A Very Simple SIP User Agent

Sun Gang, gangs@kth.se

1 Introduction

This assignment is to design and implement a SIP User Agent namely SIP Speaker which can be regarded as a robot answering machine. The SIP waits for incoming calls and answers then when received. When the call is answered, the SIP Speaker will play a sound. After that, the SIP Speaker will terminate the call. And a web server is integrated too. This server is used to manage answering message that SIP Speaker players.

2 Installation

Set the CLASSPATH to include the lib files of JMF and Freetts.

1.Compile:

Make sure the directory structure ik2213/assignment2 is maintained, then run "javac ik2213/assignment2/*.java" to compile all the java code.

2.Run & Configure:

Make sure the configuration file is in current working directory, then start the SIPSpeaker as "java ik2213/assignment2/SIPSpeaker [-c config_file_name] [-user sip_uri] [-http http_bind_address]" To stop the server, go to URL "http://http_bind_address" and press "close".

3.Use:

Dial "sip_uri", the call will be established and the audio message will be played. The call will be terminated when audio message ends or user hands up the phone. To manage answering message, go to URL "http://http_bind_address".

The program works fine with both SJphone and Ekiga on our own laptops, but when running on development server (studdev.ssvl.kth.se) throught ssh -X (Enable X11 forwarding), the server gives warning: "libxcb: WARNING! Program tries to unlock a connection without having acquired a lock first, which indicates a programming error". SJphone does not play any sound in this situation, but Ekiga still works fine.

3 Probems and Solutions

Configuration management:

The main class **SIPSpeaker** is responsible to manage configuration. Configuration parameters include: sip user name, sip interface address, sip port number, http interface address, http portnumber, default message text, default audio file name and current audio file name. During application startup, **SIPSpeaker** will first parse command-line arguments, then try to load and set the configurations. All command-line arguments (including configuration file name) are optional. Parameters specified in command-line arguments have higher priority than those in configuration file, if some parameter is missing in both command-line and configuration file, default value will be used. During application shutdown, all parameters will be saved back into the configuration file.

HTTP server & FreeTTS:

After get the configuration properties, server will check that if the default WAV file exists or not. If it doesn't exist, server will create the default WAV file using FreeTTS library. And then HTTP server will start. Server listens to the incoming request on the port which defined in configuration file. By checking the HTTP request head, server decides that it should response the administration page for GET request or creates a new **HandlePost** class for POST request.**HandlePost** is used for handle POST request including extracting the variable's value and doing process corresponding the request. If the value of variable "button" is "Close", server will invoke the shutdown method in class **SIPSpeaker**. If the value of variable "button" is "Delete" or the speak content that user want to use is empty, server will delete the user defined WAV file, and use the default WAV file. If the value of variable "button" is "Change", server will create a new user defined WAV file or cover the old one by using FreeTTS. FreeTTS is a speech synthesis system written entirely in the JavaTM

programming language. We modified the Helloworld demo, and made it suitable for our program. The voice of "kevin16" is used here, and the content of speech is extracted from the POST data. Finally, the text will be transformed to a new WAV file.

UDP transmission:

PacketManager is used to handle UDP transmission. It opens an UDP socket using specified sip address and port number. In order to support multiple users, **PacketManager** uses a HashMap to manage and store different call sessions. Remote socket (ip address and port number) is used as key while **CallHandler** object is value, different call sessions will be delivered to different **CallHandlers. PacketManager** implements Runnable interface, its main processing loop will try to receive an incoming UDP packet, retrieve the remote socket, and look up the HashMap to get the corresponding **CallHandler** to handle this packet. If it is a new call request, a new **CallHandler** will be created. After a call session closes, its key/value pair in HashMap will be removed. After new answering message is created, **PacketManager** will start using new audio message immediately and update the current audio message when there is no active call. If no current audio message is available, **PacketManager** will use the default audio message.

Call handling:

CallHandler is used to handle call sessions. When receiving an UDP packet, **CallHandler** will create a **SessionInformation** object to retrieve and store the SIP message, check the SIP information and make sure this message is valid and not duplicate, and then it will send out appropriate SIP & SDP response if needed. After a call session is setup, it will use **SessionInformation** to create an **AudioTransmitter** object, and start a new thread to transmit RTP audio stream. When the audio stream is fully transmitted or the remote user hands up, **CallHandler** will close the call session.

SIP & SDP parsing:

SessionInformation is used to parse and store SIP & SDP message, generate SIP & SDP response from request information. When parsing SDP request, it only retrieves the remote RTP socket information. In SDP response, **SessionInformation** specifies that SIPSpeaker will use Ulaw PCM coding and work in send-only mode. To create an **AudioTransmitter** object, **SessionInformation** will randomly select an available UDP port (>1024) for RTP transmission.

RTP transmission:

AudioTransmitter is responsible for RTP transmission. It imports the audio file for current message, encode the audio and prepare for transmission. It provides functions to controltransmission (start, stop, get duration time).

4 Conclusion

The SIPSpeaker application implements basic functions as a VoIP answering machine, it can handle multiple simultaneous calls, and it provides a web interface to manager answering messages.To make the application more practical, the following aspects can be addressed: (1) enhance the web interface to specify different messages for different caller; (2) support more RTP audio coding; (3) enable caller to leave an audio message; and (4) manage and store calling history.