Application Note

Media Resource Control Protocol

Implementing a Media Resource Control Protocol (MRCP) Client Application with Dialogic® Telecommunications Products

Executive Summary

The Media Resource Control Protocol (MRCP), a proposed standard within the Internet Engineering Task Force (IETF), provides the means for a client device requiring audio streams to control stream processing resources in the network. Applying this definition to Dialogic's telecommunications product line, MRCP can be used to control speech synthesizers and recognizers that provide speech recognition, and to stream audio from a common location to a user. There are several benefits for adopting MRCP for media requirements:

- MRCP provides a mechanism to provide the same audio to both phone and web customer interfaces
- MRCP does not lock a solution into one vendor
- During the development effort, Dialogic worked with several major speech vendors who are moving to an MRCP-only interface for their text-to-speech and speech recognition products

This application note focuses on the work Dialogic has done to establish a starting point for integrating MRCP version 1.0 into applications based on Dialogic® telecommunications products. This application note refers to the following software:

- MRCP Client Library
- MRCP Client Verification application
- A sample integration application using the MRCP client library

Table of Contents

Introduction

The Media Resource Control Protocol (MRCP) represents a paradigm shift for telephony application developers using speech technologies. Instead of using vendor-specific commands to generate text-to-speech (TTS) or perform automatic speech recognition (ASR), MRCP commands are sent from the client to the server application. Additionally, instead of loading pre-recorded audio files locally, commands are sent to an MRCP server containing a URL of the file to be played to the caller. These commands generate an audio stream sent to the client application which should then direct it to the caller. This application note covers the mechanics of directing the audio to a caller via Dialogic® telecommunications components from Dialogic. In summary, MRCP presents a new application layer protocol using existing network protocols to control a user's audio experience.

When this application note was written, MRCP version 1.0 was supported by a majority of speech services vendors providing MRCP support. Version 1.0 uses the Real Time Streaming Protocol (RTSP) to establish connections from an MRCP client application to an MRCP server. All MRCP commands are then tunneled via RTSP Announce messages between the MRCP client and server. Version 2.0 of MRCP will replace the use of RTSP for command and control with the session initiation protocol (SIP). Although Dialogic's MRCP client library was designed to be easily extensible to support MRCP version 2.0, Dialogic has developed a separate MRCP version 2.0 reference library. Refer to the "Architecture" section for additional information.

MRCP commands between the client and the server are used to establish (bi-directional) audio paths between the client and server using the Real time Transport Protocol (RTP). RTP packets sent from the client application to the MRCP server contain the caller's speech, commonly referred to as utterances, sent for recognition. Conversely, RTP packets sent from the server to the client contain either TTS (speech synthesis) or streamed audio retrieved from a web server. A majority of the speech vendors require this RTP traffic to be encoded using the G.711 coder to ensure quality audio for recognition and delivery to the caller. Dialogic's MRCP client library contains an RTP broker to ensure that MRCP is possible across Dialogic's telecommunications product line regardless of the components' RTP capabilities. Refer to the "Architecture" section for additional information.

Architecture

MRCP supports a multitiered client server architecture. In a typical MRCP implementation, multiple clients establish sessions with one or more MRCP servers that can be directed to stream audio, located on multiple web servers, to the clients. It is possible to use one MRCP server for recognition and a separate server for TTS or streaming file play. Implementers may choose MRCP servers from multiple vendors based on a vendor's strength or ability to support a particular language such as French or Spanish.

Figure 1 illustrates a network topology for an MRCP client server implementation. Due to the time-sensitive nature of MRCP traffic:

- MRCP servers are to be deployed on the same network segment as the MRCP clients
- MRCP servers and clients are deployed on the same network segment as the web server containing audio content to be streamed
- Bandwidth on the network segment supporting MRCP are managed to give priority to MRCP commands and RTP traffic

In an MRCP implementation based on Dialogic's telecommunications products, the MRCP clients would contain the products used to communicate with the caller. Dialogic's MRCP client library, available as an accompanying download to this document, provides an RTP broker that allows products without RTP support to be used in an MRCP-based solution.

MRCP Client Library

Dialogic's MRCP client library is a simple, lightweight class library written in C++. This library supports the functionality required for developing an MRCP-based application. Functionalities like MRCP state machines, state transitions, and memory management must be provided in the client application developed using the library.

The classes defined within the MRCP client library include:

- MrcpClient class
- MrcpRecognize class
- MrcpRtpHandler class
- RtspPacket class
- MrcpUtils class

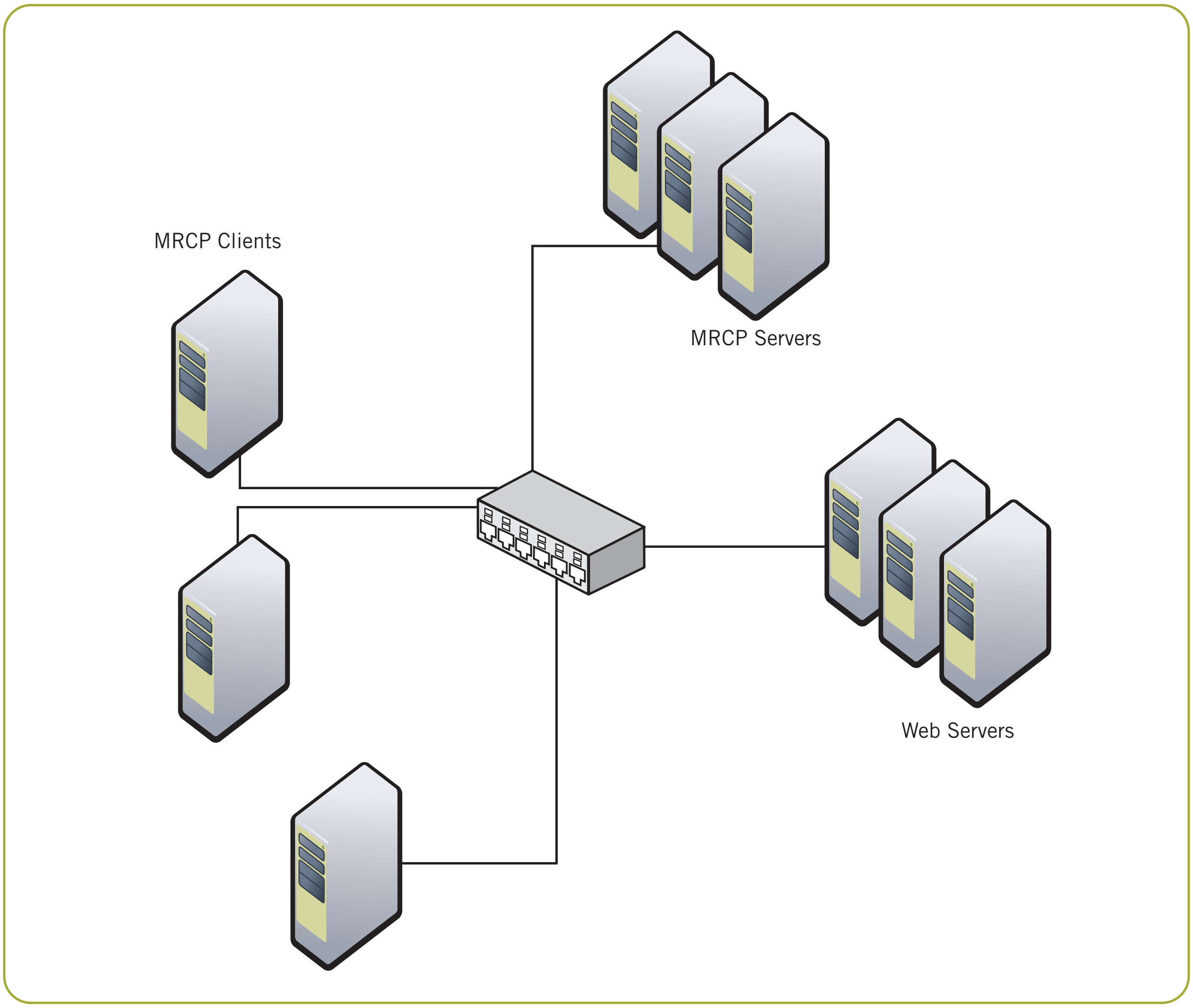

Figure 1. MRCP Multitiered Client Server Implementation

Class hierarchy within the class library is flat, designed to be easily understood and extended, adding complexity or features as desired. Detailed information on each class follows.

MrcpClient Class

This is the main class that must be instantiated by all applications using the MRCP client functionality provided within the application. A single instantiation of the MrcpClient class will support one recognizer and one synthesizer session. The recognizer functionality supports speech recognition on the MRCP server, while the synthesizer session supports TTS and streaming file play.

Session identifiers returned by the MRCP server when a session is initiated are maintained within the library, eliminating the need for the client application to track this value. The library also maintains sequence identifiers (increments occur as needed) and has access to pending request sequence numbers (available as needed). Additional information can be found in the "MRCP Client Library Usage Overview" section, as well as the *Media Resource Control Protocol (MRCP) Client Library User's Guide* document.

MrcpRecognize Class

This class must be instantiated by the client application wishing to use the speech recognition method within the MrcpClient class. The MrcpRecognize class contains all accessor methods necessary to set the parameters used by the MRCP server to perform speech recognition. Each parameter within the class has a default value that provided a good level of recognition during the development and testing of the MRCP client library.

Construction and destruction of this class are the responsibility of the client application using the library. Additional information is provided within the "MRCP Client Library Usage Overview" section of this document.

MrcpRtpHandler Class

This class is instantiated by the MrcpClient class when a recognizer or synthesizer session with the MRCP server begins providing RTP broker services. This class strips the RTP headers from packets sent from the MRCP server and adds RTP headers to packets sent from the MRCP client. Use of this class helps support a wide range of Dialogic's communications products in an MRCP-based solution. Any of Dialogic's communications products that support **dx_playiottdata** and **ec_stream** can use this library to develop an MRCP client application.

Although construction and destruction of this class are handled within the MrcpClient class, the application using the library must provide Standard Template Library (STL) container class pointers of type STL string on speak and recognize methods in the MrcpClient class. Additionally, methods contained within the MrcpClient class provide a mechanism to shut down the RTP streams and may be needed based on the design of the client application. Additional information can be found in the "MRCP Client Library Usage Overview" section of this document.

RtspPacket Class

This class provides an abstract implementation and is responsible for adding RTSP headers to all MRCP commands sent to the MRCP server. The MRCP client library uses the version parameter in the MrcpClient class constructor to instantiate the protocol class matching the version of MRCP in use. Dialogic has developed an MRCP version 2.0 reference library. See the "For More Information" section for the related application notes.

MrcpUtils Class

This class is never instantiated; rather, its methods are called directly by classes contained within the MRCP client library. The methods included within the class provide network socket functionality, parsing functions, and data conversion methods.

MRCP Client Library Usage Overview

This section provides information regarding the use of the MRCP client library as learned during the development of the MRCP client library and related documents. Refer to the *Media Resource Control Protocol (MRCP) Client Library User's Guide* and the MRCP Client Library source code for additional information on library use. The sample application available for download with this document (see the "For More Information" section) contains a basic MRCP client implementation example, and code snippets from it will be presented in the following sections.

Getting Started

The first step in using the MRCP client library is instantiating the MrcpClient class. Generally, an MRCP client application establishes two sessions with the MRCP server during the course of a call. One session is used for recognition; the other is used for speech synthesis or streaming file play.

The InitializeMrcpSession method in the MrcpClient class is used to establish both the synthesizer and recognizer sessions with an MRCP server. This method is called once for each established session. Generally, the recognizer and synthesizer sessions are torn down as soon as their task has completed or has been interrupted. The TearDownMrcpSession method in the MrcpClient class provides the means to tear down the MRCP session to the server.

Using Automatic Speech Recognition

Generally, MRCP client applications provide automatic speech recognition (ASR) and have a caller's speech or utterances trigger barge-in processing. In order to support ASR in a barge-in mode, the recognizer session must be established with the MRCP server in advance of any audio being played that supports barge-in. MRCP client library methods must be called in this order to perform recognition:

- InitializeMrcpSession
- MrcpDefineGrammar
- MrcpRecognizeSpeech
- TearDownMrcpSession

All code samples shown are from the reference application that is available for download with this application note.

InitializeMrcpSession

This method initializes the MRCP session with the MRCP server for a recognizer session.

```
string mrcpClient = m_ipParms.mrcpIpClient;
                              //this is the client address of the Dialogic
comms server
string mrcpPort = m_ipParms.mrcpPort;
nRet = m_pMrcpClient->InitializeMrcpSession(
                                          "recognizer",mrcpClient,mrcpPort);
if (200 != nRet)
{
    printf("<———-MRCP initialization ERROR = %d\n",nRet);
    return nRet;
}
```

When the recognizer session is established, the response message sent from the MRCP server identifies the MRCP server sockets to which the collected audio must be streamed for recognition. The MRCP client library parses the MRCP server response for the RTP port and stores it for use throughout the session.

MrcpDefineGrammar

This method identifies the grammar to be loaded from a web server and compiled, as well as the universal resource identifier (URI) that identifies that grammar for use.

```
string sContType = " text/uri-list";
string sContId = " com.Dialogic.mrcp.gram-1";
string sGrammar = "http://192.168.100.124/location.grxml";

nRet = m_pMrcpClient->MrcpDefineGrammar(sContType, sContId, sGrammar);
if (200 != nRet)
{
      printf("<—————MRCP define grammar ERROR = %d\n",nRet);
      return nRet;
}
```

In the sample shown, `sContType` identifies the format of the grammar file stored on the server. A value of "text/uri-list" indicates that the format of the grammar is extensible markup language (XML)-based, either a grammar XML (GRXML) or other XML format file.

The value contained in `sContId` is the URI that links the compiled grammar to the recognition procedure when called. If specified, the URI is stored and used in the MrcpRecognizeSpeech method.

Finally, `sGrammar` contains the http address of the grammar file to load and compile.

Grammar can also be dynamically defined by passing the grammar definition to the MRCP server as a parameter in the MrcpRecognizeSpeech method. This method of defining grammar is not recommended for large grammars due to the delay it may add to the recognition process.

MrcpRecognizeSpeech

This method sets up the parameters for a recognition session with the MRCP server. The setup parameters are contained within the MrcpRecognize class instantiated in the following sample. If a call to the MrcpDefineGrammar method has been made and a URI specified, that URI will be used in the recognition process. The default value is com.Dialogic.mrcp.gram-1.

```
typedef std::list<std::string> RecognizeBuffers;

MrcpRecognize *mrcpRec = new MrcpRecognize();
nRet = m_pMrcpClient->MrcpRecognizeSpeech(*mrcpRec, &m_recognizeBuffs);
if (200 != nRet)
{
      printf("<———-MRCP recognize ERROR = %d\n",nRet);
      return nRet;
}
nRet = m_pMrcpClient->MrcpStartRecognizeTimer();
if (200 != nRet)
{
   printf("<————MRCP start recognize timer error = %d\n",nRet);
   return nRet;
}
```

The MrcpStartRecognizeTimer method must be called as soon as the MRCP client application wants to activate the recognition engine on the MRCP server.

Although all MRCP commands are asynchronous, most complete in a near synchronous time frame. One exception is the MrcpRecognizeSpeech command. MRCP ASR does not begin to process received audio until the MRCP recognition start timers command (supported with the MrcpStartRecognizeTimer method) is given. Once recognition is underway, the MRCP server will return a "proceeding" message followed by a "recognition-complete" message. An STL list of strings container, constructed by the client application, is used to send received audio to the MRCP client library for processing to send to the MRCP server. Completion events are sent from the MRCP server. The class library handles the events and provides a method to access these events.

The Dialogic® application programming interface (API), **ec_stream**, is used to get the caller's utterances. For each buffer generated, the MRCP client application should use the GetListLock method to lock the STL list of strings container, add the buffer to the list, and release the lock using the UnlockList method.

```
m_xpbRec.wFileFormat = FILE_FORMAT_VOX;
m_xpbRec.wDataFormat = DATA_FORMAT_MULAW;
m_xpbRec.nSamplesPerSec = DRT_8KHZ;
m_xpbRec.wBitsPerSample = 8;

if (ec_stream(m_handle, 0, &m_xpbRec, &CVoiceDevice::csp_write,
EV_ASYNC)==-1)
{
   printf(     "ec_stream(%s) error:%s\n",
               m_name, ATDV_ERRMSGP(m_handle));
         return -1;
}
```

```
Sleep(5000);
if (ec_stopch(m_handle, FULLDUPLEX, EV_ASYNC) == -1)
{
printf("ec_stopch)%s) error:%s\n", m_name,
                ATDV_ERRMSGP(m_handle));
         return -1;
}
```

Although the sample MRCP client application available for download with this application note is synchronous, the following code snippet from that application could be run in a thread to maintain an MRCP state machine.

```
while (true)
{
         res = m_pMrcpClient->GetRequestStatus("recognizer",nRet,false);
         if (res.find("RECOGNITION-COMPLETE")!= string::npos)
         {
               m_pMrcpClient->ShutdownAsrLoop(true);
               break;
         }
}
```

GetRequestStatus will return each time a status message for the recognition request is received. The ShutdownAsrLoop method must be called to close the MRCP client library thread that's running an infinite loop to package caller speech for the MRCP server.

Although not used in the accompanying sample application, any events received by the event processor such as dual tone multifunction (DTMF; touchtone) or other forms of barge-in may require the client application to send a recognize stop command (MrcpRecognizeStop) to the MRCP server.

The GetRecognitionValue method can be called to parse the recognition-complete message received from the MRCP server. It returns the value recognized from the caller's utterances based on the grammar defined.

TearDownMrcpSession

This method should be called as soon as the requested audio function has completed or has been interrupted. Prompt calling of teardown will

- Reduce the risk of creating "orphaned" sessions on the MRCP server
- Reduce the license and server resource usage

```
nRet = m_pMrcpClient->TearDownMrcpSession("recognizer");
if (200 != nRet)
{
      printf ("teardown recognizer ERROR = %d\n",nRet);
      return nRet;
}
```

Figure 2 shows the state transitions that are possible within an MRCP-based solution for ASR processing.

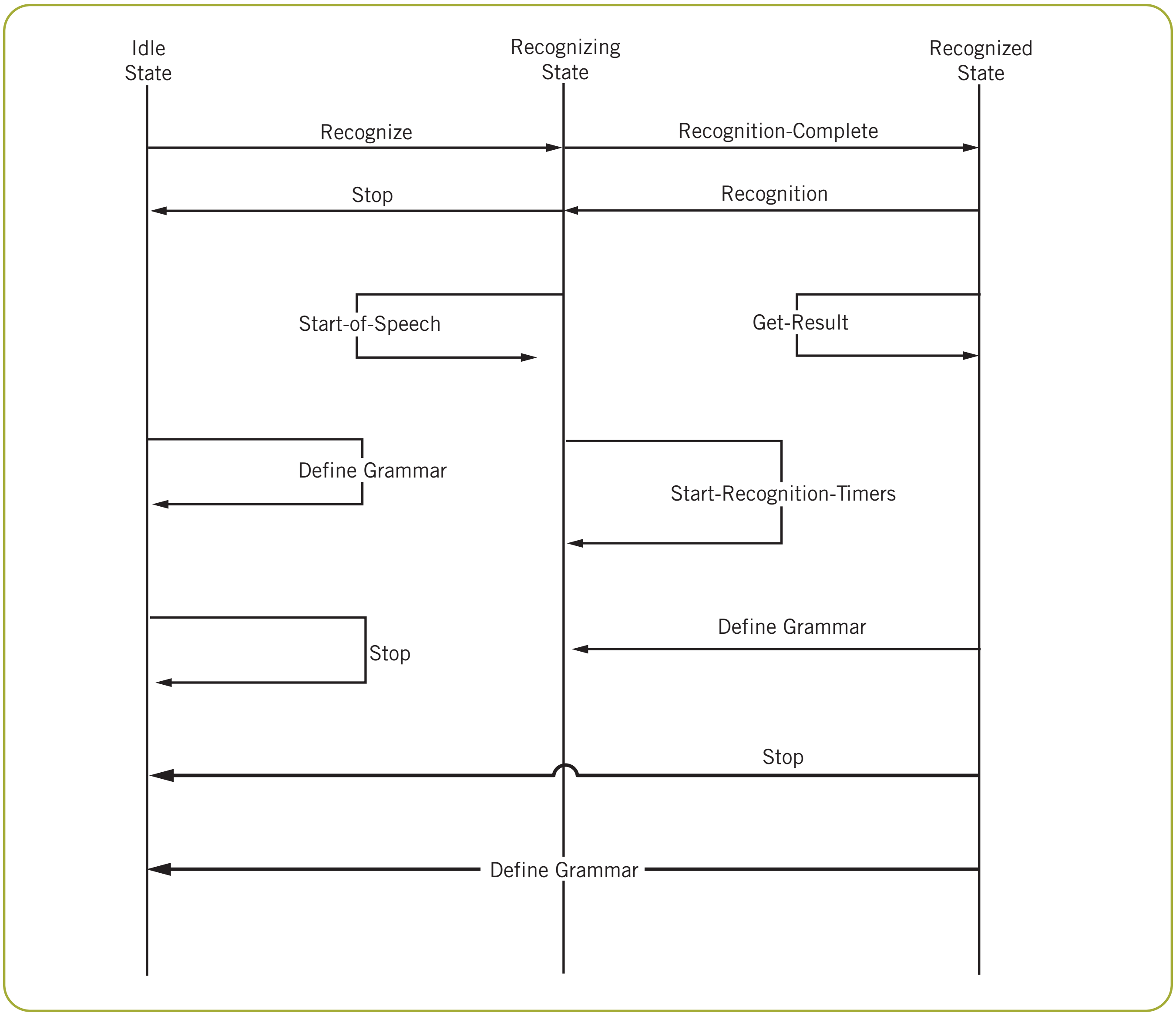

Figure 2. MRCP Recognition State Transitions

Generally, an MRCP state machine is implemented within the MRCP client application.

Using Text-to-Speech and File Streaming

The MRCP architecture defines how all pre-recorded audio played during the course of a call can be streamed from a web server to the MRCP client application. Hosting the audio files used in an MRCP application on a central web server provides the capability for the same audio to be played to both callers and web users. That is, the same audio could be played to users accessing a company's web site as is played to callers calling the company.

TTS support is similar to the streaming play of a file. Audio is generated on the MRCP server and is streamed to the MRCP client application. A common use of TTS is to speak a customer's input back to them for confirmation, or to play dynamically generated information retrieved from a database. MRCP client library methods must be called in this order to provide either streaming file play or TTS:

- InitializeMrcpSession
- MrcpSpeak
- TearDownMrcpSession

All code samples shown are from the reference application that is available for download with this application note (see the "For More Information" section).

InitializeMrcpSession

This method initializes the MRCP session with the MRCP server for a synthesizer session.

```
string mrcpClient = m_ipParms.mrcpIpClient;
                  //this is the client address of the Dialogic comms server
string mrcpPort = m_ipParms.mrcpPort;

nRet = m_pMrcpClient->InitializeMrcpSession(“synthesizer”,mrcpClient,mrcpPort);
if (200 != nRet)
{
   printf(“<——-MRCP initialization ERROR = %d\n”,nRet);
   return nRet;
}
```

The code snippet above is very similar to the use of the InitializeMrcpSession defined within the “Using Automatic Speech Recognition” section of this document. One difference from an MRCP client perspective is that the MRCP command provides the MRCP server with the socket number that it will listen to in order to receive the audio.

MrcpSpeak

This method both initiates a streaming file play from an MRCP server and triggers a TTS audio stream from the MRCP server.

In the following code, the TTS capabilities of the MRCP server are invoked to speak the phrase, “MRCP text to speech sample”.

```
string s_ttsAudioSample = “MRCP Text to speech sample”;
nRet = m_pMrcpClient->MrcpSpeak(s_ttsAudioSample, &m_audioBuffs);
if (MIN_MRCP_SEQUENCE > nRet)
{
   printf(“<————MRCP speak ERROR %d\n”,nRet);
   return nRet;
}
```

The following code provides an example of using the MrcpSpeak method to trigger a streaming file play.

```
string s_audioFile = “<audio
src=\”http://192.168.100.124/this_is_an_Example.ulaw\”
                           fetchtimeout=\”7s\” fetchint=\”prefetch\” />”;
nRet = m_pMrcpClient->MrcpSpeak(s_AudioContent, &m_audioBuffs);
if (MIN_MRCP_SEQUENCE > nRet)
{
   printf(“<————MRCP speak ERROR %d\n”,nRet);
   return nRet;
}
```

Note that although the two code examples use the same method, the audio parameter for a streaming file play is the URL of the file to be played.

Although all MRCP commands are asynchronous, most complete in a near synchronous time frame. One exception is the MrcpSpeak command. Once the speak request has been received and processed, the MRCP server will return a “proceeding” message followed by a “speak-complete” message. An STL list of strings container, constructed by the client application, is used to collect audio received from the MRCP server. The MRCP client library removes the RTP headers before adding the audio buffer to the STL container.

Dialogic's voice APIs, **dx_setuio** and **dx_playiottdata**, are used to present the MRCP-supplied audio to the caller. For each buffer retrieved from the STL list, the MRCP client application should use the GetListLock method to lock the STL list of strings container, extract and remove the buffer from the list, and release the lock using the UnlockList method. GetListLock and UnlockList make use of mutexes to prevent contention when working with the STL containers.

```
            // set the UIO callbacks
      m_uioblk.u_read = CVoiceDevice::uio_read;
      m_uioblk.u_write = CVoiceDevice::uio_write;
      m_uioblk.u_seek = CVoiceDevice::uio_seek;

      if (dx_setuio(m_uioblk) == -1)
      {
            printf("error setting uio for streaming \n");
      }
   .
   .
      if (dx_playiottdata(m_handle, &m_iott, 0, &m_xpb, EV_ASYNC)==-1)
      {
            printf("dx_playiottdata error:%s\n", ATDV_ERRMSGP(m_handle));
            return -1;
      }
```

Although the sample MRCP client application available for download with this application note is synchronous, the following code snippet from that application could be run in a thread to maintain an MRCP state machine.

```
      while (true)
      {
             res = m_pMrcpClient->GetRequestStatus("synthesizer",nRet,false);
             if (res.find("SPEAK-COMPLETE")!= string::npos)
            {
                  m_pMrcpClient->ShutdownTtsLoop(true);
                  break;
            }
      }
```

GetRequestStatus will return each time a status message for the MrcpSpeak request is received. The ShutdownTtsLoop method must be called to close the MRCP client library thread running in an infinite loop to receive audio from the MRCP server.

Although not used in the accompanying sample application, any events received by the event processor, such as DTMF or other forms of barge-in, may require the client application to send an MrcpSpeakStop or an MrcpSpeakBargeIn command to the MRCP server.

In a more complete MRCP implementation, MrcpSpeakStop would be used in cases where the audio could be resumed while MrcpSpeakBargeIn would be used to halt the audio stream.

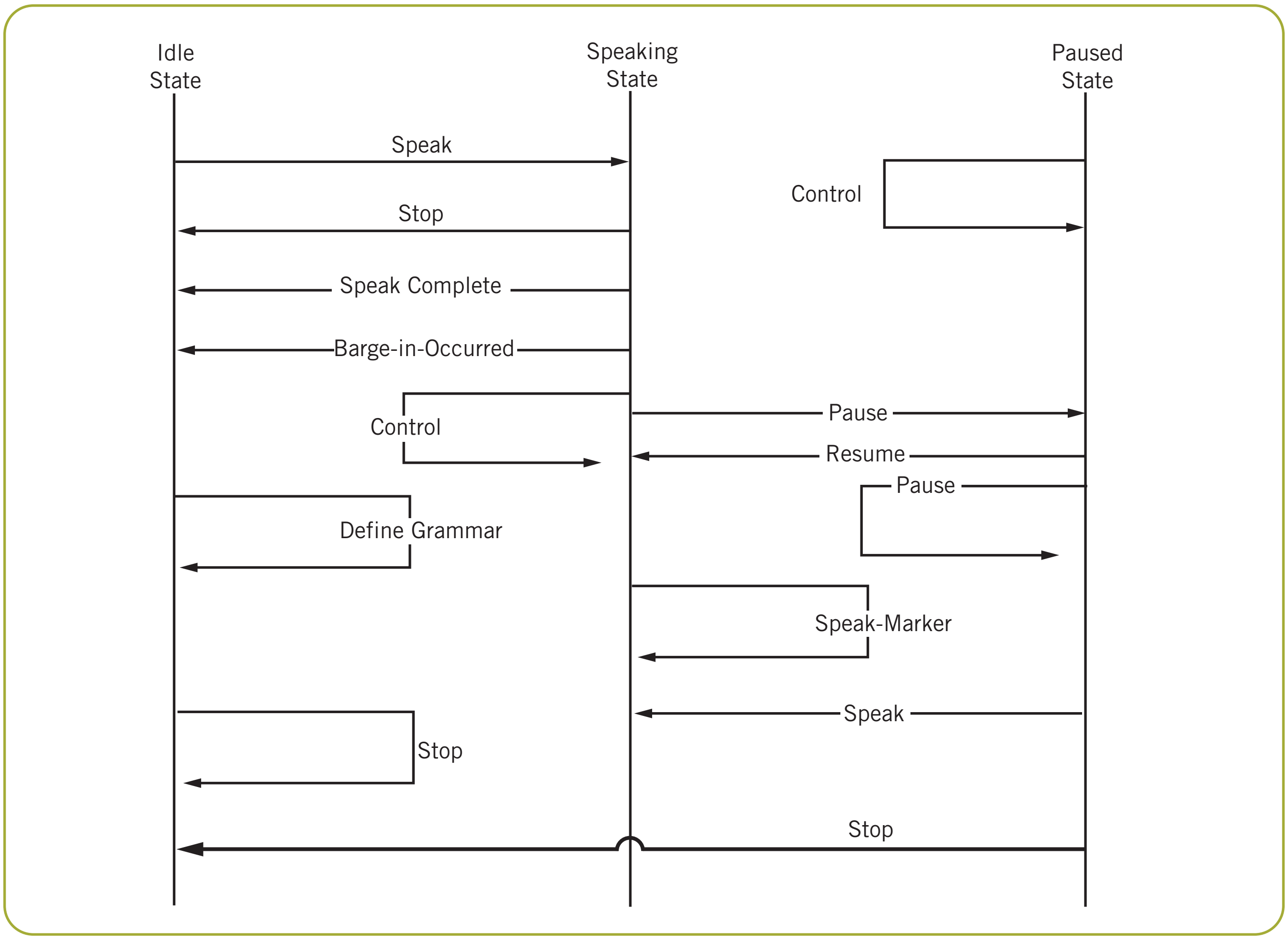

Figure 3. TTS State Transition Diagram

TearDownMrcpSession

This method should be called as soon as the requested audio function has completed or has been interrupted. Prompt calling of teardown will

- Reduce the risk of creating "orphaned" sessions on the MRCP server
- Reduce the license and server resource usage

```
    nRet = m_pMrcpClient->TearDownMrcpSession("synthesizer");
    if (200 != nRet)
    {
        printf ("teardown recognizer ERROR = %d\n",nRet);
        return nRet;
    }
```

Figure 3 shows the state transitions that are possible within an MRCP-based solution for TTS processing.

The implementation of a MRCP state machine within the MRCP client application is strongly recommended.

Summary

MRCP is a rapidly growing standard that is gaining wide support in the speech and telecommunications markets of today. This application note provides an overview and a starting point for developing MRCP client applications utilizing Dialogic® communications products. Combining the information in this application note with the information contained in the *Media Resource Control Protocol (MRCP) Client Library User's Guide* helps define a migration path to MRCP-based applications for legacy telecommunications applications with Dialogic® components.

Acronyms

API	Application Programming Interface
ASR	Automatic Speech Recognition
DTMF	Dual Tone Multifunction
IETF	Internet Engineering Task Force
GRXML	Grammar XML
MRCP	Media Resource Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SIP	Session Initiation Protocol
STL	Standard Template Library
TTS	Text-To-Speech
URI	Universal Resource Identifier
URL	Universal Resource Locator
XML	eXtensible Markup Language

For More Information

Internet Engineering Task Force; Internet-Draft; draft-shanmugham-mrcp-01 MRCP: Media Resource Control Protocol

A Zip file containing the sample application for this application note can be downloaded at http://www.dialogic.com/goto/?10666

MRCP V1 Client Library User's Guide — http://www.dialogic.com/goto/?9603

A Zip file, *MRCP V1 Client Library,* containing the source code for the MRCP V1 Client Library can be downloaded at http://www.dialogic.com/goto/?10568

MRCP V2 Client Library User's Guide — http://www.dialogic.com/goto/?10285

A Zip file, *MRCP V2 Client Library,* containing the source code for the MRCP V2 Client Library be downloaded at http://sourceforge.net/projects/openmrcpclient

To learn more, visit our site on the World Wide Web at **http://www.dialogic.com**

Dialogic Corporation
9800 Cavendish Blvd., 5th floor
Montreal, Quebec
CANADA H4M 2V9