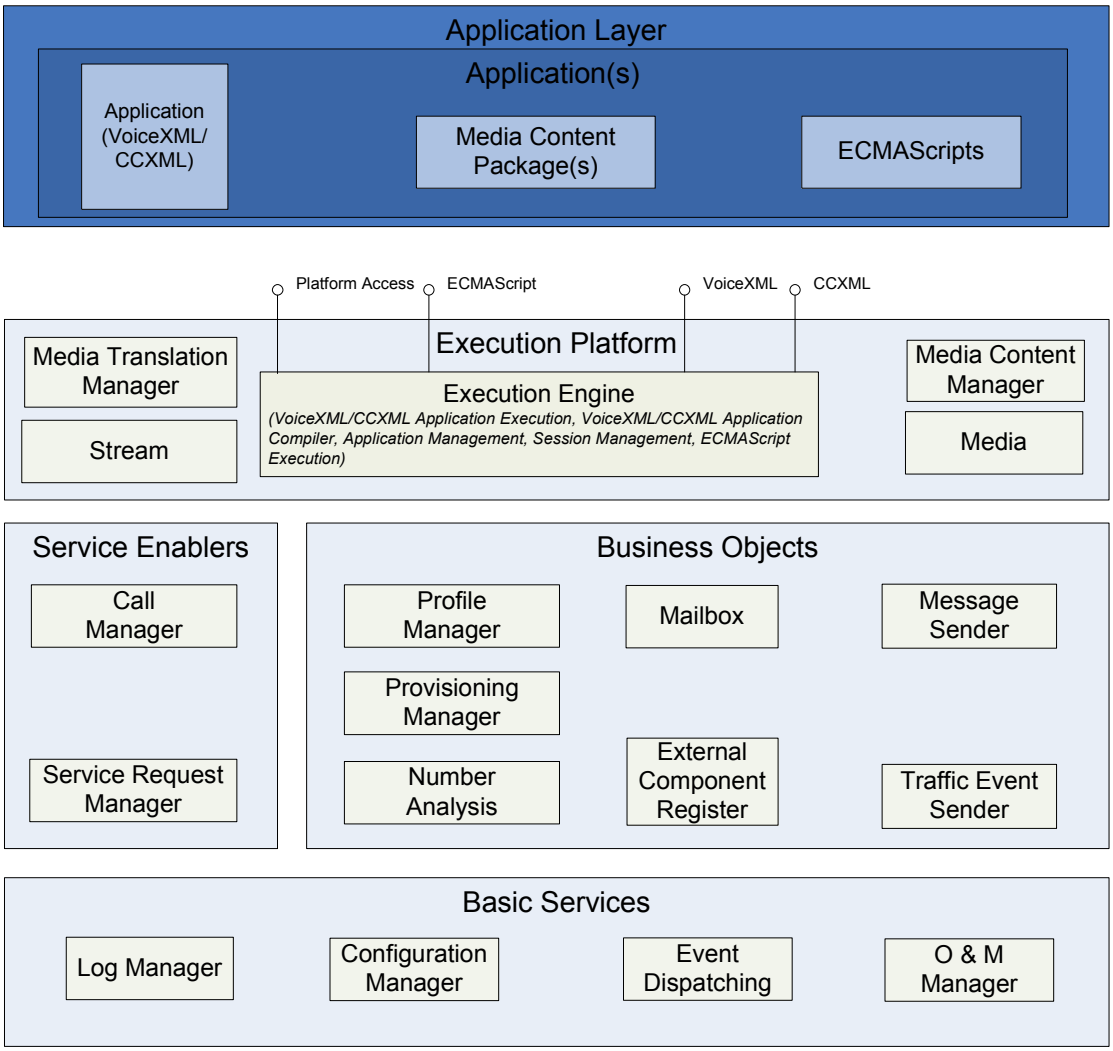


Figure 2 MAS Conceptual areas

3.1.1.1 Application Layer

The Application Layer is not part of MAS but included here because MAS expects that such a layer is created when an Application is installed on top of MAS.

The Application Layer consists of one Application, providing end-user functionality utilizing functionality provided by MAS. An application is built using

VoiceXML/CCXML, ECMAScripts. An Application can have one or several related Media Content Packages.

It is not possible to extend the functionality using compiled Java code embedded in the Application.

3.1.1.2 Execution platform

The components responsible for compilation and execution of VoiceXML/CCXML applications reside in the Execution Platform area. It also contains components providing media functionality. The Execution platform offers the following interfaces to the Application

- VoiceXML
- CCXML
- ECMA script
- Platform Access (enables the Application to access e.g. the Business objects)

3.1.1.3 Service Enablers

Service Enablers are used by remote clients to initiate the execution of a Service.

3.1.1.4 Business Objects

Business Objects contains functionality typically needed by a Voice Mail Application to e.g. access system databases.

3.1.1.5 Basic Services

Components residing in the Basic Services are used for common functionality needed by any component (including themselves).

3.1.2 Statistical interface

Through this interface a component can publish internal statistical information of interest. For example total number of sessions, number of failed requests etc. This will be the base for among other things In Service Performance.

3.2 Components

Below are all MAS components described in alphabetical order.

3.2.1 Application and Media Content package manager

The Application and Media Content package manager handles the interface for creation of Applications and Media Content Package. This is a manual interface which means that an administrator (a human being) use the interface to create an Application or a Media Content Package to be used in run-time.

A Media Content Package is input to Media Content Manager. An Application is input to the Execution Engine.

When the Application is created it contains information on which Services it provides. It also contains information on each Service how it is accessed (i.e. which protocol and on which port). This information is used by MAS to

1. register the service in MCR.
2. open up the ports for the protocol in question.

3.2.2 Call Manager

Call Manager is a Service Enabler for SIP requested services. Currently it can only handle one service per MAS. This is due to the fact that there is currently no defined algorithm that based on a SIP request to decide which service is requested.

It maintains a Call State. Call State is for example "alerted", "connected" etc.

It starts Application sessions related to the requested service using the Execution Engine.

The services are provided via an encapsulated SIP interface.

The component is responsible for collecting statistics related to each call.

3.2.3 Configuration Manager

The Configuration Manager provides information to other components that the configuration has changed using the Event Dispatcher.

The configuration is in XML format and the XML schema is included in the MAS delivery.

Each component that needs configuration specifies its own part of the configuration.

3.2.3.1 Fault handling

Configuration Manager will notify its clients when it fails to return a configuration parameter and there is no default value (provided by the client).

Configuration Manager validates the configuration by use of an XML schema and will create a log entry whenever it discovers errors in the configuration file.

If a client to Configuration Manager requests a parameter which has no default value it is responsible to create a log entry in case it doesn't receive the requested parameter.

3.2.4 Event Dispatcher

The Event Dispatcher is not a real component. Instead it is part of the Basic Services as a general function available for the components. Whenever a component informs about a particular event it will use the Event Dispatcher. Basically this means that it sends an event to the Event Dispatcher. Components interested in a particular event register themselves as a receiver of this event in the Event Dispatcher. Components can also un-register from the Event Dispatcher. The Event Dispatcher maintains an event queue for each session and one common for events not related to a session. This implies that each

component must register to the Event Dispatcher when a new session is created to be able to retrieve session related events.

3.2.5 Execution Engine

The Execution Engine is the central part of MAS and its main functional areas are described in the subsections below.

3.2.5.1 Startup etc.

When MAS starts, the Execution engine is responsible to, using configuration, instantiate all components needed (including which Application).

The Execution Engine is responsible starting applications and locates an Application based on the service name. The Execution Engine configuration contains information on which service this Application provides, which implies that it is the Execution Engine that knows which services are available. If the service doesn't exist in the Execution Engine, the service request fails.

3.2.5.2 Application Execution

The Execution Engine executes Applications (written in VoiceXML, CCXML, ECMAScript and Java) when requested by for example the Call Manager

3.2.5.3 Platform Access interface

This interface is used by the Application to access various inner parts of the MAS platform in a controlled way. Basically it offers access to Business objects e.g. Profile Manager, Mailbox etc. In order to use this interface the Application via ECMA script imports certain objects into the ECMA script environment and then makes use of these objects as if they would be a part of ECMA.

3.2.5.4 Session handling

When a Session starts the Execution Engine creates a session container that will be used for all session related information. It also exports an interface to this information to be used by other components.

When a Session terminates, the Execution Engine informs Platform Access that a Session is over. A component used by Platform Access and that needs to be called when a session is finished, for example to perform any necessary cleanup tasks, provides a **close** method in one, several or all of its provided interfaces (each component specifies if and how). If a component has such methods, Platform Access calls them before the component goes "out of scope".

3.2.5.5 Provide Connection monitoring

The Execution Engine provides information on each connection if requested.

3.2.6 External Component Register

The External Component Register is a client to MCR.

It allows other components to register their Services to MCR. Furthermore it provides an interface for other components to ask MCR for a single system component instance providing a certain Service i.e. Consumed Service. The

component may inform the External Component Register that a specific instance is not working and a new instance is needed. In case MCR is not reachable or has no resources for the service in question, configured fallback instances can be returned instead. It is also possible to override MCR and always use the fallback instance. The configuration has fallback/override entries per Service.

The algorithms on how to re-select a failing instances is handled within the Component External Register according to the external components recommendations.

3.2.7 Log Manager

The Log Manager provides the possibility for each component to log errors, warnings or information to a log file. It is also possible to trace activities on a particular end-user call. It is possible to configure different log levels for different components/classes. Log configuration is possible to change during run-time.

3.2.8 Mailbox

The Mailbox is a client to the Storage service.

It has an encapsulated interface to an IMAP server.

Mailbox handles emails on the IMAP server according to the ref. [1]

3.2.9 Media Object

The component contains Media Object types used by components that exchange media related information, for example recorded messages, pre-recorded prompts, text to speech conversion data etc.

The Media Object may contain the data itself or a reference to the data, for example a URL.

3.2.10 Media Content Manager

Provide access to Media Content Packages.

An Application may concurrently use several Media Content Packages for example to support multiple languages.

Conceptually the client first retrieves a Media Content Resource. From this resource it then requests a particular Media Content by specifying its identity and possibly a number of parameters (for example a date or a number) to be used for this entity. The Media Content Manager returns a list of Media Objects to the client.

3.2.11 Media Content Package

This component contains the Media Content required for a particular Application component.

A Media Content Package contains Media Content i.e. a set of Media Objects referenced with an identity. It has characteristics such as:
- Language
- Encoding (e.g. g711, text)
- Country

- Variant (e.g. type of voice)
- etc.

The Media Content Package is not part of MAS but included here because MAS expects that such a layer is created when an Application is installed on top of MAS.

3.2.12 Media Translation Manager

The Media Translation Manager is a client to the Text To Speech and Automatic Speech Recognition engines.

In case a Text to Speech engine hasn't any capability to do fast forward or rewind the Media Translation must be able to buffer received audio and on request do fast forward or rewind.

It has an encapsulated RTSP/MRCP interface.

3.2.13 Message Sender

The Message Sender is a client to the SMTPStorage service (send a message to an email receiver).

It has an encapsulated interface to SMTP.

It either uses the Component Register to get a system component that provides the Storage service or use a system component provided in the request. If sending fails, the message is discarded after a configurable number of retries.

3.2.14 Number Analyze

The Number Analyze handles number analyses functions.

This is basically a set of rules for converting telephone numbers to a specific format.

3.2.15 Operate and Maintain Manager

The Operate and Maintain Manager handles the SNMP interface for Supervision Information. This interface is implemented in a separate process to minimize impact on the traffic part of MAS. Hence there is an interface between the SNMP agent part and the traffic part.

It handles Load Regulation by interacting with Service Enablers.

It is responsible for doing self diagnostic by interrogating the provided services interfaces. This diagnostic executes in a process separate from the traffic part. For this release only SIP and XMP interfaces responsiveness is checked.

The interface between the Operate and Maintain Manager and the rest of the MAS is designed in a reusable fashion i.e. it doesn't assume a specific MIB for instance.

The Operate and Maintain manager also provides the following functionality:

- Installation
- Start/Stop
- Lock/Unlock/Shutdown

- View statistics and supervision information.
- Connection monitor
- Backup/Restore
- Process supervision

3.2.16 Profile Manager

The Profile Manager is a client to the UserRegister service.

The Profile Manager component has an encapsulated interface to databases used for example a directory server and can provide a Mailbox component.

3.2.17 Provision Manager

The Provision Manager is a client to the Provisioning service.

The Provision Manager handles the interface for creation and deletion of subscribers. Because the characteristics of this interface (in particular when creating subscribers) is that requests may take some time to perform they are possible to perform asynchronously.

The Provision Manager has an encapsulated interface to the CAI interface and is responsible to keep a pool of logged in connections open to the CAI server.

3.2.18 Service Request Manager

The Service Request Manager is a Service Enabler for XMP requested services and is a client to the External Subscriber Information service.

Service Request Manager has an encapsulated interface to XMP.

It starts Application sessions related to the requested service using the Execution Engine.

The component is responsible for collecting statistics related to each service request.

3.2.19 Stream

Stream handles the interface for streamed media for the service handled by the Call Manager.

Stream has an encapsulated interface to RTP.

It streams media to/from remote receivers/senders. A stream can be joined to another stream, which means that RTP packets received on one stream is sent out on the other stream. This join is one-way which means that e.g. the RTP for incoming packets on one stream is sent on the RTP for outgoing packets. If the two streams shall be fully joined, two join requests are needed.

Stream handles one or several RTP streams.

3.2.20 Traffic Event Sender

The Traffic Event Sender is a client to the EventReporting service.

It has an encapsulated interface to Radius-MA and Event Subscription Server (ESS)

It uses the Message Sender service to relay Traffic Event.

It is responsible for queuing and resend event in case the EventReporting service is down.

4 Function Behavior

This section describes a number of scenarios MAS supports. Most of these scenarios pertain to what a typical application would perform i.e. how it utilizes the functionality offered by MAS. There are some other, not service related functions described here as well.

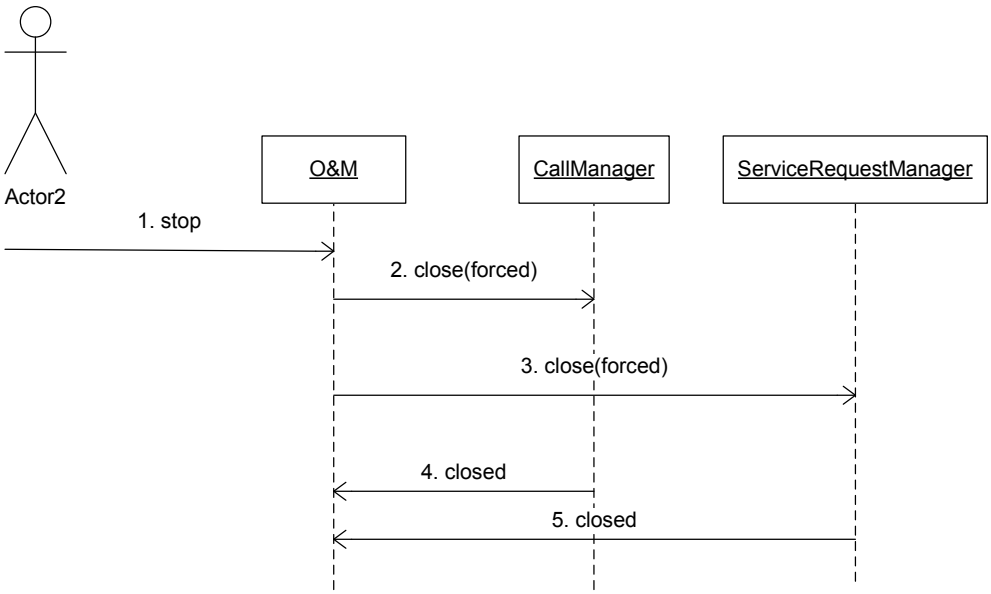
4.1 Start-up

When MAS starts, the Execution Engine based on configuration mentioned in section 3.1, loads and initiates all Components used.

4.2 Stop

When the operator stops the MAS, O&M closes all service enablers, and exits after the service enablers have reported that they are closed. If the service enablers have not called closed() within 30[1] seconds, MAS will exit anyway.

See also section 4.5.1.



1. The operator stops MAS.

[1] This time was chosen to be protocol-independent.

2. O&M invokes close(forced) on CallManager
3. O&M invokes close(forced) on ServiceRequestManager.
4. When CallManager is closed, it invokes closed() on O&M.
5. When ServiceRequestManager is closed, it invokes closed() on O&M.
6. At this point, O&M exits the MAS.

4.3 Access Application via Voice Call

This section describes how an Application can be accessed via a Voice Call. In all scenarios the following characteristics applies:

1. When a call has been answered, the Application can only change to a Media Content Resource that matches the existing call resource. For instance if the existing call is answered with a video resource, it cannot be later on be changed to a voice only resource.
2. If the incoming call is a video call and the Media Types selected by the application is voice only, the call will be rejected by MAS because the media negotiation will fail. This implies that if both video and voice calls shall be accepted, Media Content Packages for both voice and video Media Types must be installed.

4.3.1 UC Initiate Call Using Media Content without Early Media

This flow describes the Initiate call method using Media Content but without Early Media.

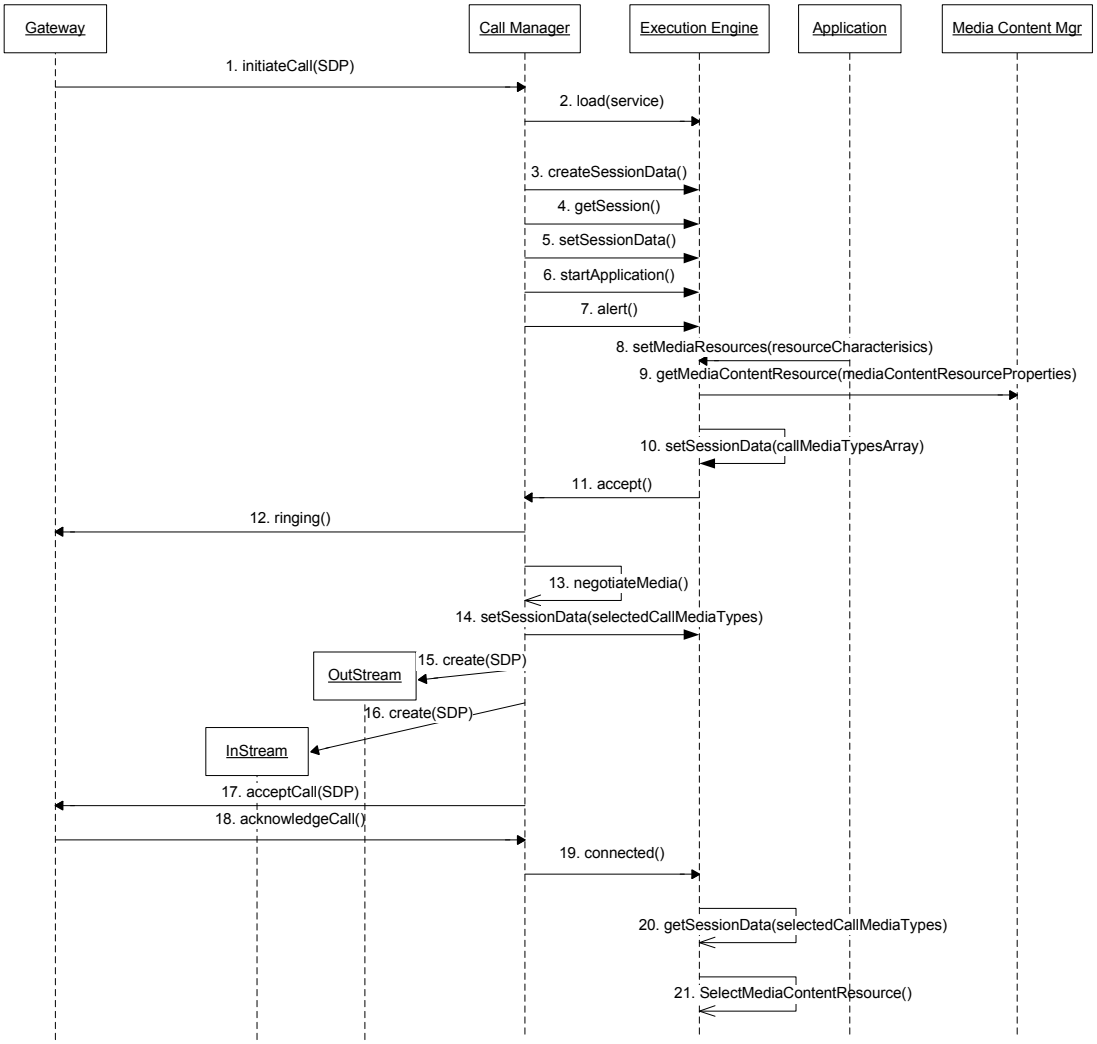


Figure 3 Initiate call with Media Content, w/o Early Media

1. The Gateway initiates a call to MAS providing the SDP information.
2. Call Manager loads the service it earlier has been initiated to provide.
3. Call Manager requests that Execution Engine creates the session data object.
4. Call Manager retrieves the session and ...
5. ... set call related information in the session data
6. Call Manager starts the Application that provides the service.
7. Call Manager sends the alert event to the Execution Engine to inform that there is an inbound call alerting.
8. When the Application has been notified that there is an alerting inbound call it, based on the call data set a Media Content Resource with characteristics indicating for instance language, variant etc.
9. Execution Engine request a list of Media Content Resources fulfilling the characteristics ...

10. ... and sets the call media types array in the session object. This array contains which media types the call manager later on shall use when negotiate the media with the Gateway.
11. Execution Engine, based on request from the Application accepts the call.
12. Call Manager informs the Gateway that MAS has received the request by sending the ringing event.
13. Call Manager negotiate media based on the SDP received earlier and the media found in the session object (call media types array).
14. When media types have been negotiated the selected call media type are set in the session object.
15. Call Manager creates the outbound and ...
16. ... inbound Stream with parameters corresponding to the selected call media types.
17. Call Manager forwards the accept to the Gateway which ...
18. ... acknowledge.
19. Call Manager sends the connected event to Execution Engine.
20. Execution Engine retrieves the selected call media types from the session object and ...
21. ... selects one of the Media Content Resources retrieved earlier as the one to use at future requests from the Application to retrieve media content.

4.3.2 UC Initiate Call using Media Content and Early Media

This flow describes the Initiate call method using Media Content. The flow is similar to the one described in section 4.3.1 13and shows how early media is handled. The main difference is that instead of using setMediaResources, the Application uses the setEarlyMediaResource which makes the Execution Engine to inform Call Manager that the Application has information to play pre-connect. Call Manager use this information to send a progressing event to the Gateway to inform that this UA will use outbound early media for this call.

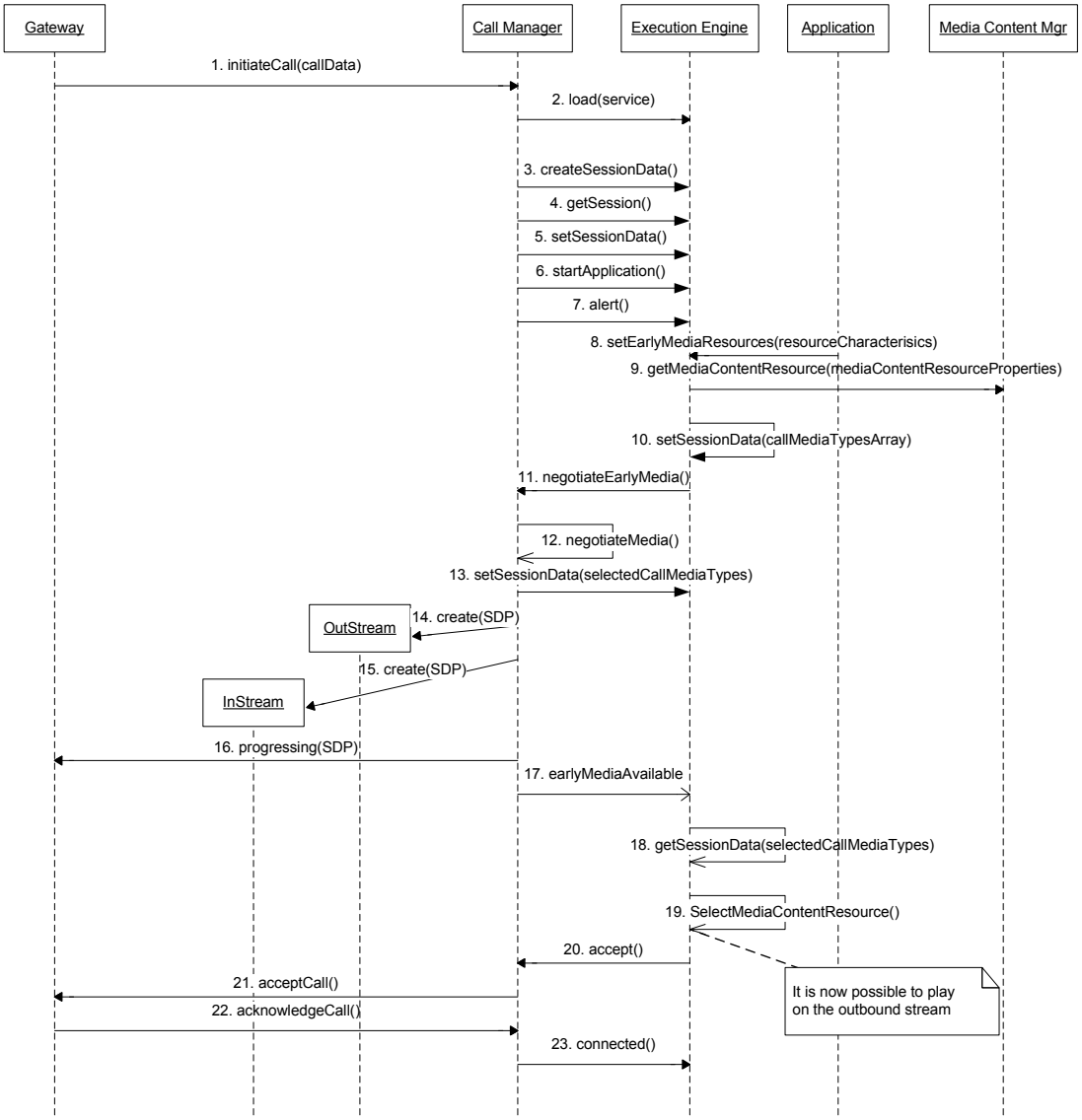


Figure 4 Initiate Call using Media Content and Early Media

4.3.3 UC Initiate Call without Media Content

This flow is very similar to what is shown in section 4.3.1. The difference is that the Application never issues the setMediaResource request and hence the mediaCallTypesArray will not be set in the session data. In this case, Call Manager uses the information in its configuration to determine which media types to negotiate the call with.

4.3.4 UC Deposit

This flow describes a Deposit use case.

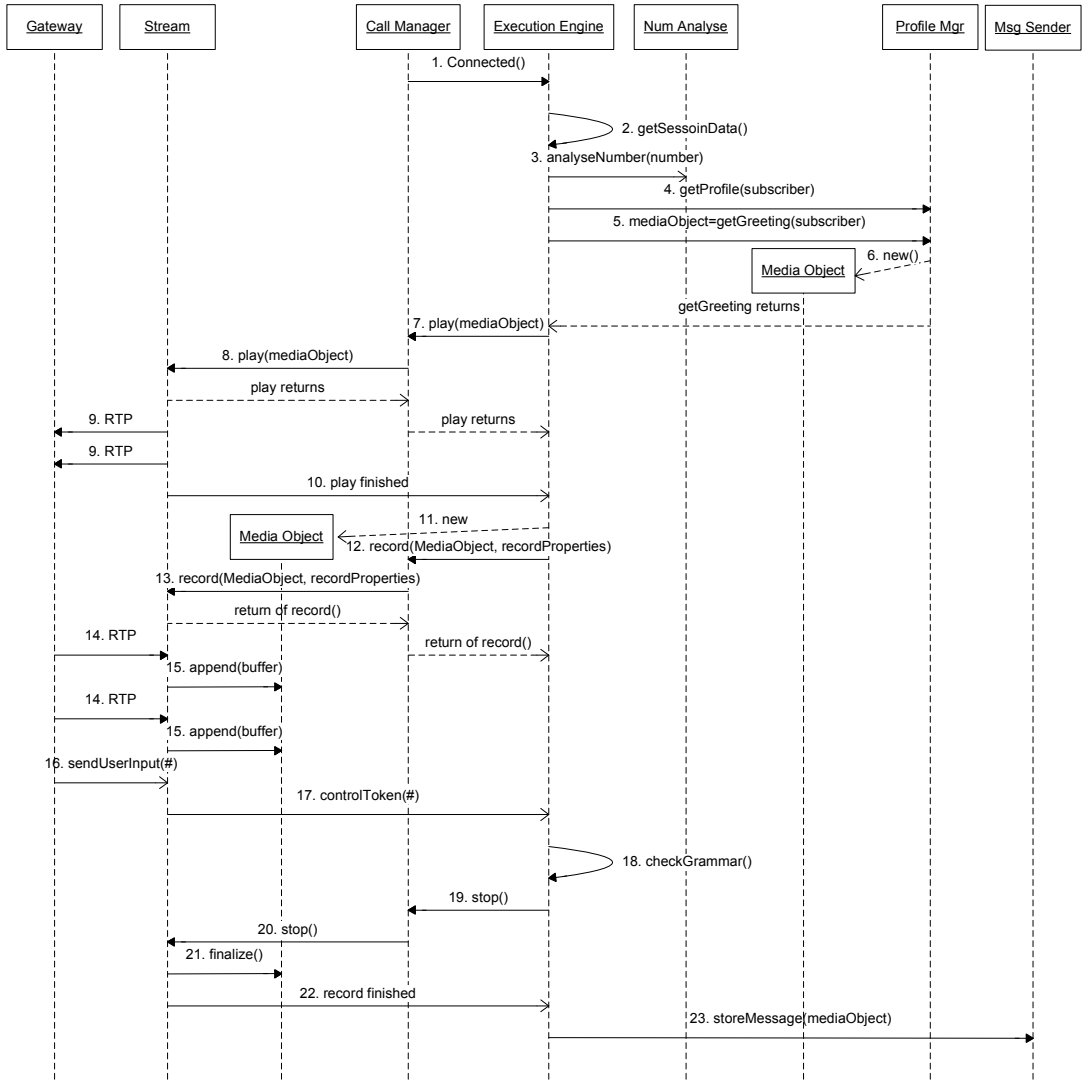


Figure 5 UC Deposit

Prerequisites: The call is in connected state.

1. The Call Manager tells the Application that a particular call got into connected state.
2. The Application retrieves call related information from the Execution Engine.
3. The Application does number analyses on telephone numbers stored in sessionData.

4. The Application retrieves the profile of a subscriber using the telephone number as the key.
5. The Application retrieves the greeting for the subscriber from the Profile Manager
6. Profile Manager creates a new Media Object that contains a reference to a file which then is returned.
7. The Execution Engine issues a play on Call Manager providing the greeting stored in a Media Object
8. Call Manager forwards the request to Stream and the play request returns.
9. Stream use the data in the Media Object and send those as RTP packets to the Gateway.
10. When the Media Object's all data has been sent on RTP the play finished event is sent to Execution Engine.
11. The Application wants to record and the Execution Engine creates a new Media Object and …
12. … issues a record on Call Manager providing the Media Object and the record properties. RecordProperties contains information when and how the recording should end for example maxtime, dtmf entered etc.
13. Call Manager forwards the request to Stream and record returns.
14. Stream receives RTP packets and ...
15. ... store those in a local buffer.
16. The end-user presses a DTMF key and Stream receives a userInput
17. Stream send the control token corresponding to the DTMF key pressed and …
18. … the Execution Engine check the grammar if the token matches and if so …
19. … the Execution Engine requests Call Manager to stop the recording.
20. Call Manager forwards the request to Stream
21. Stream finalizes the recording by adding the buffered media to the Media Object.
22. Stream sends the Record finished event to Execution Engine.
23. The Application creates a message and adds the Media Object into the message and store the message using a particular receiving host. The host used is retrieved from Profile (not shown in the sequence diagram though). If the storeMessage fails the Message Sender contain functionality to find alternative ways to store the message, for instance by asking Component Register for another host to use.

4.3.5 UC Play Prompt

This section describes the Play prompt UC using the Media Content Manager.

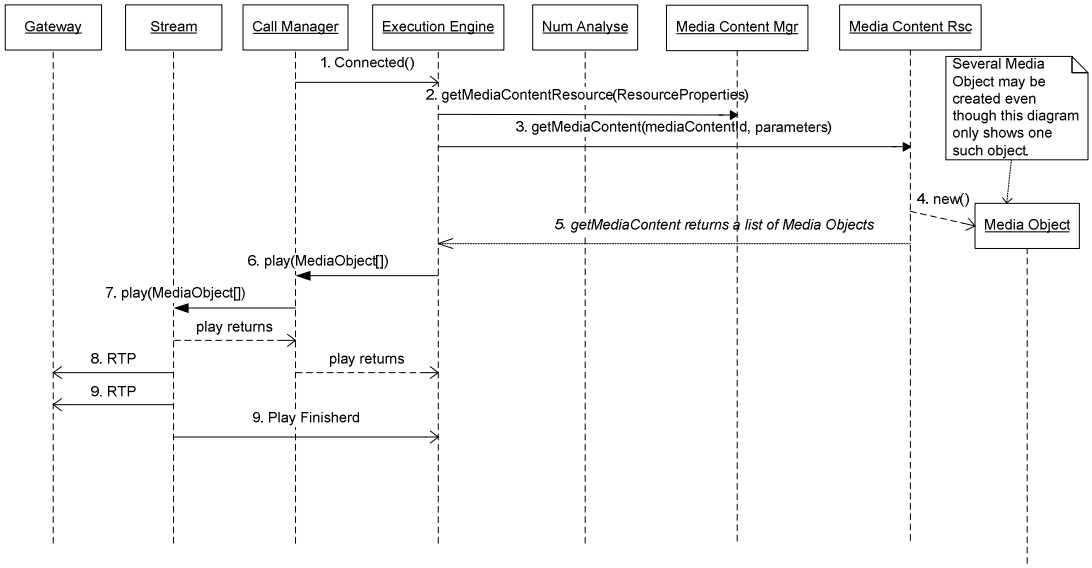


Figure 6 Play Prompt

1. Call Manager informs EE that the Call is connected.
2. The Execution Engine selects a Media Content Resource based on the Media Types to use for this call.
3. The Application requests a Media Content from the selected Media Content Resource.
4. A new Media Object is created for each part in the Media Content.
5. The list of Media Objects representing the Media Content is returned to the Execution Engine.
6. The Execution issues a play on the Call Manager providing the list of Media Object.
7. Call Manager forwards the request to Stream and play returns.
8. Stream sends the data from the Media Object on RTP.
9. When all the entire list of Media Objects has been played, the play finished event is sent to the Execution Engine.

4.3.6 UC Transfer Call

This section describes the Transfer Call UC.

4.3.6.1 Background

When transferring a Connection, normally there is initially one established Connection joined to a VoiceXML Dialog. The end-user at some point decides to make another call by issuing a transfer request. The platform then creates a new Connection and eventually joins these two Connections together.

Assume there are two Connections, C1 and C2 each of them have two Streams Sin and Sout. When the two Connections are joined together the involved Streams will be joined as follows:

- Sin1 to Sout2 (i.e. a new source to Sout2)

- Sin2 to Sout1 (i.e. a new source to Sout1)

The figure below shows how the involved Streams, Connections and Dialog are connected to each other before and after the join.

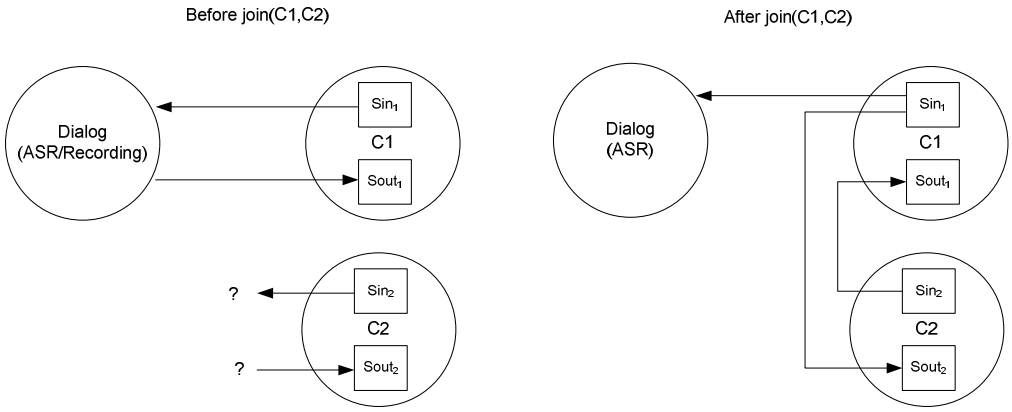


Figure 7 Example of Joining Connections

An important aspect is that in case Sin1 is joined to another receiver of its data, this join will persist even if Sin1 sends data to another receiver as well. On the other hand, if Sout2 earlier has been joined to receive data from another source, this join will be split up. That is a Stream can send to any number of others but only receive from one.

4.3.6.2 Characteristics of the transfer

When Connections are joined the Execution Engine keeps track on which Connections are joined together.

When two Connections are joined, Execution Engine, based on Application requests, performs all explicit and implicit un-join required (for example between a Dialog and a Connection).

Before a join can take place the involved Connections must have certain states. The list below shows what states are supported:

C1	C2
Connected	Connected
Connected	Progressing

Table 1 Supported Connection state for join

Call Manager is responsible to check if the Connections requested to be joined has the required state. If the state of the Connections is not supported, Call Manager will reject the request, otherwise Call Manager will join the involved Stream as described in section 4.3.6.3.

If Stream has an ongoing play request and gets a new request to join to another Stream, it will cancel the play and send the Play Finished event to the Execution Engine with cause "interrupted", before joining to the other Stream. A play request to a Stream already joined to another Stream will be rejected by Stream.

As mentioned above, join of two Connections may imply that an implicit unjoin between one of the Connections and a Dialog will be done. If the Application for some reason anyway tries to do a play on that Connection the request will fail.

The Application can specify that a particular new outbound call (createcall) shall be a voice call or a video call.

4.3.6.3 Transfer flow

Prerequisites: A call is connected and an end-user has requested to make an outbound call.

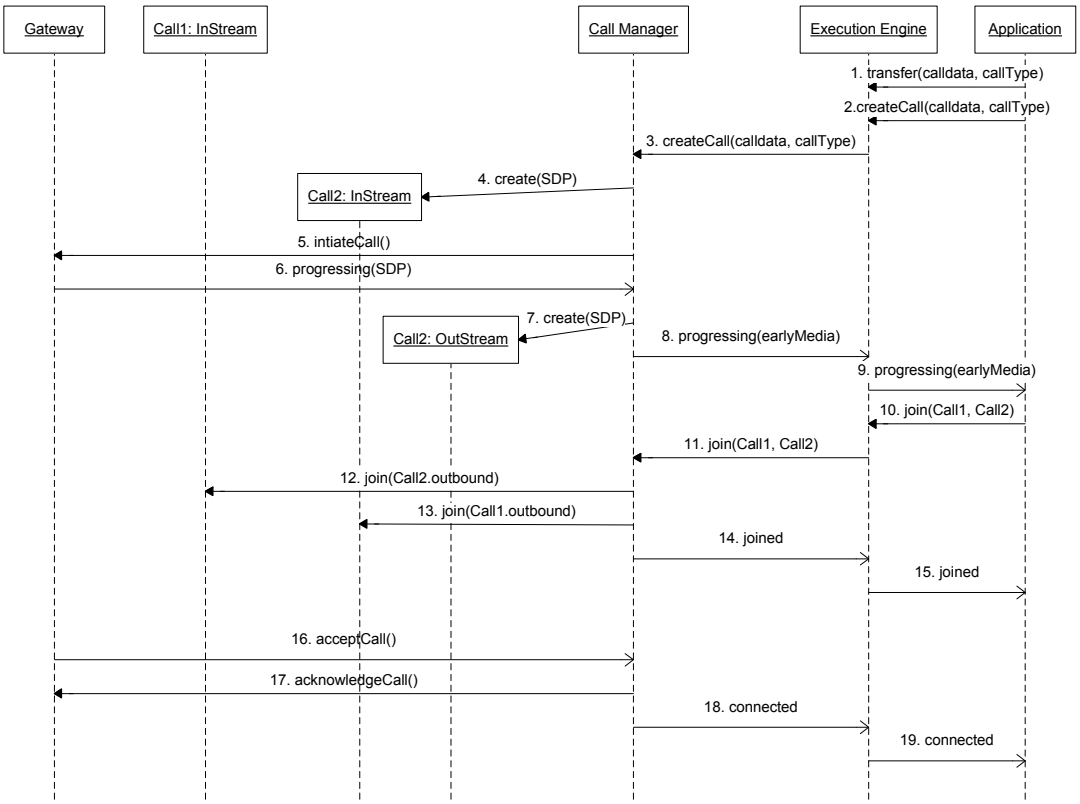


Figure 8 UC Transfer Call

In the figure above Call1 is the inbound call and Call2 is the outbound call.
The scenario below shows when there are early media available in the outbound call. If that's not the case the scenario will be slightly different i.e. the join will not take place until Call 2 is connected.

1. The Application issues a VoiceXML transfer call requests to the Execution Engine. Calldata contains the phone number for the new call and if the transfer shall be blind or bridged (currently only bridged transfer is supported).
2. The Application then issues a CCXML createcall request to the Execution Engine providing the same information.

3. Execution Engine Creates a new call request to Call Manager
4. Call Manager creates the inbound stream for the new call
5. Call Manager initiate a call towards the Gateway (the call request may initially be sent to a SSP and then redirected to a gateway but that is not illustrated here).
6. Gateway sends a progress which in this scenario indicates that there are inbound media available.
7. Based on the SDP information in the progress an inbound Stream is created for call 2.
8. Call Manager sends a progressing event to the Execution Engine indicating that there are early media available and ...
9. ... Execution Engine forwards this event to the Appliation.
10. The Application then decides to join the two calls in full duplex.
11. Execution Engine forwards the request to the Call Manager.
12. Call Manager joins Call1 inbound Stream with Call2 outbound Stream.
13. Call Manager joins Call2 inbound Stream with Call1 outbound Stream.
14. When the join is completed, Call Manager sends an event to Execution Engine.
15. Execution Engine forwards this event to the Application.
16. Eventually the Gateway accepts the outbound call.
17. Call Manager acknowledge the call
18. Call Manager send the connected event to Execution Engine
19. Execution Engine forwards the event to the Application.

4.3.7 UC TTS

This section describes the TTS UC i.e. how the translateTextToSpeech is utilized by an Application. Prerequisites a connected call and an end user has requested a play on a text.

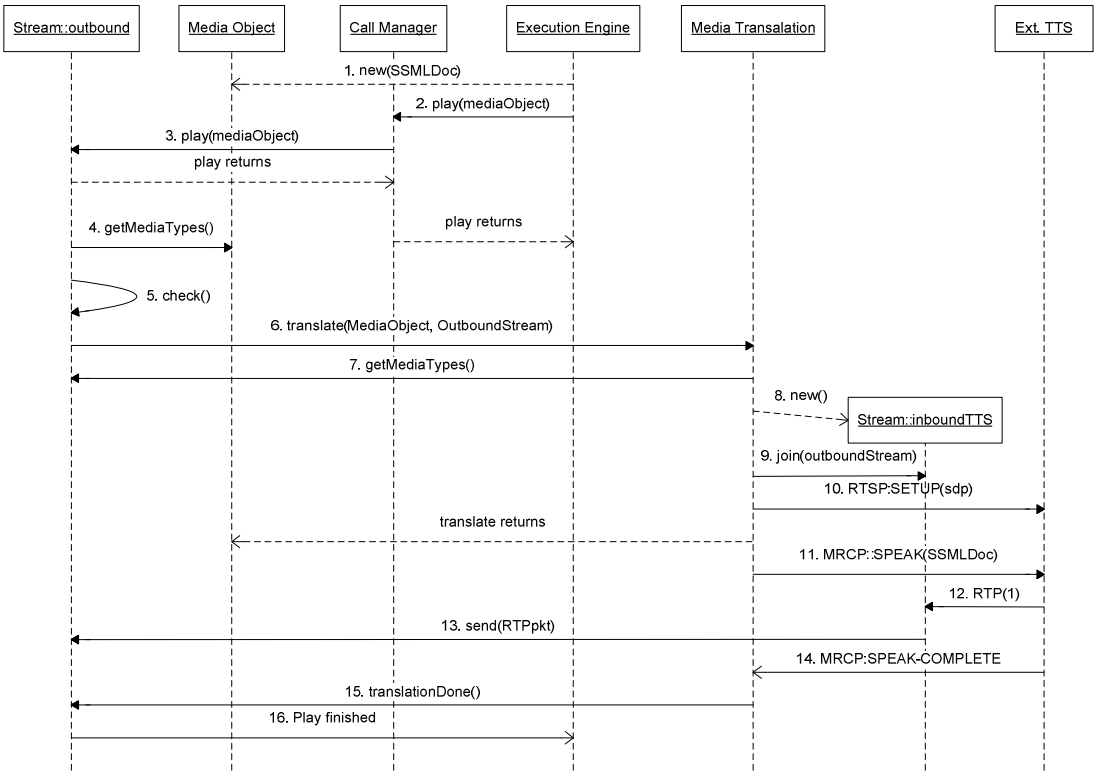


Figure 9 UC TTS

1. The Application creates a new Media Object that contains the text to be played (defined by the SSML Document).
2. The Execution Engine issues a play on Call Manager providing the Media Object.
3. Call Manager forwards the request to stream and the play request returns.
4. Stream retrieves the Media Types of the Media Object and …
5. … checks if this stream can handle those. Because it is text it concludes that it is not possible to play the text directly on the stream.
6. Stream then asks Media Translation Manager to translate the Media Object to a for Stream suitable format (e.g. g.711). SSML document contains information on how the text shall be converted (voice, emphases etc.)
7. Media Translation Manager retrieves the Media Types from the outbound Stream on which it shall play the Media Object.
8. The Media Translation then creates a new Stream object (that will have the same mediaproperties as the outbound stream) to be used for the translated text (inbound RTP) and …
9. … join this Stream with the outbound one.
10. Media Translation initiates the external TTS using RTSP and translate() returns.

11. Media Translation sends the SSML document to the external TTS. The translation can now start.
12. When the translated data (RTP from TTS) is received from the TTS engine …
13. … it is sent to the outbound Stream directly.
14. When the entire text has been translated, the TTS server sends the SPEAK COMPLETE message to Media Translation Manager.
15. Media Translation requests the Translation Done on Stream
16. Stream sends the play finished event to Execution Engine.

4.3.8 UC ASR

This section describes the ASR UC i.e. how the translateTextToSpeech is utilized by an Appliction.

Prerequisites; there is a connected call and an ASR grammar has recently been activated.

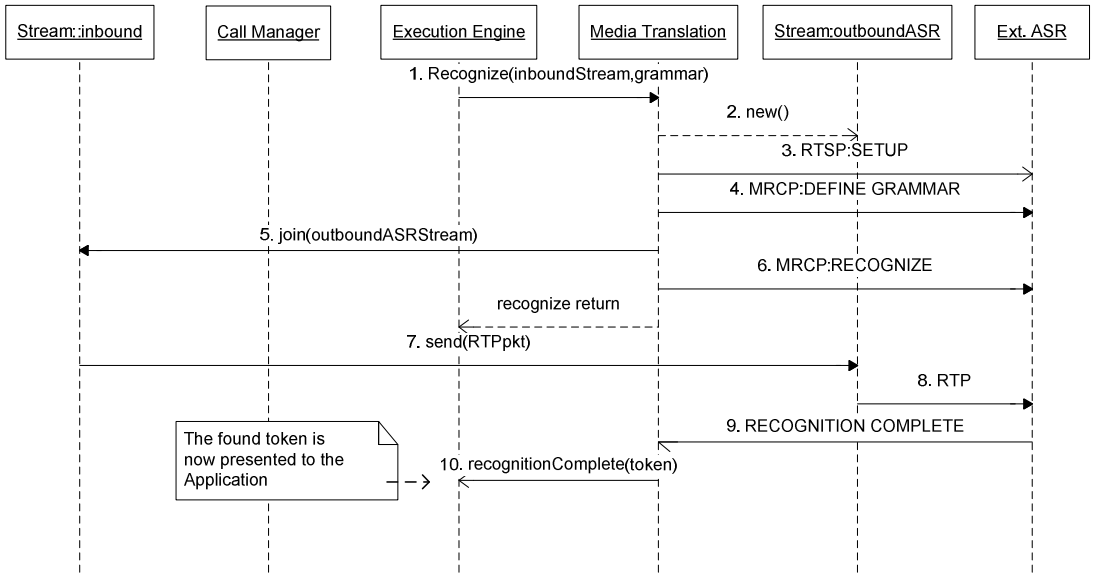


Figure 10 UC ASR

1. Execution Engine requests a recognition from Media Translation
2. Media Translation crates a new Stream object (that will have the same Media Types as the inbound stream) that will be used to send RTP data to the ASR engine.
3. The Media Translation initiates the ASR using RTS and …
4. … MCRP.
5. Media Translation joins the inbound Stream to the outbound ASR stream and …

6. … requests that the recognition starts using MRCP.
7. When RTP packets are received from the inbound stream they are sent to the outboundASRStream which then …
8. …forward them to the ASR engine.
9. When the ASR engine found a match it informs the Media Translation.
10. Media Translation informs the Execution Engine that a match has been found by sending the recognition complete event. A token contains the confidence as well.

4.3.9 Support of Scalable User Directory (SUD)

MAS support the SUD in two means:

1. Restrict searches in the user directory to one community only
2. Restrict searches in the user directory to one partition only.

The Application decides if a search shall be restricted or not and requests MAS to act accordingly by using methods in the Platform Access interface.

4.3.9.1 Community search restriction

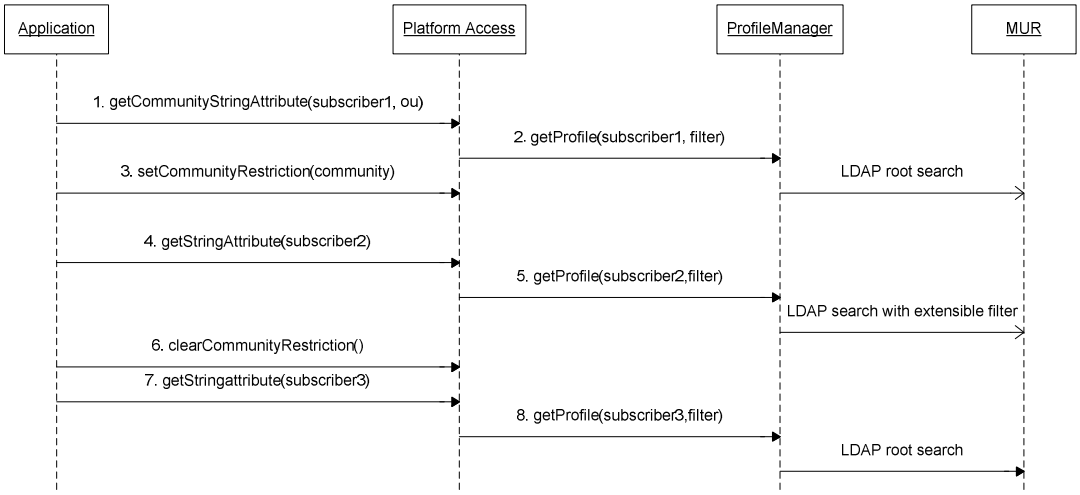


Figure 11 Community search

1. Application retrieves the community name for a particular subscriber
2. Since community restriction has not been set, MAS does a root search in MUR to locate the subscriber.
3. Application instructs MAS to restrict all consecutive searches by using the earlier retrieved community.
4. Application makes another search for a subscriber profile
5. Because community restriction now has been set, MAS searches in the specified community only.
6. The Application clears the community restriction.
7. Application makes a new search for a subscriber profile

8. Since the community restriction has been cleared, MAS performs a root search in MUR.

4.3.9.2 Partition search restriction

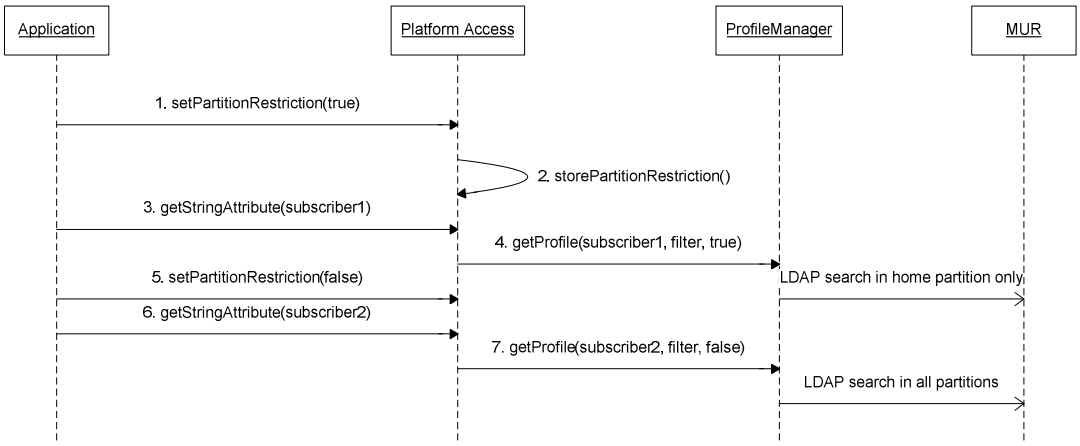


Figure 12 Partition restriction

1. Application instructs MAS to restrict searches in MUR to be only in the preferred partition.
2. MAS stores the restriction result for this session.
3. Application makes a search for a subscriber.
4. MAS makes a restricted search in MUR by providing the limitscope parameter to the request.
5. Application instructs MAS to do searches in all partitions again i.e. w/o providing the limitscope parameter.
6. Application makes another search for a subscriber.
7. MAS search in MUR without partition search restriction.

4.4 Provisioning

4.4.1 Self provisioning

This scenario shows how an Application manages subscriptions. When this scenario is started the Application has tried to identify the call as a deposit or retrieval call but failed and instead decided to create a subscription. Besides subscription attributes the Application also provides a provisioning administrator and optionally a name of a Class of Service.

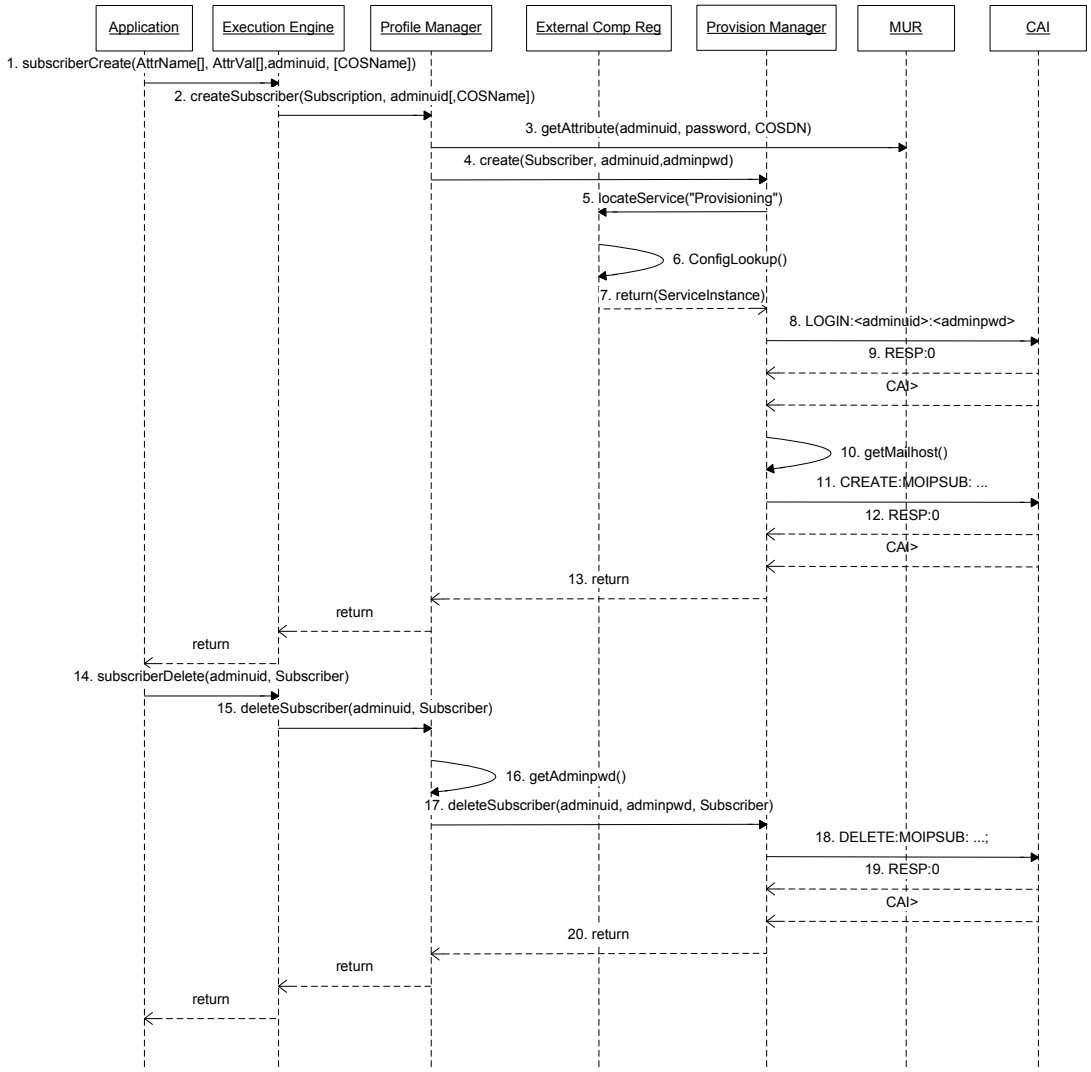


Figure 13 Provisioning

1. The Application via Platform Access provides a provisioning admin uid together with a list of subscriber attribute names and values (at least a telephone number is provided). Additionally other attributes can be provided according to the CAI interface. The Application can optionally provide a Class Of Service name this subscriber shall belong to.
2. The Platform Access in Execution Engine just passes this request to Profile Manager.
3. Profile Manager realizes that this is a provisioning request and retrieves the administrator's password from MUR and the COSDN for the provision administrator. If the Application provided a COS name, its corresponding COSDN is instead retrieved from either the cached COS:s or MUR using the provisioning administrator's Community).

4. Profile Manager, using the Attribute Mappings, adapts the request to the CAI attribute naming rules and sends the create request to Provision Manager.
5. In case there is no existing connection for this adminuid, Provisoning Manager search in External Component Register for the provisioning service
6. External Component Register try to lookup the service in MCR, but it will not be found (because this is currently not a Service). Instead the configuration is used. If several instances are found in the configuration, the MER algorithm is used to select one instance. The MCR lookup can be prohibited if the configuration specifies that MCR should be ignored for this service.
7. External Component Register returns the selected instance.
8. Provisioning Manager sends a LOGIN request to the CAI server providing its user identity and password.
9. The CAI server sends a response code and then the CAI prompt.
10. If the MAILHOST attribute in the Subscriber object is empty, Provisioning Manager check its configuration which mailhost this subscriber shall be provisioned into.
11. Provision Manager sends the CAI Create command with.
12. The CAI server sends the response code and the CAI prompt.
13. The Provisioning Manager returns the create request.
14. Later on the Application decides to delete a subscriber and sends the subscriberDelete request providing the admin uid and a list of subscriber attributes (only the telephone number is relevant for this request)
15. The Platform Access in Execution Engine just passes this request to the Profile Manager.
16. Because Profile Manager already have looked up this adminuid, it just retrieves the information from its cache before sending it to Provisioning Manager
17. Profile Manager sends the delete request to Provision Manager providing the provisioning uid and password together with the subscription.
18. Because there is already an established connection to CAI for this adminuid, Provision Manager can send the delete request immediately to CAI.
19. The CAI server sends a response code and the CAI prompt.
20. The Provisioning Manager returns the delete request.

4.5 Provide Supervision information

4.5.1 Provided service supervision

This section shows how the provided services offered by MAS are supervised.

MAS is in general Service agnostic which means that it only knows which Service Enablers it has and what protocols they support. When the Application is loaded the Execution Engine initiates the Service Enabler and creates a Provided Service entry in the Operate and Maintain manager. The Operate and Maintain Manager then updates the status of the provided service based on regular service requests.

The Provided Service Entry is published in a MIB and possible to retrieve by an SNMP manager. The Provided Service Entry contains the information specified in ref. [2]

4.5.1.1 Operational State

The operational state depends on the MAS only, and not for example on the accessibility of external components.

4.5.1.2 DiagnoseService

DiagnoseService is the process during which O&M polls a service to examine its health. DiagnoseService may report "up" or "down"[2].

The result of DiagnoseService is not depending on external components, registration in external components, or the MAS administrative state.

As soon as the MAS has started, DiagnoseService is regularly performed. The DiagnoseService shall always be possible to perform, regardless of MAS administrative and operational state.

4.5.1.3 Functions used by O&M on the Service Enablers

Open is a command telling the Service Enabler to open itself towards the "outer world".

This is what the service enabler shall do when invoked in open():

1. What this means is up to the Service Enabler, but typically it means that the Service Enabler shall register itself in an external component as being ready to handle traffic. Service Request manager **does not register** in MCR at this point, since doing that would be a violation of how MCR is to be used.
2. If the Service Enabler fails to register, it will continue trying until the registration succeeds.
3. When done, the Service Enabler calls opened() on O&M.

Close is a command telling the Service Enabler to close itself towards the "outer world".

This is what the service enabler shall do when invoked in close():

1. The Service Enabler shall unregister itself in an external component. A Service Enabler shall not unregister if unregistering is a violation of the contract between the service enabler and the external component. Service Request manager **does not unregister** from MCR at this point, since doing that would be a violation of how MCR is to be used
2. If the "forced" Boolean set supplied in the close() invocation is true, the Service Enabler shall immediately close all current sessions and then report to O&M that it has been closed.
3. If the "forced" Boolean set supplied in the close() invocation is false, the Service Enabler shall allow all current sessions to terminate normally.

[2] The value "impaired" defined on system level is not used within MAS.

4. When done, the Service Enabler calls closed() on O&M. After the Service Enabler has reported closed() to O&M, it shall reject all attempts to start a service.

4.5.1.4 Sequence Diagram

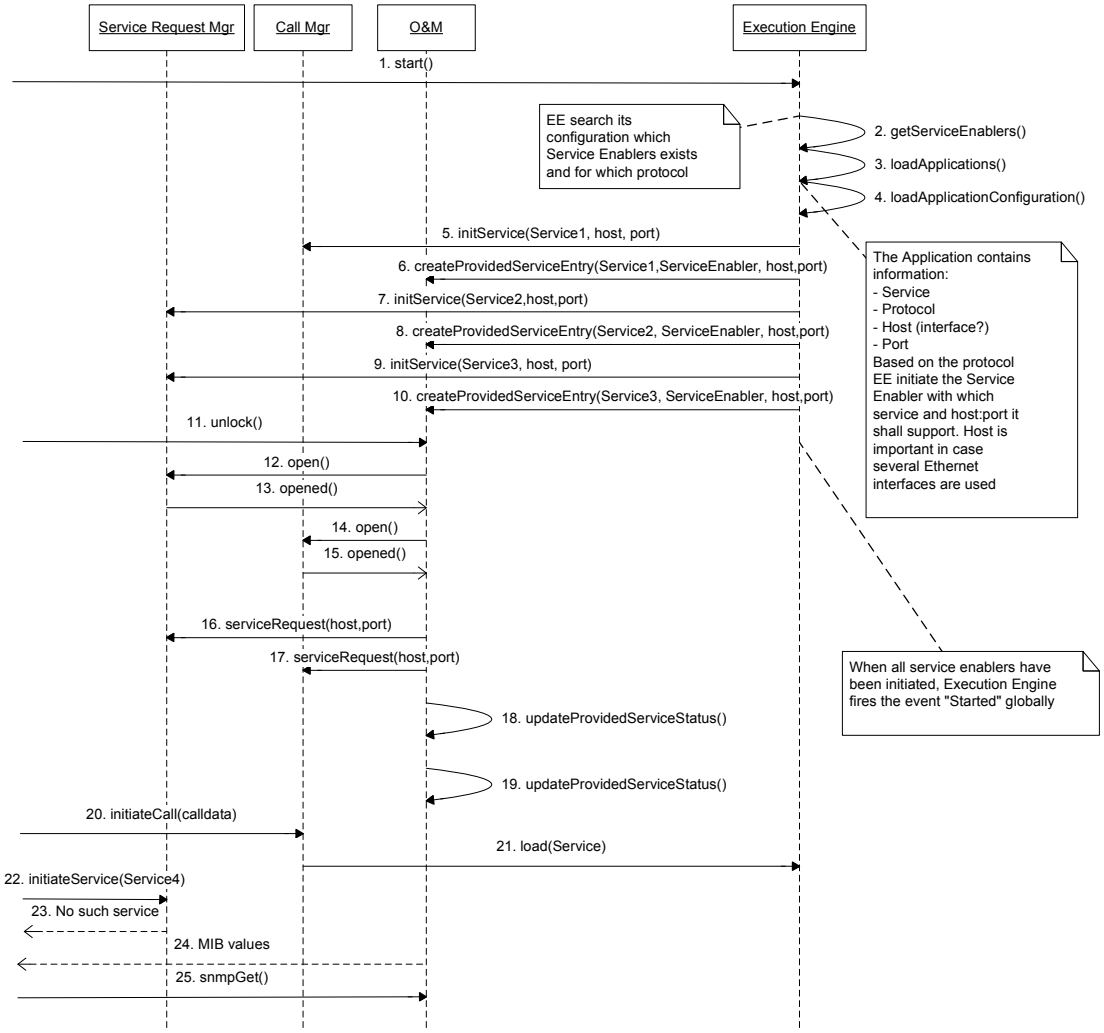


Figure 14 Provided service supervision

1. The operator starts the MAS.
2. Execution Engine searches the configuration to find out which Service Enablers exists and which protocol they support.
3. Execution Engine loads the installed Applications
4. Execution Engine loads the Application configuration. This contains which services the Application provides and for each service, protocol, host and port used.
5. Based on the information found from the Application and Service Enabler configuration, Execution Engine initiates the Services in the Call Manager (as a Service Enabler for SIP). Because of the SIP nature, Call Manager

can only enable one (1) Service. If Execution Engine tries to initiate a second service Call Manager will reject this request. When the initiateService method returns it can be assumed that the Service Enabler is functional (from its point of view).

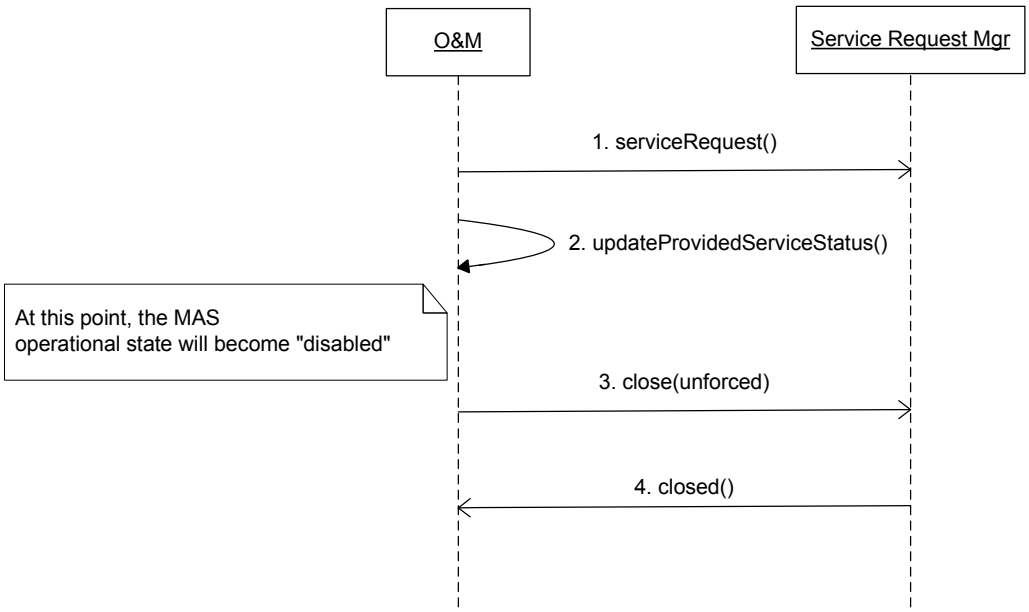
6. Execution Engine creates one Provided Service in O&M because:
   - O&M must be aware on which services it shall do self monitoring on.
   - O&M must publish the Provided Service status in the MIB. And the same
   …
7. … for Service Request Manager ...
8. ... all ...
9. ... provided ...
10. ... services.
11. The operator eventually unlocks MAS
12. O&M opens Call Manager.
13. Call Manager reports back when it is opened.
14. O&M opens the Service Request Manager.
15. Service Request Manager reports back when it is opened.
16. O&M monitors the Service Request Manager (because it is a Service Enabler) by issuing a service request.
17. O&M monitors the Call Manager by issuing a service request.
18. Based on the service request result O&M updates all the Provided Service entry status enabled by the Call Manager (only if it indicates a worse status than it already has)
19. And the same for Service Request Manager.
20. A remote SIP client initiates a SIP call to Call Manager.
21. Call Manager looks up which service it has initiated and request that service to be loaded by the Execution engine.
22. A remote XMP client performs a service request to Service Request manager, requesting the service "Service4", which has not been initiated.
23. Service Request Manager rejects[3] this request because this service is not initiated.
24. A Network Manager requests Provided Service status by issuing an SNMP get to O&M.
25. O&M returns the status of the Provided Service.

4.5.1.5 A Service Enabler reports "down"

When a Service Enabler reports that it is down, O&M will tell this Service Enabler to close(unforced), and the MAS operational state is set to disabled. O&M allows the other Service Enablers to continue as before.

In the example below, the Service Request Manager reports "down".

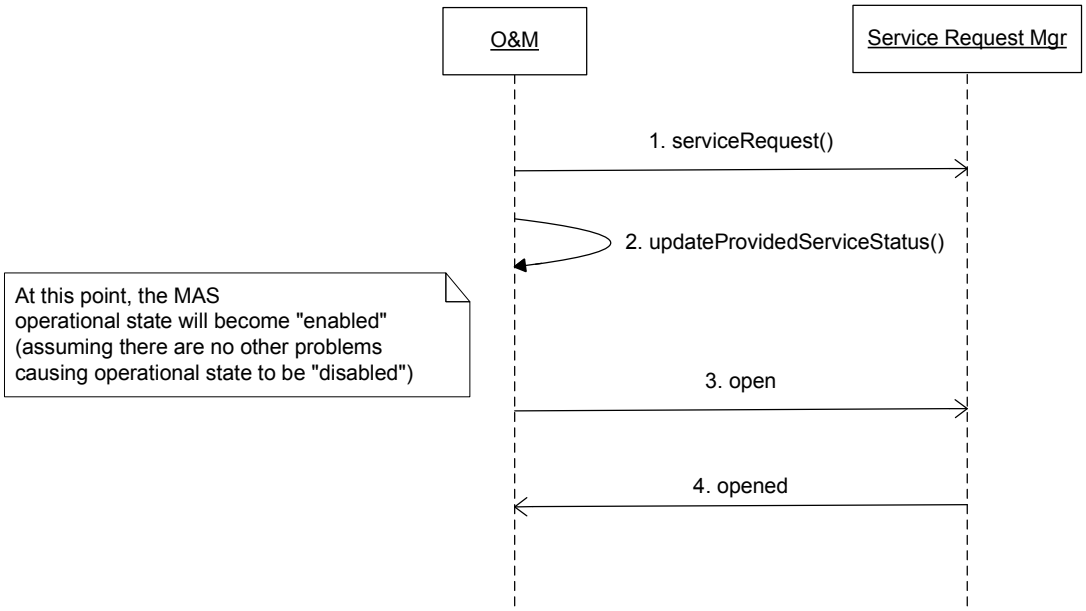
[3] This is true if the Service Enabler has several services for the same port. If not the service request will fail completely (i.e. it will not be possible to even establish a session to MAS).



1. O&M monitors the Service Request Manager by issuing a service request.
2. In this case, the result is "down", and O&M updates the Provided Service entry status accordingly. O&M sets MAS operational state to "disabled".
3. O&M tells Service Request Manager to close(unforced).
4. When all sessions in Service Request Manager have terminated normally, Service Request Manager invokes closed() on O&M.

4.5.1.6 A Service Enabler reports "up" after having reported "down"

If a Service Enabler reports "up" on DiagnoseService request after have earlier reported "down", O&M will invoke open(), if and only if MAS administrative state is unlocked.



1. O&M monitors the Service Request Manager by issuing a service request.
2. In this case, the result is “up”, and O&M updates the Provided Service entry status accordingly. O&M sets MAS operational state to “enabled”.
3. O&M tells Service Request Manager to open().
4. The Service Enabler reports back when it is opened.

4.5.2 Service Enabler statistics

This section shows how the Service Enablers within MAS provides statistical information.

It is assumed that the O&M already has a Provided Service entry when this scenario happens.

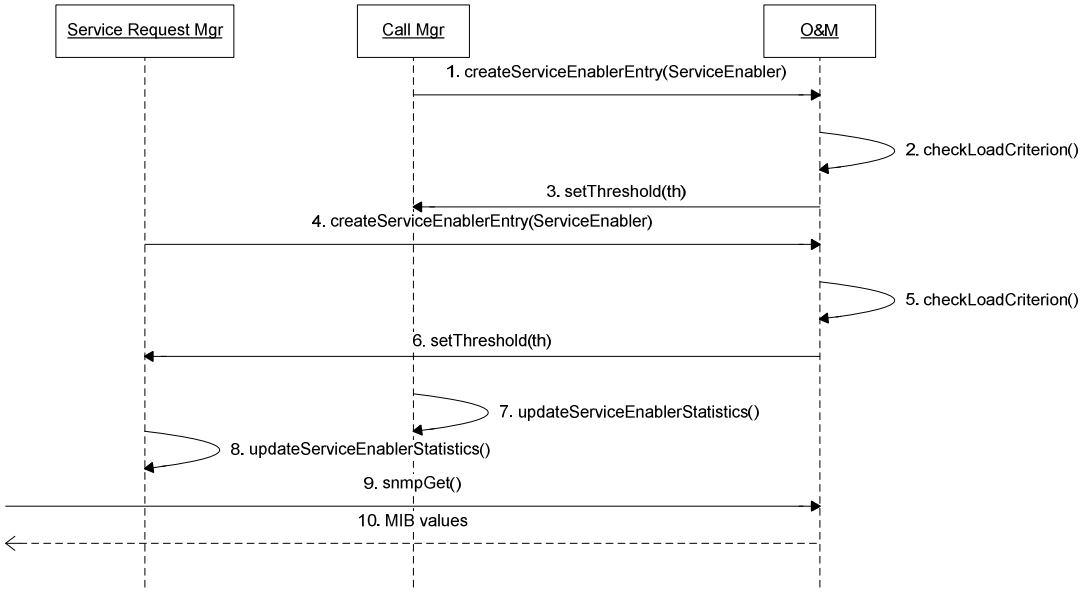


Figure 15 Service Enabler statistics

1. Call Manager (as a Service Enabler) creates a Service Enabler statistics entry in O&M
2. O&M checks the load criterion (i.e. the configured value) for Call Manager and ...
3. ... sets the threshold for the amount of calls to accept.
4. Service Request Manager (as a Service Enabler) creates a Service Enabler statistics entry in O&M
5. O&M checks the load criterion (i.e. the configured value) for Call Manager and ...
6. ... sets the threshold for the amount of calls to accept.
7. Call Manager continuously updates the Service Enabler statistics.
8. Service Request Manager continuously updates the Service Enabler statistics.
9. The SNMP manager requests the Service Enabler statistics and …
10. … O&M returns the values.

4.5.3 UC Monitor connection

This section shows how connection related information is monitored. The following information is shown for each connection (Session below corresponds to a CCXML session):

SessionId	The identity of the session using this connection. This information can be used to see which connections are used for a particular session.
Service	The Service using this connection.
Session initiator	One of XMP client id, SessionId (in case this the entry is created from an existing session) or Telephony

Connection type	Video or voice connection type
Direction	Inbound or outbound
Connection State	The CCXML connection state
Inbound activity	If record of media is active or not
Outbound activity	If play of media is active or not
ANI	The telephone number of the calling party
DNIS	The telephone number of the called party
RDNIS	The telephone number of the redirected called party
Connection	Far end IP-address and port used for the connection, one for each protocol involved.

Example on how the display might look like:

SId	Svc	SessInit	Tp	Sts	Dir	In	Out	A	B	C	Connection
109	Notif	ntf1@host.com	Vi	C	O	I	P	0601234	060133	-	RTP 1.3.4.5:5000; SIP 1.3.4.9:5070
110	Pager	ntf2@host.com	Vo	A	O	I	I	0601234	060432	-	RTP 1.3.4.5:5010; SIP 1.3.4.10:5070
111	VoiceMail	Telephony	Vo	C	I	R	I	0601234	060133	0601234	RTP 1.3.5.6:5000; SIP 1.5.6.7:5070
-	VoiceMail	SId:111	Vo	C	I	R	I	0601234	060455	-	RTP 1.4.3.5:5000; SIP 1.4.5.23:5070

The monitor is supposed to be used by an operator (a human being).
When a row in the presented data is removed the other rows is not affected i.e. the remaining rows shall not be scrolled up. A new session is added at the first available free row. The reason for this is that an operator is watching a particular row will not be affected if other connections is removed or added.

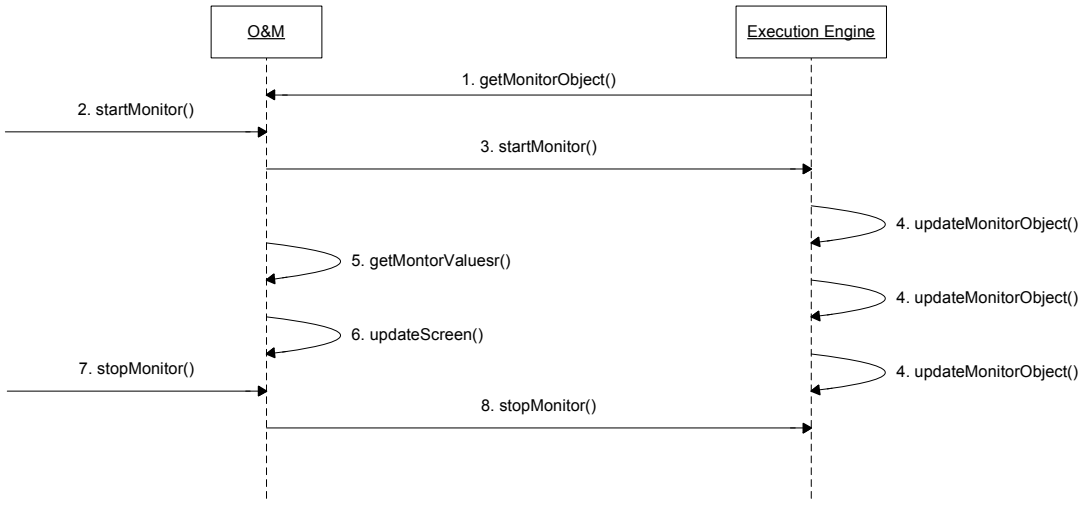


Figure 16 Connection monitoring

1. When MAS starts, Execution Engine requests a Monitor Object from O&M
2. Eventually the operator starts the monitor.
3. O&M send a start request of a monitor in the Execution Engine. The reason why the startMonitor request is used is that it allows the Execution Engine to collect the connection information only when required. If this is not a problem the implementation might be empty.

4. Execution Engine continuously update the Monitor Object
5. O&M, which have a reference to the same Monitor Object, retrieves connection information and ...
6. … and presents this information to the operator.
7. The operator stops the monitor.
8. O&M send a stop request to the Execution Engine.

4.6 Operation

This section describes how MAS is operated.

4.6.1 Managing administrative state

This section describes how the administrative state is used within MAS. Basically it is the Service Enablers that is controlled with this state.

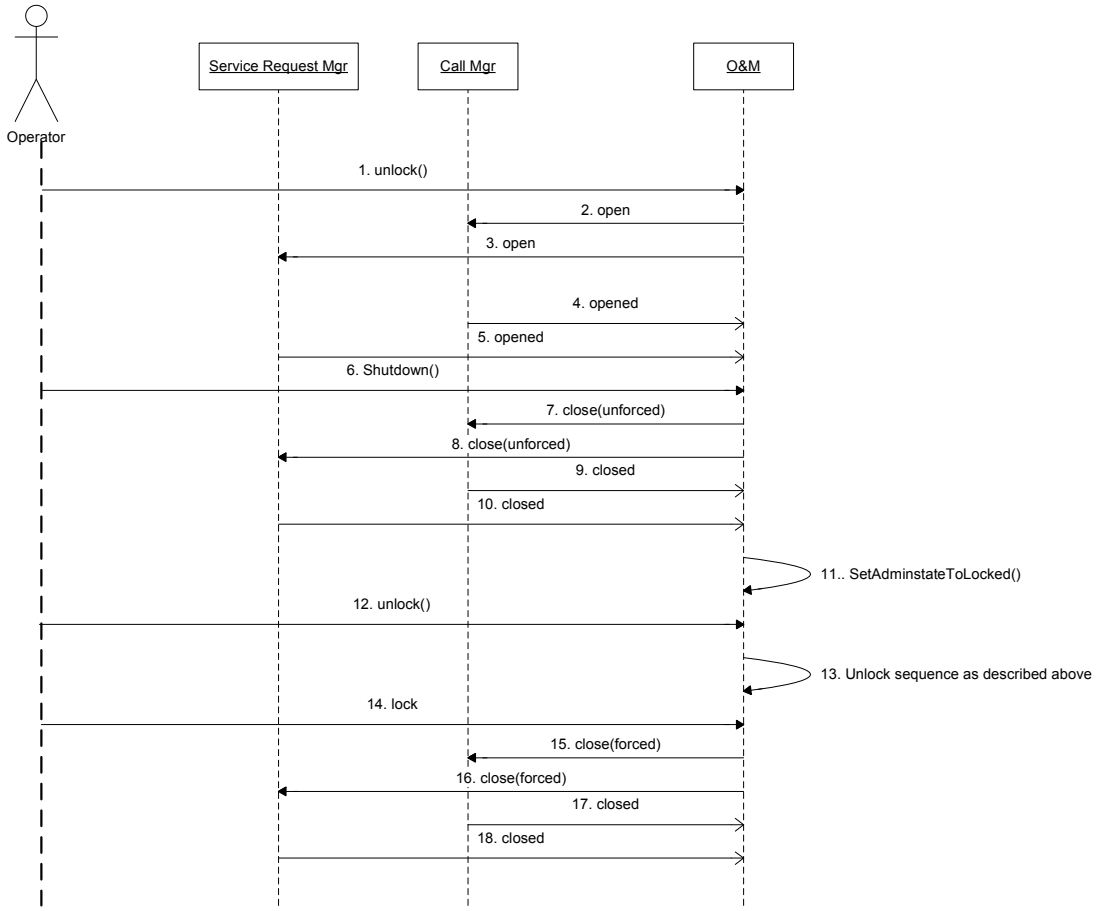


Figure 17 Managing Administrative state

1. The operator issues an unlock request. O&M immediately changes value of administrative state to "unlocked".
2. O&M opens the Call Manager. Call Manager now start accepts traffic according to its threshold.

3. O&M opens the Service Request Manager. Service Request Manager now start accepts traffic according to its threshold.
4. The Call Manager reports that it is now opened.
5. The Service Request Manager reports that it is now opened.
6. The operator issues a shutdown request. O&M immediately changes value of administrative state to "shutdown".
7. O&M closes the Call Manager, which then stops accepting new traffic. The "unforced" flag supplied instructs Call Manager to let all current sessions terminate normally.
8. O&M closes the Service Request Manager, which then stops accepting new traffic. The "unforced" flag supplied instructs Service Request Manager to let all current sessions terminate normally.
9. When all sessions have terminated, Call Manager reports that it is closed.
10. When all sessions have terminated, Service Request Manager reports that it is closed.
11. When all Service Enabler have reported that they are closed, O&M sets administrative state to locked.
12. The operator issues an unlock request
13. O&M performs the same sequence as described in 1-5.
14. The operator issues a lock request. O&M immediately changes value of administrative state to "locked".
15. O&M closes the Call Manager, which then immediately terminates all current calls.
16. O&M closes the Service Request Manager, which then immediately terminates all current service requests by sending an XMP response to its clients with a corresponding error code.
17. When all sessions have terminated, Call Manager reports that it is closed.
18. When all sessions have terminated, Service Request Manager reports that it is closed.

4.6.2 Configuration handling

4.6.2.1 Dynamic load of configuration

This section describes the how an operator can initiate a dynamic reload of the MAS configuration.

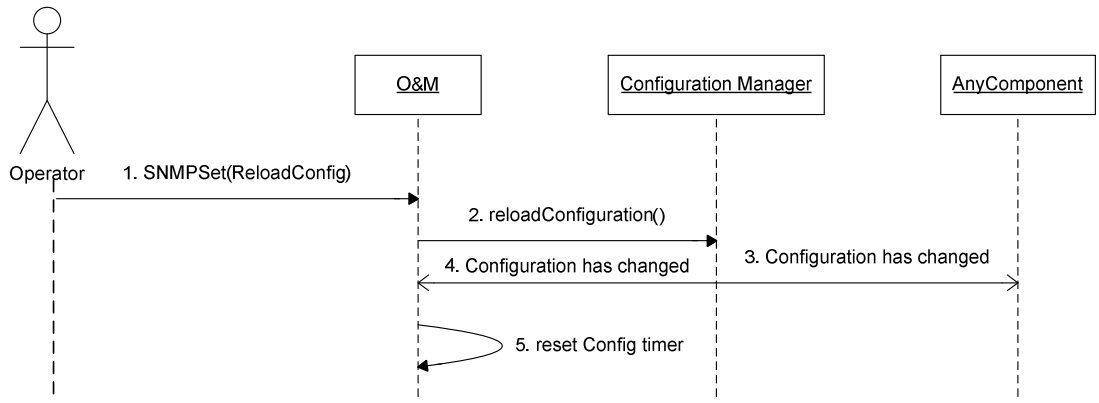


Figure 18 Dynamic load of configuration

1. The operator issues a ReloadConfig request
2. O&M request the Configuration Manager to reload the configuration
3. The Configuration Manager sends the Configuration event to all components that has registered as a receiver of that event.
4. When O&M receives this event …
5. … it resets the time that shows the time since last re-configuration.

4.6.2.2 Fault handling

The subsections below show how a faulty configuration file shall be handled. This can happen in case a system operator fails to edit the configuration in the correct way.

4.6.2.2.1 Start-up with faulty configuration files.

1. Configuration Manager creates a log entry describing what the fault is.
2. If no configuration backup is found, Configuration Manager terminates the MAS.

4.6.2.2.2 Start-up with correct configuration files

1. Configuration Manager creates a configuration backup

4.6.2.2.3 Load configuration with faulty configuration files

1. Configuration Manager creates a log entry describing what the fault is.
2. Configuration Manager does not send the Configuration has changed event
3. Configuration Manager keeps its existing, correct configuration
4. Configuration Manager notifies the system operator that the configuration was not loaded by returning an error to the load configuration command and set the configuration status in the MIB to "failed".
5. A client that requests a configuration parameter gets the existing correct configuration

4.6.2.2.4 Load configuration with correct configuration files

1. Configuration Manager creates a configuration backup

4.7 Out Dial Notification Service

This section describes how an Outdial Notification service request is performed. The description is valid for Pager Notification and Call MWI Notification services as well.

4.7.1 initiateOutdialNotification method

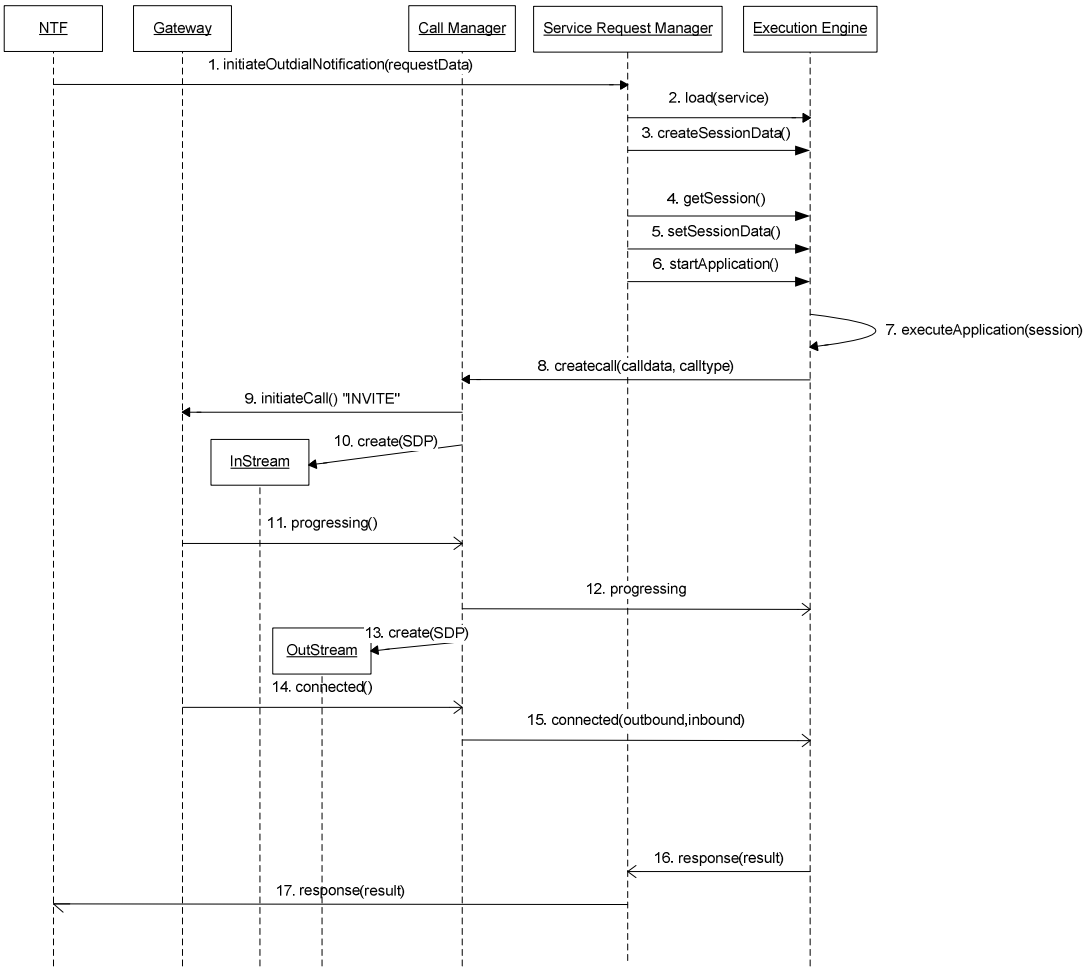


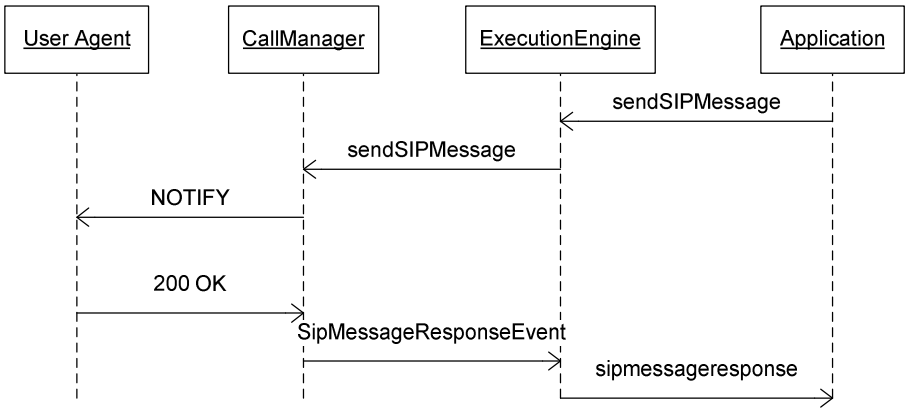
Figure 19 Initiate out dial notification

1. NTF (or any other XMP client) request a service from MAS.
2. Service Request Manager asks the Execution Engine to load an application corresponding to a certain service.
3. Call Manager creates a sessionData object in Execution Engine used to store session related data.
4. Service Request Manager retrieves the SessionData object and…
5. …sets certain data for example XMP client identity etc.
6. Service Request Manager then request to start the Application.
7. Execution Engine executes the Application.
8. Eventually the Execution Engine (based on the Application needs) request Call Manager to make an outbound call.
9. Call Manager initiates a SIP session and ...

10. ... creates the inbound Stream
11. The Gateway sends the progressing information
12. The Call Manager sends the alerting event to the Execution Engine.
13. Call Manager then creates the outbound Stream.
14. The call is accepted by the Gateway.
15. Call Manager sends the connected event to Execution Engine. The call is now established and the Outdial Notification application can start.
16. Execution Engine sends a response to Service Request Manager when the Application decides that the service is done.
17. Service Request Manager forwards the response to the external XMP Client. Service Request Mgr handles Validity timeout and also if response shall be sent to the client or not. Service Request Manager shall also notify the Execution Engine that if the validity timer expires it shall terminate the Session.

4.8 SIP Message Waiting

The SIP Message Waiting (see ref. [3]) can be sent on behalf of the application. The application uses PlatformAccess to initiate the SIP request. PlatformAccess has a general SIP request method, which the application uses in this case. The PlatformAccess method is agnostic of the particular SIP request the application initiates, and just passes the parameters to CallManager.



1. The application initiates a SIP Message Waiting request via PlatformAccess.
2. PlatformAccess invokes CallManager, supplying the parameters.
3. CallManager examines the parameters, and make a SIP NOTIFY request towards the gateway.
4. Eventually, the gateway responds with a 200 OK SIP message.

5. CallManager constructs a SipMessageResponseEvent event, including information about the outcome of the Message Waiting. In this scenario, "200 OK" was received, which means that the request was successful.
6. ExecutionEngine constructs a corresponding event, including the information received from CallManager, and delivers the recently constructed event to the application, which now is able to examine the outcome of the SIP request.

5 Thread Model Information

The MAS is multithreaded, but different components utilize threads in different fashions. The following section aims to outline the most significant parts. The functional description for each component describes the threading information used in each component in more detail.

5.1 Execution Engine

The model is founded on work in the different execution context to be carried out by threads allocated from a thread pool. Each execution context requests a thread from the pool when there is something to execute, e.g. evaluation of some ECMA Script, prompts to queued, or transitions to perform, an event to respond to etc. As soon as the execution context enters wait state again, e.g. when waiting for a response event from another component, or user input, the thread is returned to the pool. The context will awake again once an event occur, triggering another thread to be picked up from the pool in order to respond to the event.

The thread pool properties, such as max number of threads, can be configured in the system wide configuration file ComponentConfig.xml.

5.2 Stream

The Stream component is split into two different parts, one part implemented in C++ and one part implemented in Java.

The C++ part is responsible for the low-level RTP stack management. It may use multiple thread models, ranging from one thread per RTP stream, to a fixed size thread pool handling the RTP streams, to one single handler thread.

The java portion uses one thread per session/connection, in order to handle communication between the java and the C++ parts.

5.3 Call Manager

The Call Manager uses a configurable worker thread pool to handler the requests/responses, in a similar manner as the Execution Engine. The thread pool can be configured in the ComponentConfig.xml.

6 Design Rules

6.1 Case Sensitiveness

In order to minimize case-insensitive comparisons regarding textual information, the implementation of MAS shall follow the guidelines in this section.

The following MAS components have an external interface (including to real people) that might send textual information to MAS:

- Service Request Manager (XMP)
- Profile Manager (LDAP)
- External Component Register (LDAP)
- Call Manager (SIP)
- Stream (RTP)
- Mailbox (IMAP)
- Operate and Maintain Manager (SNMP)
- Execution Engine (VoiceXML/CCXML)
- Media Content Manager (Media Content Packages)
- Configuration Manager (XML configuration files)

Generally all text shall be handled according to the information it represents. In most cases it mean that standards used to handle the text, defines the case sensitiveness.

When a MAS component itself defines textual information, it shall always be used case sensitive and defined in lowercase (unless the information itself requires capitals).

When a MAS component creates textual information that is case insensitive it shall always use lower case.

A MAS component that receives textual information on its external interfaces shall do **one of**:

1. Case-insensitive textual information is converted to lowercase in the interfaces to the parts that encapsulates the protocols that receives the information.

2. Case-sensitive textual information is always handled as is.

Other MAS components can then always use case exact comparisons.

For the following components the case sensitiveness is not clearly defined shall be handled as described below:

- Service Request Manager. All attributes are handled as case sensitive and handled according to 2 above.
- Profile Manager. Currently there is no information in MUR regarding an attributes case. This means that both 1 and 2 above applies. Profile Manager shall provide functionality to specify (configure) per attribute if 1 or 2 shall be used.

- External Component Register. Because the information in MCR is created by software, they shall be handled according to 2 above.
- Media Content Manager. All information included in a Media Content Resource is handled according to 1 above except for the Media Content itself which is handled according to 2 above.
- Configuration Manager. All parameters are handled according to 2 above. Configuration Manager validates the configuration files with respect to case, using an XML schema.

7 References

[1] IWD-Message Format
3/155 19-1/HDB 101 02 Uen

[2] IWD-Managed Object Interface
2/155 19 CRH 109 083 Uen

[3] IWD – SIP Message Waiting
2.IWD.MAS0001

8 Terminology

SUD	Scalable User Directory
Term	Definition
User Agent	A logical entity that can act as both a user agent client and user agent server.

.