FD – Media Translation Manager

Content

History

Version	Date	Adjustments
A	2006-10-06	First revision. (BMEME)

1 Introduction

This document describes the Media Translation Manager (MTM). The MTM is involved in the use cases translate text to speech (TTS) and translate speech to text (ASR) (see ref. [1]

1.1 Text to Speech

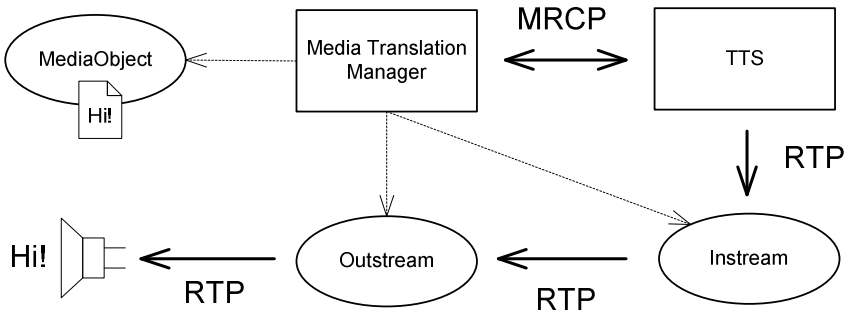
Text to speech transformation utilizes 3PP TTS engines for performing the actual transformation. The MTM is given the text which is to be spoken and an output

stream. The MTM is responsible for ensuring that the TTS engine is given the text in format that can be interpreted by the TTS engine and for directing the output from the TTS engine to the output stream.

The communication with the TTS engine comes in two flavors. The first is direct communication over MRCP/RTSP. The second is communication through MCC over XMP. The flavor to be used is configurable. It depend on if an MCC is installed or not.

1.1.1 Text to Speech over MRCP

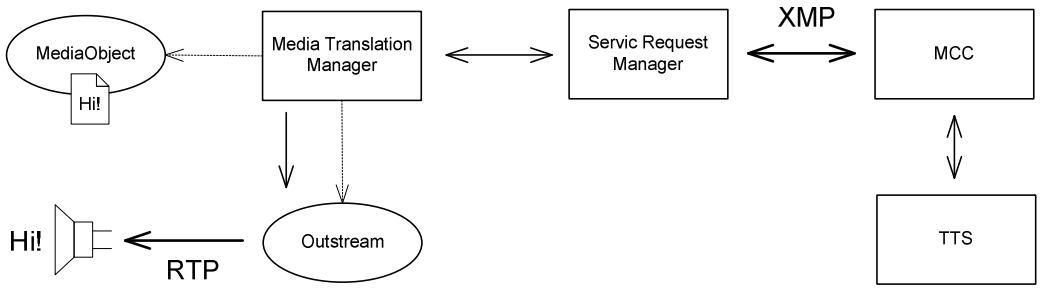
The MTM is configured for TTS over MRCP and is given a MediaObject and an OutputStream.



The MTM initiates an MRCP session based upon media properties defined by the OutputStream. The MTM ensures that the text, provided by the MediaObject, is in SSML, if necessary by transformation. The MTM creates an InputStream that streams the TTS engine output to the OutputStream and starts the speech by sending the SSML to the TTS engine. When the TTS engine signals that the speech is complete the MTM tear down the session and signals that the speech is completed.

1.1.2 Text to Speech using MCC

The MTM is configured for TTS using MCC and is given a MediaObject and an OutputStream.

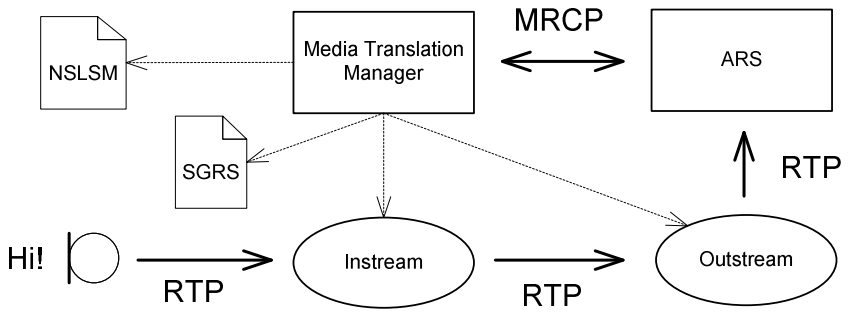


The MTM ensures that the text, provided by the MediaObject, is in a proper format (SSML or plain text), if necessary by transformation. The MTM issues a TTS request through the Service Request Manager (SRM). The SRM is responsible

for issuing an XMP TTS request to the MCC. The MCC transforms the text by utilizing the TTS engine and returns the audio data. The audio data propagates back to the MTM. The MTM streams the audio data through the OutputStream and signals speech completed at the end of data.

1.2 Automatic Speech Recognition

The MTM is given an InputStream and an SGRS grammar definition.

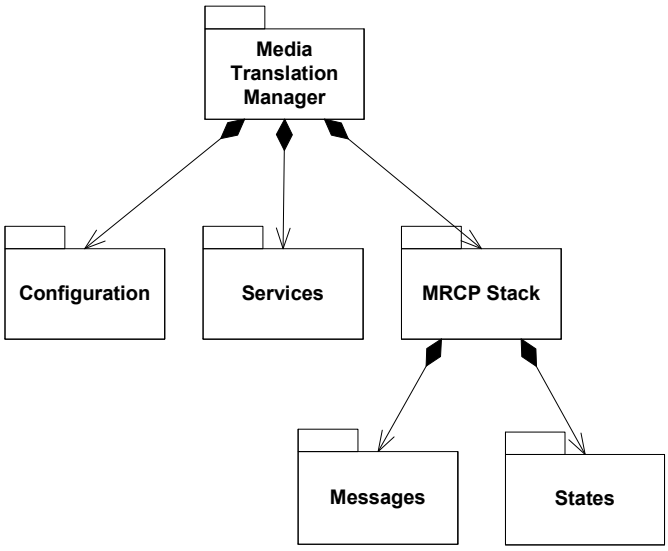


The MTM initiates an MRCP ASR session based upon media properties defined by the InputStream and the grammar definition. The MTM creates an OutputStream that streams the speech to the ASR engine and requests the ASR engine to start recognition. When recognition is completed the ASR engine returns the result in an NSLSML text. The MTM returns the NSLSML to the caller.

2 Function Structure

2.1 Package Structure

The Media Translation Manager is realized in the following packages.



The package Media Translation Manager contains the public parts of the MTM component.

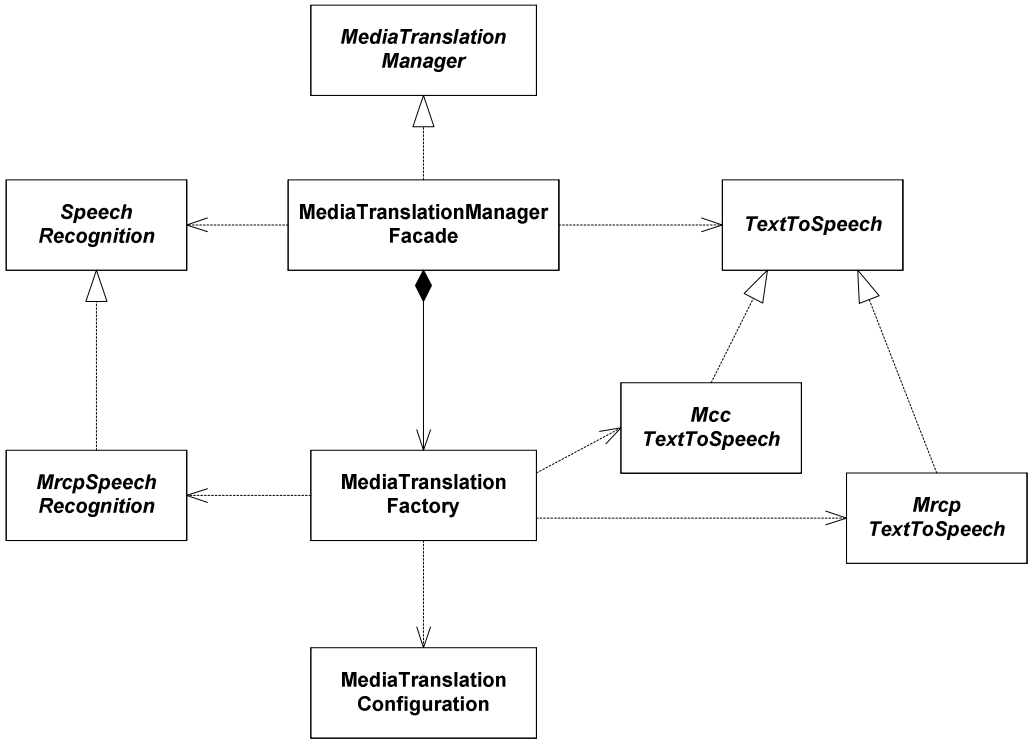
The package MRCP Stack contains the parts which handles TTS and ASR sessions.

The package Messages contains classes which are modeling MRCP/RTSP messages and parsing of such messages.

The package States contains classes handling the different states involved in TTS and ASR sessions.

2.2 Media Translation Manager

The following diagram is an overview of the classes in the Media Translation Manager Package.



The following sections describe the classes in individual.

2.2.1 MediaTranslationManager

This is a public interface which provides services for creating instances of TextToSpeech and SpeechRecognition.

2.2.2 TextToSpeech

This is the interface of the service translate text to speech, TTS.

2.2.3 SpeechRecognition

This is the interface of the service translate speech to text, ASR.

2.2.4 MediaTranslationManagerFacade

This class is the implementation of the interface MediaTranslationManager and provides means to create instances (through MediaTranslationFactory) of TextToSpeech and SpeechRecognizer.

This class is a façade for set injection (through Spring) of external references (ServiceLocator, ServiceRequestManager, StreamFactory and MediaObjectFactory).

2.2.5 MediaTranslationFactory

The MediaTranslationFactory is responsible for the creation of TextToSpeech and SpeechRecognizers. The factory provides methods for the creation of each. The factory determines which kind of TTS/ASR to create from the MediaTranslationConfiguration (currently TTS over MRCP or XMP, ASR MRCP only).

2.2.6 MediaTranslationConfiguration

This class holds information about things like which kind of TTS/ASR to create (actually which kind of protocol to use when communicating with the external TTS/ASR engine).

2.2.7 MccTextToSpeech

This class implements the TextToSpeech interface. MccTextToSpeech utilizes the MCC component through the ServiceRequestManager (utilizing XMP).

2.2.8 MrcpTextToSpeech

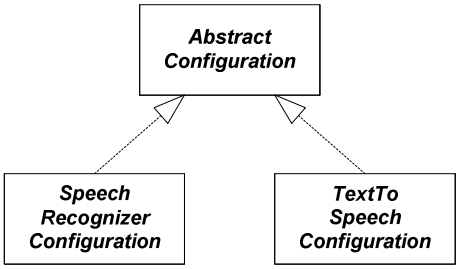
This class implements the TextToSpeech interface. MrcpTextToSpeech utilizes an external TTS engine through the MRCP protocol.

2.2.9 MrcpSpeechRecognizer

This class implements the SpeechRecognizer interface. MrcpSpeechRecognizer utilizes an external ASR engine through the MRCP protocol.

2.3 Configuration

This package implements convenience wrappers for the general configuration handling. The classes here in wraps an simplifies parameter lookup (with default values).



The following sections describe the classes in individual.

2.3.1 AbstractConfiguration

This class generalizes the service (TTS/ASR) configuration. There should be one for each.

2.3.2 SpeechRecognizerConfiguration

This class holds the configuration of the speech recognition service.

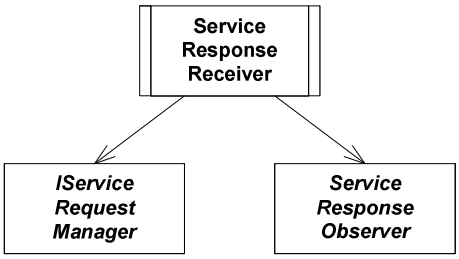
2.3.3 TextToSpeechConfiguration

This class holds the configuration of the text to speech service.

2.4 Services

This package implements the handling of responses to external service requests.

When MTM has issued a service request a ServiceResponseReceiver is created. The SerivceResponseReceiver is waiting for the ServiceRequestManager to complete the transaction. MTM is notified through the ServiceResponseObserver.



The following sections describe the classes in individual.

2.4.1 ServiceResponseReceiver

The purpose of the ServiceResponseReceiver is handle responses from asynchronous service requests.

This class is responsible for monitoring a ServiceRequestManager and wait for a specific transaction to complete. Once the transaction completes the result is sent to the registered ServiceResponseReceiver.

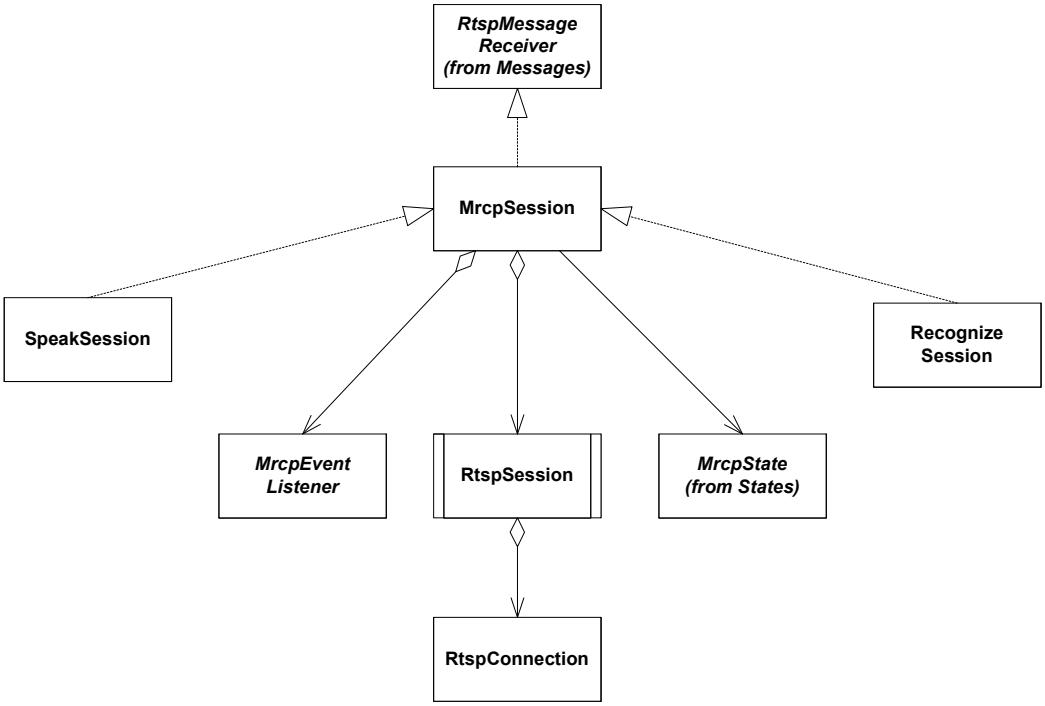
2.4.2 ServiceResponseObserver

This is an interface which is implemented by anyone (MTM in particular) in interest of service responses.

NOTE: in this design the purpose is to receive the result of TTS performed by the MCC component.

2.5 MRCP Stack

This package implements an MRCP stack (see ref. [7] based upon RTSP (see ref. [6] .



The following sections describe the classes in individual.

2.5.1 MrcpSession

This class represents an MRCP session. An MRCP session relies upon an RTSP session. The MrcpSession implements the interface RtspMessageReceiver in order to be able to handle MRCP events (RTSP requests, see ref. [7] The MRCP events are supposed to be handled by an MrcpEventListener (typically the user of the session).

2.5.2 MrcpEventListener

The user of an MrcpSession is supposed to implement and register an MrcpEventListener in order to handle MRCP events (typically asynchronous status messages except for the ASR NLSML response).

2.5.3 RtspSession

This class represents an RTSP session. RtspSession is responsible for the transmitting and receiving RTSP messages. Due to the asynchronous nature of MRCP the RTSP messages are requests and responses (in both directions).

In order to be network independent, which is good at least when designing the basic tests, the socket handling is implemented in RtspConnection (actually in RtspConnectionImpl).

2.5.4 RtspConnection

This is an interface for an RTSP connection. RtspConnection is an abstraction of a TCP/IP connection. Read and write are handled through InputStream/OutputStream (java.io).

Since this is an interface the TCP/IP communication is simple to mock and hence it is simple to test the MRCP stack.

2.5.5 SpeakSession

This class is a specialization of an MrcpSession. The SpeakSession provides TTS related access methods.

2.5.6 RecognizeSession

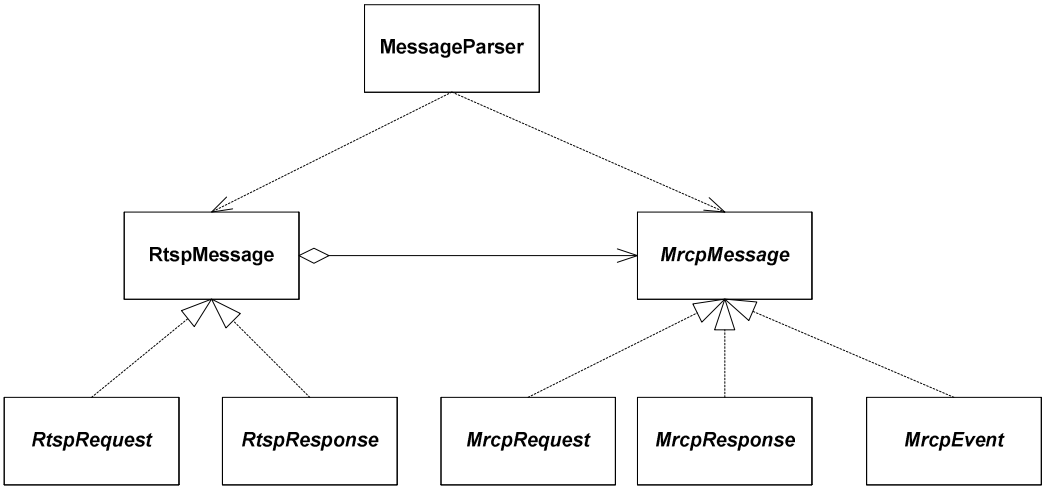
This class is a specialization of an MrcpSession. The RecognizeSession provides ASR related access methods.

2.6 Messages

This package contains the various MRCP and RTSP messages (requests, responses and events). The actual messages are not mentioned here since proper documentation is available in ref. [7] and [6]

Here is an overview of the structure of the message base classes. An MRCP message is transported as an attachment of an RTSP message. An RTSP message can be either a request or a response (to a request). An MRCP message can be one of: request, response or event. The RTSP request/response is synchronous. Hence an RTSP response is a response to an RTSP request.

Since an MRCP message is carried by an RTSP message an MRCP response/event is carried by an RTSP request (see ref. [7]).



The following sections describe the classes in individual.

2.6.1 RtspMessage

This class is a eneralization of any RTPS message. The RTSP message can have a reference to an MrcpMessage.

2.6.2 MrcpMessage

This class is a generalization of any MRCP message.

2.6.3 RtspRequest

This class specialize an RTSP message into a request.

2.6.4 RtspResponse

This class specialize an RTSP message into a response

2.6.5 MrcpRequest

This class specialize an MRCP message into a request.

2.6.6 MrcpResponse

This class specialize an MRCP message into a response.

2.6.7 MrcpEvent

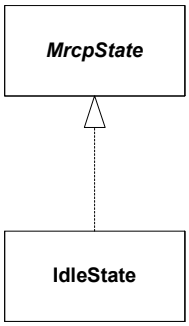
This class specialize an MRCP message into an event.

2.6.8 MessageParser

This class is responsible for parsing incoming messages into instances of (MRCP/RTSP) request/response/event.

2.7 States

This package implements the MRCP states of the TTS/ASR state machine (see ref. [7] . Each state is implemented as Singleton. Since the state classes are one-to-one with the MRCP specification there is no need to list them all in this document.



The following sections describe the classes in individual.

2.7.1 MrcpState

This is an abstract class, a generalization, of an MRCP state. MrcpState provide an interface for reacting upon MRCP events and MRCP session user actions. It is up to each specialization (e.g. the IdleState) to implement the specific behavior and transitions.

2.7.2 IdleState

This is the MRCP idle state which is a specialization of MrcpState. The IdleState implements means to enter the TTS/ASR active state (speak/recognize).

NOTE: It is up to the MrcpSession specialization (SpeechSession/RecognizeSession) to access the proper methods (e.g. it is not possible to start recognition from a speech session).

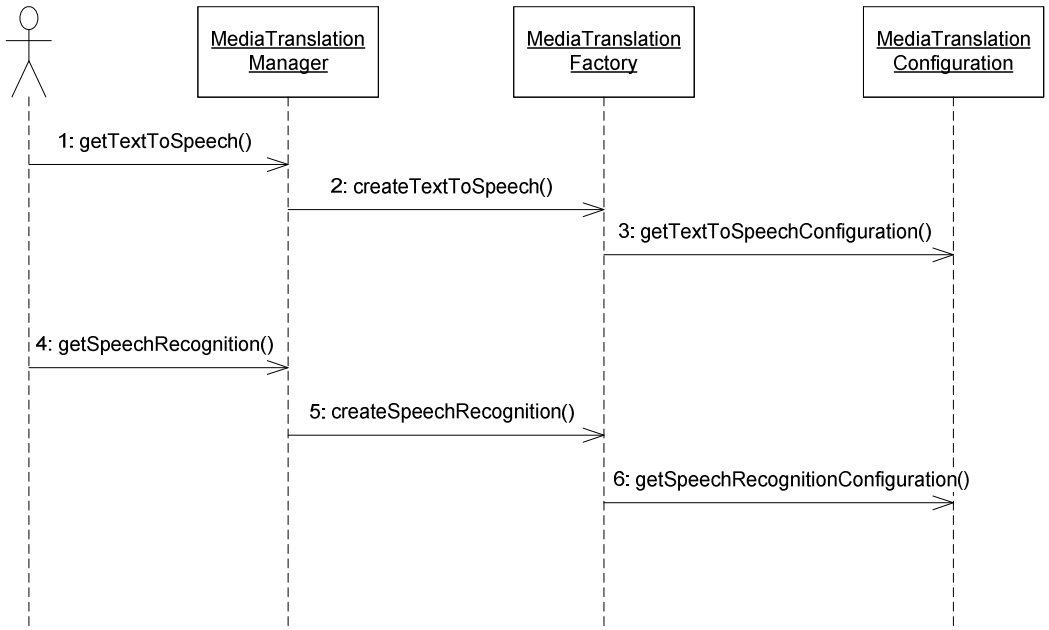
3 Function Behavior

In this section the function of the Media Translation Manager is described. The purpose here is not to describe every little detail but to give an overview and the general picture of the design. The details are found in the code.

The general flow of the MTM use case is first to create a service instance and then utilize its features.

3.1 Create Services

Here is an overview of the interaction of the classes involved when creating either a TextToSpeech or SpeechRecognition instance. Yes, it's trivial, but the purpose is to show that the configuration is used by the factory.



This is the flow of the creation of a TextToSpeech:

1. A user (typically Stream) calls method getTextToSpeech (implemented by MediaTranslationManagerFacade) in order to have an instance of text to speech.
2. The façade utilizes the MediaTranslationFactory to create an instance of TextToSpeech.
3. The MediaTranslationFactory retrieves the text to speech configuration and determines which kind of TTS to create (MrcpTextToSpeech or MccTextToSpeech).

This is the flow of the creation of a SpeechRecognition:

1. A user (typically Stream) calls method getSpeechRecognition (implemented by MediaTranslationManagerFacade) in order to have an instance speech recognition.
2. The façade utilizes the MediaTranslationFactory to create an instance of TextToSpeech.
3. The MediaTranslationFactory retrieves the speech recognition configuration and determines which kind of ASR to create (MrcpSpeechRecognition).

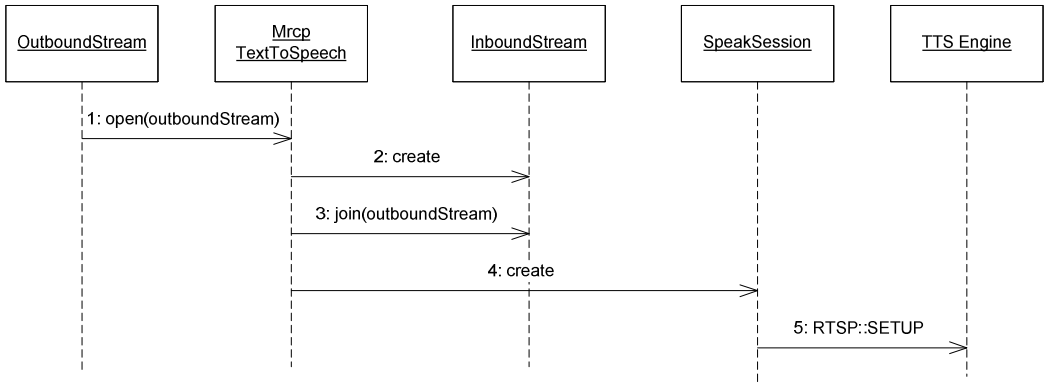
Once created the instances are ready for use.

3.2 Text to Speech over MRCP

This section shows the activities involved when utilizing Text to Speech over MRCP.

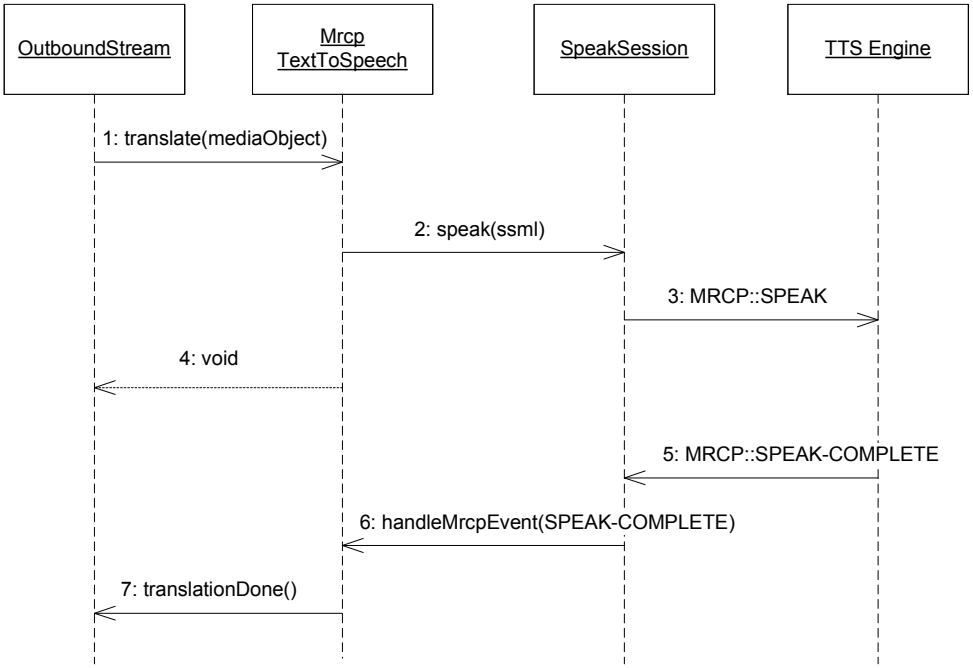
3.2.1 Open

The first step is to set up an MRCP TTS session.



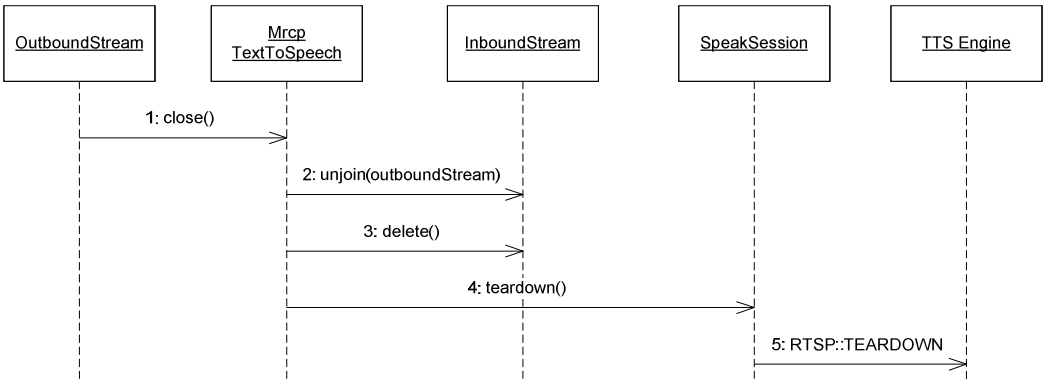
1. The user (typically an OutboundStream) opens the session and provides the OutboundStream to which the generated speech will be sent.
2. MrcpTextToSpeech creates an InboundStream which is compatible with the OutboundStream.
3. MrcpTextToSpeech joins the inbound and outbound streams.
4. MrcpTextToSpeech creates a SpeakSession "connected" to the inbound stream.
5. An RTSP set up request is sent to the TTS engine. The TTS engine sets up a TTS session which will be streaming to the inbound stream and sends back a ok response.

3.2.2 Translate



1. The user starts the translation by providing a MediaObject containing the text to be spoken.
2. MrcpTextToSpeech retrieves the text from the MediaObject and ensures that the text is SSML (plain text is wrapped) and sends the SSML to the SpeakSession.
3. The speak session sends a MRCP speak request (which includes the SSML) to the TTS engine. The TTS engine starts the translation and sends back an MRCP in progress response.
4. MrcpTextToSpeech returns the thread over to the caller/user.
5. Once the TTS engine has finished the translation it sends a MRCP speak complete event.
6. When the speak completes the SpeakSession calls notifies the MrcpTextToSpeech.
7. MrcpTextToSpeech notifies the OutboundStream that it has completed the translation.

3.2.3 Close



1. The user calls close.
2. MrcpTextToSpeech unjoins the streams.
3. MrcpTextToSpeech deletes the InboundStream.
4. MrcpTextToSpeech stops the MRCP TTS session.
5. The speak session sends a RTSP session teardown request. The TTS engine terminate the RTSP session.

3.2.4 Error handling

The activities here above describe the ideal behavior. The following alternatives are handled as errors. What ever the cause is it will result in a call of the translation failed method (on the outbound stream).

3.2.4.1 Unknown text format

The media translation manager is responsible for ensuring that the text sent to the TTS engine is SSML. If the media object contains other formats than plain text and SSML the media translation manager will fail to translate.

3.2.4.2 Unknown audio format

If the outbound stream does not support PCMU audio text to speech translation can not be performed.

3.2.4.3 MRCP error

MRCP error messages are handled.

3.2.4.4 Stream errors

The media translation manager can fail to create an inbound stream, or to join the inbound and outbound streams.

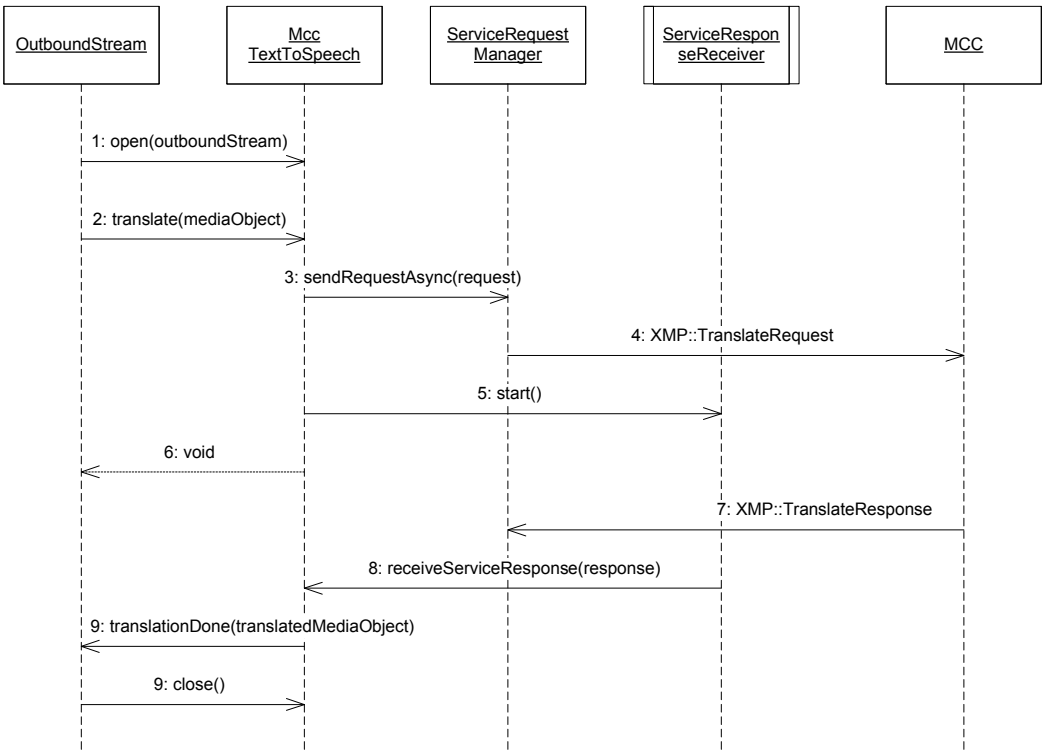
3.2.4.5 Text translation failure

If the TTS engine encounters an SSML parse error it will respond to the MRCP speak request with an MRCP error response.

3.2.4.6 No Speak Complete timeout

If the RTSP session is, by some reason, lost this is detected by the session handler.

3.3 Text to Speech over XMP



1. An actor (typically OutboundStream) calls open and passes an OutboundStream which the is supposed to play the speech. The MccTextToSpeech initializes some variables.
2. The actor calls translate and passes a MediaObject containing the text to be spoken. The MccTextToSpeech ensures that the text is raw text (SSML tags are stripped).
3. MccTextToSpeech creates a text to speech ServiceRequest and sends it to the ServiceRequestManager.
4.
5. MccTextToSpeech starts the ServiceResponseManager to wait for the ServiceRequest to complete (the returned transaction ID is used).
6. MccTextToSpeech returns the thread over to the caller.
7.
8. Once the service transaction completes the ServiceResponseReceiver calls the MccTextToSpeech and passes the response.

9. MccTextToSpeech retrieves the speech data from the response and inserts it into a new MediaObject and sends it to the OutboundStream.
10. Close does not perform anything in particular.

3.3.1 Error handling

The activities here above describe the ideal behavior. The following alternatives are handled as errors. What ever the cause is it will result in a call of the translation failed method (on the outbound stream).

3.3.1.1 Incompatible media

If the outbound stream does not support PCMU audio the media translation manager will fail to create a text to speech translator.

3.3.1.2 XMP errors

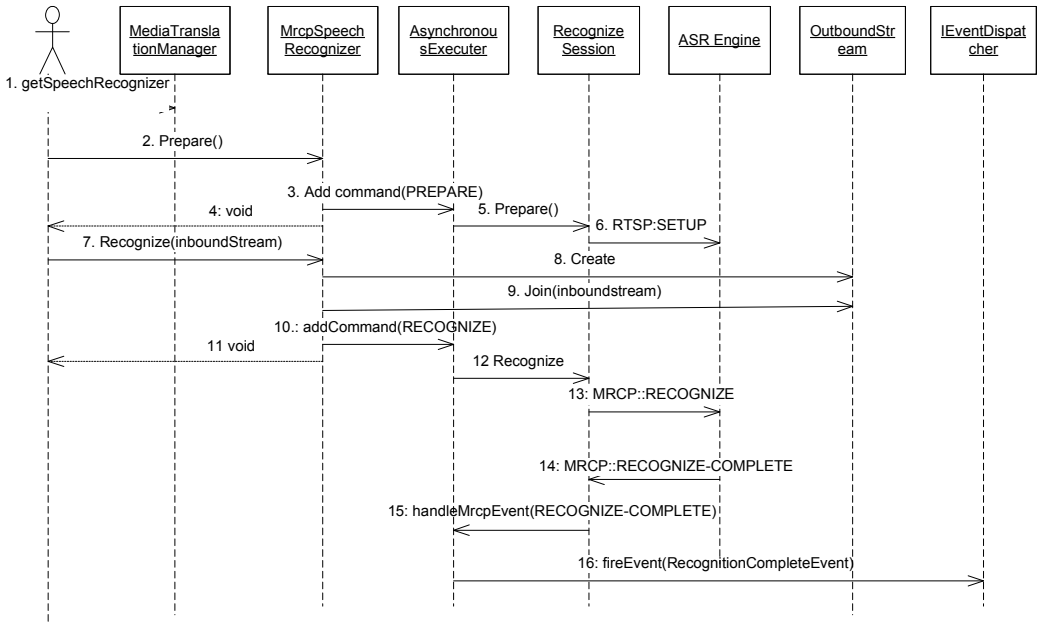
If there are any XMP errors or errors reported by the MCC encountered during the translation the media translation manager will fail to complete the text to speech translation.

3.3.1.3 Stream errors.

3.3.1.4 Response timeout.

3.4 Speech Recognition over MRCP

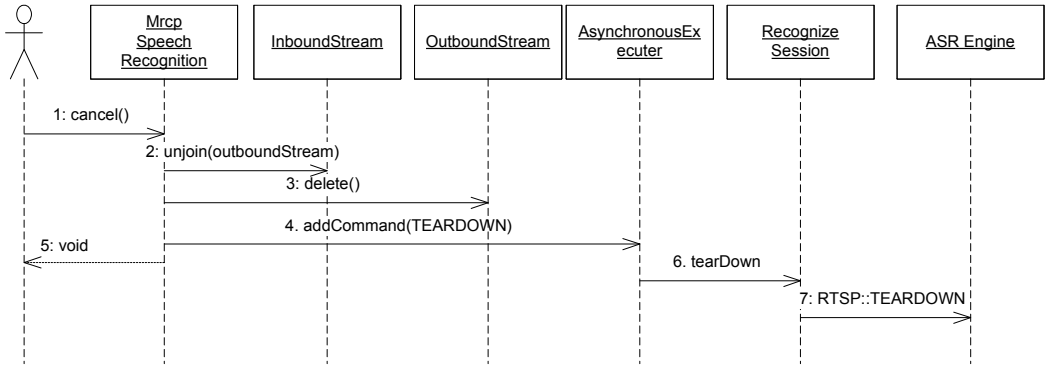
3.4.1 Recognize



1. The user (typically ExecutionEngine) starts the translation by retrieving a SpeechRecognizer. The user supplies the grammars.
2. The user invokes prepare() on the MrcpSpeechRecognizer.
3. The MrcpSpeechRecognizer adds the command PREPARE on AsynchronousExecuter. A new thread from the thread pool will be retrieved.
4. The prepare() call invoked by the user returns.
5. The AsynchronousExecuter invokes prepare() on the RecognizeSession.
6. The RecognizeSession issues a RTSP:SETUP request.
7. The user starts a recognition. The InboundStream is supplied.
8. An OutboundStream is created.
9. The inbound and outbound streams are joined.
10. The MrcpSpeechRecognizer adds the command RECOGNIZE on the AsynchronousExecuter.
11. The prepare() call invoked by the user returns.

12. The AsynchronousExecuter invokes recognize request on the RecognizeSession.
13. The RecognizeSession issues an MRCP:RECOGNIZE request.
14. A MCRP:RECOGNIZE-COMPLETE is sent back from the ASR Engine.
15. The RecognizeSession invokes the AsynchronousExecuter to inform that the recognition is complete.
16. The AsynchronousExecuter fires a RecognitionCompleteEvent, which the user (typically ExecutionEngine), which now can analyse the outcome of the recognition.

3.4.2 Cancel



1. The actor calls cancel.
2. MrcpSpeechRecognition unjoins the streams.
3. MrcpSpeechRecognition deletes the OutboundStream.
4. MrcpSpeechRecognition adds the command TEARDOWN on the AsynchronousExecuter. A new thread from the thread pool will be retrieved.
5. The cancel() call invoked by the actor returns.
6. The AsynchronousExecuter invokes teardown() on the RecognizeSession.
7. The RecognizeSession issues an RTSP:TEARDOWN request.

3.4.3 Error handling

The activities here above describe the ideal behavior. The following alternatives are handled as errors. What ever the cause the media translation manager will issue a recognition failed event.

3.4.3.1 Unknown grammar format

The media translation manager is responsible for ensuring that the grammar text sent to the ASR engine is SRGS if not the media translation manager will fail to perform the speech recognition.

3.4.3.2 Unknown audio format

If the inbound stream does not provide PCMU audio speech the media translation manager will fail to perform the speech recognition.

3.4.3.3 MRCP error

MRCP error messages are handled.

3.4.3.4 Stream errors

The media translation manager can fail to create an outbound stream, or to join the inbound and inbound streams.

3.4.3.5 Recognition failure

If the ASR engine encounters an SRGS parse error it will respond to the MRCP set grammar with an MRCP error response. The recognition will also fail if a grammar id is undefined.

3.4.3.6 No Recognition Complete timeout

If the RTSP session is, by some reason, lost this is detected by the session handler.

3.5 MRCP Stack

The MRCP Stack is implemented as a state machine. Since the MRCP specification defines state machines both for TTS and ASR the states, events etc are copied into the design described here below.

Please note that some events/actions are left out since the functionality is not required. Hence the state machines here below are subsets of the state machines in the MRCP specification. As a consequence some MRCP features are not supported or utilized and some MRCP events are not handled (ignored).

The states in the state machines reflects the states of MRCP server side and handles the actions and events necessary for the MTM to fulfill the stipulated requirements.

3.5.1 Text to Speech State Machine

The TTS state machine starts when a TTS session is opened and will remain active until the session is closed.

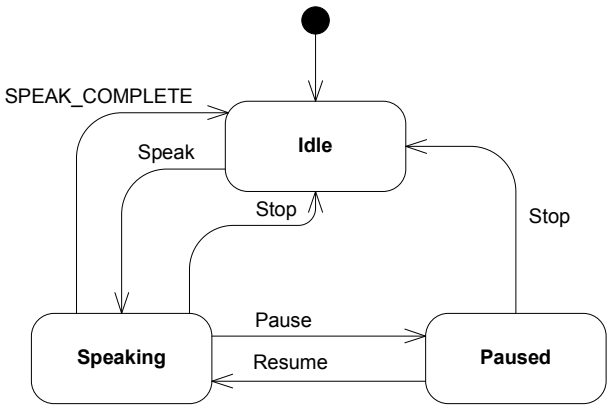


Figure 1 TTS State Machine

3.5.1.1 Idle

In the Idle state the state machine is ready to start a text to speech translation.

A translation is started by a Speak request. An MRCP *SPEAK* request is transferred to the TTS engine. The TTS engine will parse the text and respond with an MRCP *IN-PROGRESS* (or an error message) response. When receiving the *IN-PROGRESS* a transition to the Speaking state is taken.

3.5.1.2 Speaking

The state machine will remain in the Speaking state until the speak is completed (successfully or not), stopped or paused.

When receiving the MRCP *SPEAK-COMPLETE* event, the completion cause is checked in order to determine whether the translation was successful or not. The caller is notified about the result. A transition to the Idle state is taken.

The Pause command will issue an MRCP *PAUSE* request. A transition to the Paused state is taken.

The Stop command will issue an MRCP *STOP* request. A transition to the Idle state is taken.

3.5.1.3 Paused

In the Paused state the speak is paused. Possible actions are Resume or Stop.

The Resume command will issue an MRCP *RESUME* request. A transition to the Idle state is taken.

The Stop command will issue an MRCP *STOP* request. A transition to the Idle state is taken.

3.5.2 Speech Recognizer State Machine

The ASR state machine starts when an ASR session is opened and will remain active until it is closed.

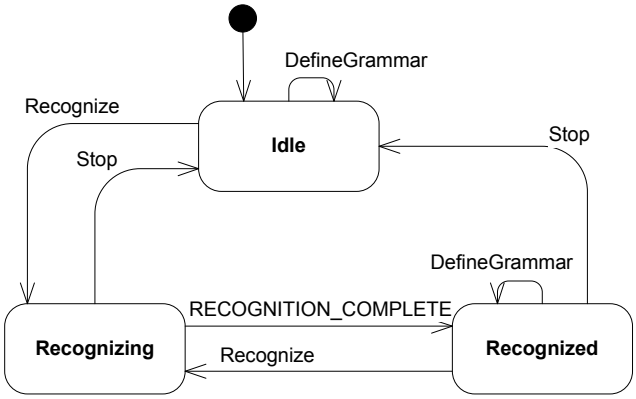


Figure 2 ASR State Machine

3.5.2.1 Idle

When in the Idle state the ASR engine is, idle, and ready for action. Possible actions/commands are DefineGrammar and Recognize.

The Recognize command will issue an MRCP *RECOGNIZE* request. The request is considered to be successful when receiving an MRCP *IN-PROGRESS* response. Upon a successful request a transition to the Recognition state is taken.
Please note that Recognize comes in two flavors, one where the grammar is attached to the request and one where the request is referring (by ID chosen by caller) to previously defined grammars.

The DefineGrammar command will issue an MRCP *DEFINE-GRAMMAR* request. The request is considered to be successful when receiving an MRCP *COMPLETE* response. The state machine will remain in the Idle state.

3.5.2.2 Recognizing

In the Recognizing state the ASR engine is considered to be busy processing the speech. Possible actions here are the RecognitionComplete event and the Stop command.

When receiving the MRCP *RECOGNITION-COMPLETE* event the completion cause is checked. If the cause is success then the recognized text (in NLSML) is extracted from the event and passed over to the caller. A transition to the state Recognized is taken.
In the case of unsuccessful the cause is reported to the caller. The cause can be one of: "no speech timeout", "could not match speech with grammar" and "general error". A transition to the Idle state is taken.

The Stop command will issue an MRCP *STOP* request. A transition to the Idle state is taken.

3.5.2.3 Recognized

This state is somewhat redundant in our implementation since the possibility to retrieve the result is not implemented here. So, this state is very much the same as the Idle state.

The Recognize command is handled in the same manner as in the Idle state.

The DefineGrammar command will issue an MRCP *DEFINE-GRAMMAR* request. The request is considered to be successful when receiving an MRCP *COMPLETE* response. The state machine will remain in the Recognized state.

The Stop command will issue an MRCP *STOP* request. A transition to the Idle state is taken.

3.6 Thread Model

The Media Translation Manager utilizes the thread pool fro creating/allocating threads.

The MRCP Stack is using one thread for handling each RTSP session. There is one RTSP session for each instance of MrcpSpeechRecognition and MrcpTextToSpeech.

The MccTextToSpeech is also consuming a thread when pending on XMP responses. Hence there is one thread for each instance of MccTextToSpeech. The thread is used by the ServiceResponseReceiver.

For every MRCP request, a new thread is retrieved from the thread pool and this thread starts up in the AsynchronousExecuter class.

4 References

[1] FS – Media Translation Manager
15/FS-MAS001

[2] FS – Stream
7/FS-MAS001

[3] FS – Service Request Manager
10/FS-MAS001

[4] FS – Media Object
13/FS-MAS001

[5] FS – Media Content Manager
14/FS-MAS001

[6] RFC 2326 – Real Time Streaming Protocol (RTSP)

[7] Internet-Draft – Media Resource Control Protocol (MRCP)

[8] RFC 2327 – SDP: Session Description Protocol

5 Terminology

Term	Explanation