M7017E Multimedia Systems

Lab 3 – IP answering machine (SIP)

By Baptiste Louis, Martin Bumba and Fisayo Sangogboye, December 2014.

Table of Contents

PROBLEM SPECIFICATION 4

Description of the problem and sub-problems 4

Declaration of assumption 4

Presentation of extra features implemented 5

FILES, FOLDERS AND DOCUMENTATION GUIDE 9

SYSTEMS DESCRIPTION 10

Internal design and structure of the application 10

PIPELINE DESCRIPTION 12

Transmitter pipeline 12

Receiver pipeline 12

PROBLEMS AND REFLECTIONS 13

Figures 16

References 17

INTRODUCTION

This report is about LAB3 of the course M7017E Multimedia Systems, proposed by Karl Anderson for Luleå University of Technology (LUT).

The objective of this Lab is to get more familiar with Session Initiation Protocol (SIP) and to continue to use the GStreamer Framework [1]. To achieve this, an IP answering machine has be developed.

The development team comprised of three international students. Baptiste Louis and Fisayo Sangogboye respectively from France and Nigeria are undergoing an Erasmus Mundus PERCCOM program. The third member, Martin Bumba is from Czech Republic and he is undergoing a master program in distributing systems and web development.

All program codes were written in JAVA using NetBeans IDE 8.0 on Windows 8.1 systems.

This project was a great experience.

NOTE

In software development, it is important to document your work. Also, it is a skill to know how to write so that other people can understand and potentially reuse your. This report follow the recommendation describe by Karl Anderson and Christer Wahlberg, LTU – Luleå (Sweden), Nov-14.

PROBLEM SPECIFICATION

Description of the problem and sub-problems

The expectation for this LAB is to implement an internet answering machine on which callers should be able to leave a message (voice mail) after a brief greeting. Then it should be possible for the user to playback, repeat and delete voice messages. GStreamer has to be used.

Figure 1 - Hardware fixe phone with integrating answering machine.

Coming from this requirement, we had to learn more about SPI protocol, understand at which level to implement it, what are the format of messages exchanged, how to open/close a session and how to implement it by coding.

We had to choose first the coding language to develop the application, an Integrated Development Environment (IDE) and its associated Operating System (OS).

Figure 2 - Netbeans IDE.

Figure 3 - GStreamer framework.

We had one week to realise, document and provide our solution in order to finish before to leave Sweden. Additionally, our background in application development and coding was different but we worked in team with the objective to learn more for each other even if this choice can affect our productivity for this project.

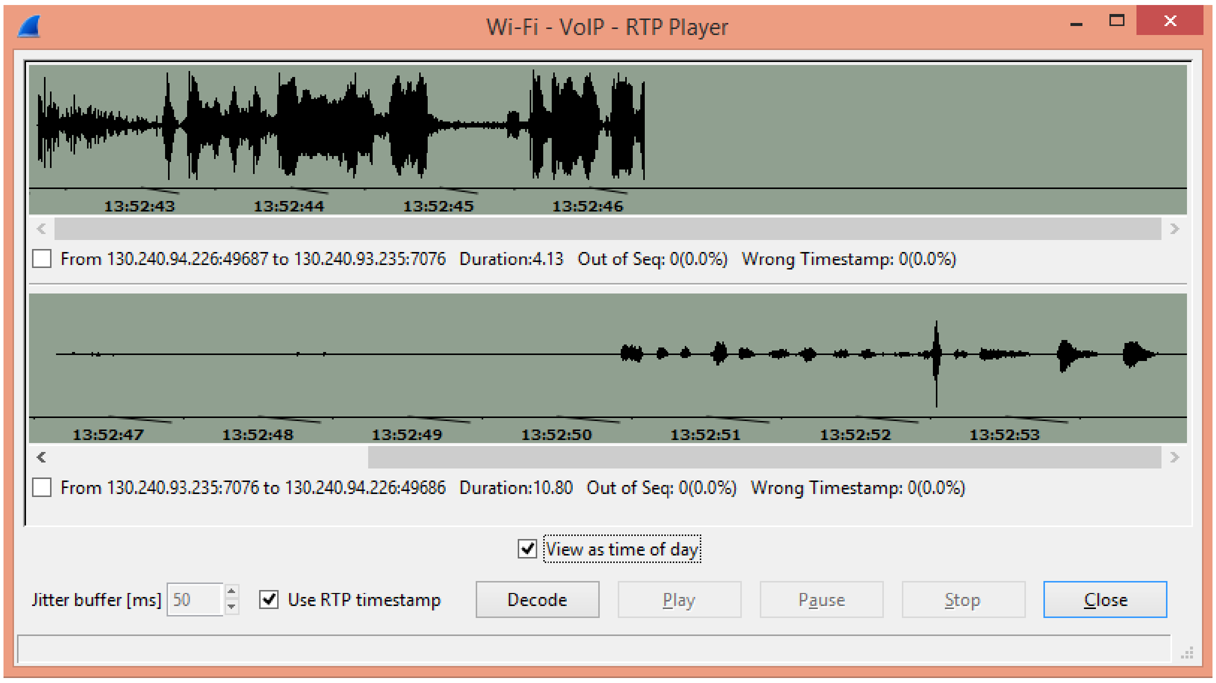
We are here to learn and it will be beneficial for our future.

Declaration of assumption

We assumed that an answering machine is a quite familiar tool even for a basic user. Nowadays, most of people has an auto-answering feature on the fixe phone and/or mobile phone. Therefore, we decided to skip some text button and implement some icon buttons in our interface. Also, we choose to implement the same basic and useful features than a user can expect for an answering machine.

Presentation of extra features implemented

Remember, the basic feature is to give the caller the possibility to leave a message (voicemail) after a brief greeting. The following capture is realized with Wireshark and show on the top the greeting and on the middle the message left by the caller.

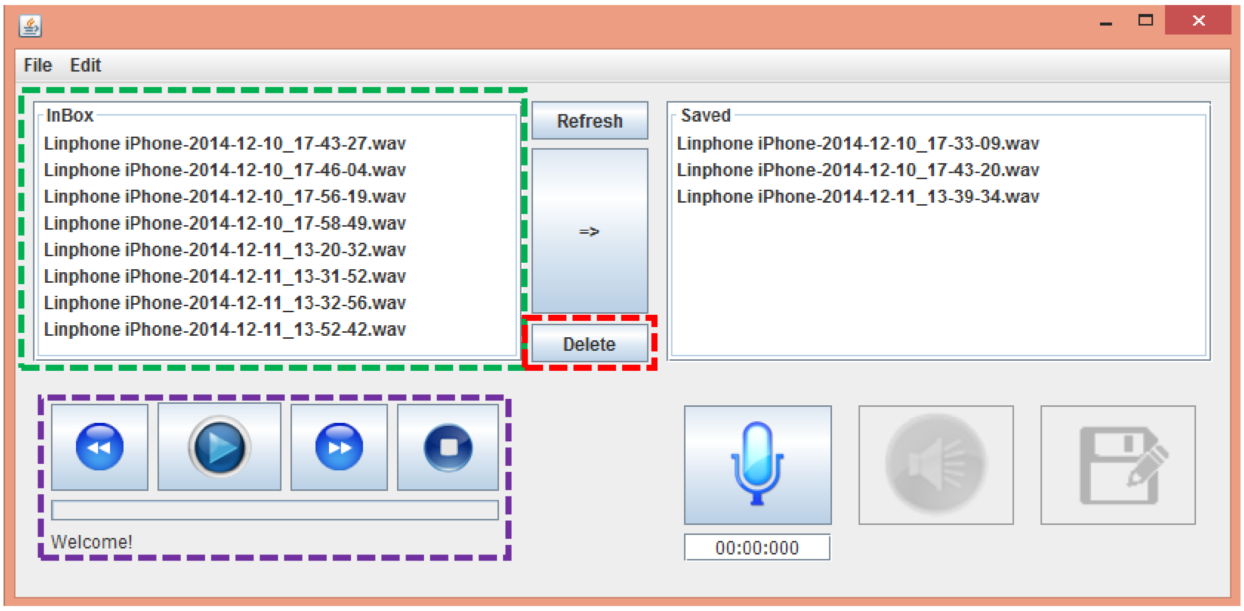


From 130.240.94.226:49687 to 130.240.93.235:7076 Duration:4.13 Out of Seq: 0(0.0%) Wrong Timestamp: 0(0.0%)

From 130.240.93.235:7076 to 130.240.94.226:49686 Duration:10.80 Out of Seq: 0(0.0%) Wrong Timestamp: 0(0.0%)

Figure 4 - Greeting followed by the caller voice message (wireshark).

Also, the user should be able to listen, playback and delete messages. We chose to develop a graphic interface where the user can quickly see if there is new messages (**InBox**), easily listen them (integrated GStreamer player) and if needed, simply delete them (**delete button**).



InBox
Linphone iPhone-2014-12-10_17-43-27.wav
Linphone iPhone-2014-12-10_17-46-04.wav
Linphone iPhone-2014-12-10_17-56-19.wav
Linphone iPhone-2014-12-10_17-58-49.wav
Linphone iPhone-2014-12-11_13-20-32.wav
Linphone iPhone-2014-12-11_13-31-52.wav
Linphone iPhone-2014-12-11_13-32-56.wav
Linphone iPhone-2014-12-11_13-52-42.wav

Saved
Linphone iPhone-2014-12-10_17-33-09.wav
Linphone iPhone-2014-12-10_17-43-20.wav
Linphone iPhone-2014-12-11_13-39-34.wav

Figure 5 – Graphic user interface of our answering machine.

Then, we thought that the following extra features improve the application and the user experience:

- We implement a login capability. The user have to enter a username and a password to access his messages.

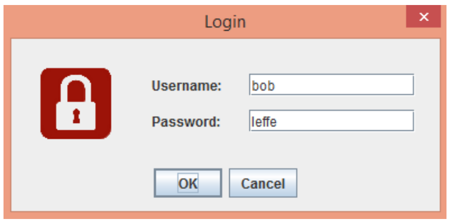
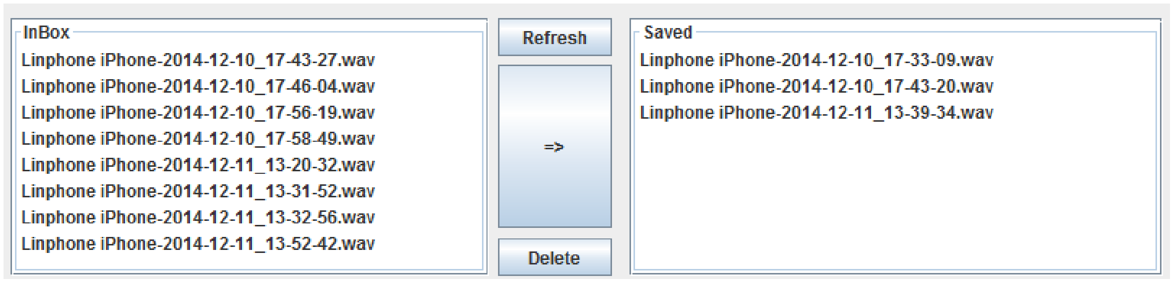


Figure 6 - Login window.

- We implement a Saved folder where the user can store some incoming messages during a time and then delete them. The refresh button should not be necessary because the application is automatically refresh after any changes in one of this two folders. However, from our discussions, we conclude that a user feel more comfortable to have the possibility to refresh himself the application.

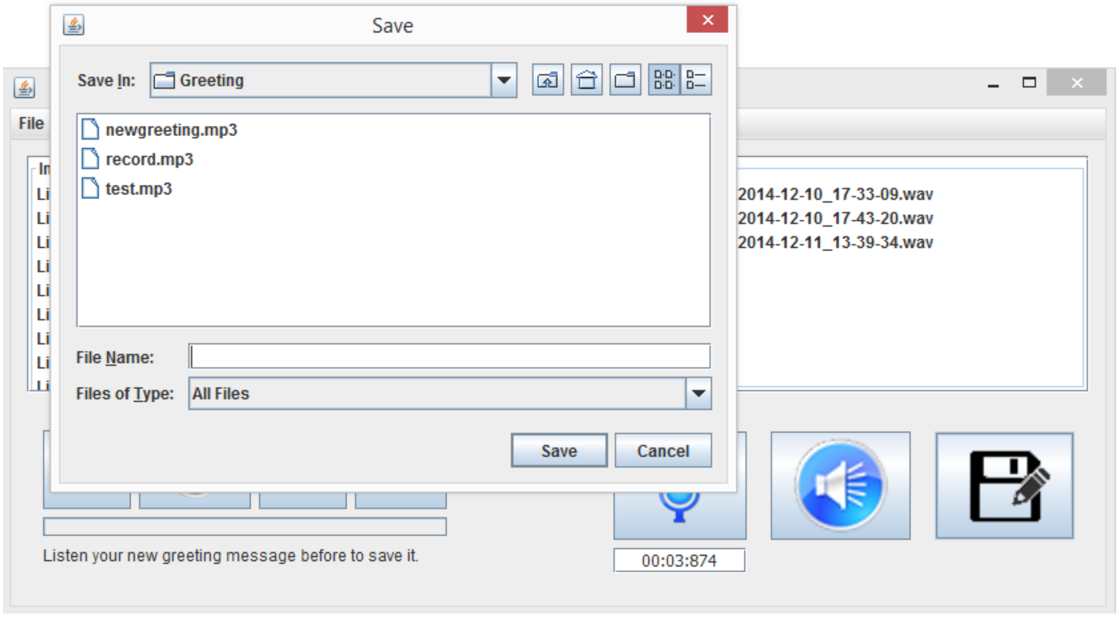


InBox
Linphone iPhone-2014-12-10_17-43-27.wav
Linphone iPhone-2014-12-10_17-46-04.wav
Linphone iPhone-2014-12-10_17-56-19.wav
Linphone iPhone-2014-12-10_17-58-49.wav
Linphone iPhone-2014-12-11_13-20-32.wav
Linphone iPhone-2014-12-11_13-31-52.wav
Linphone iPhone-2014-12-11_13-32-56.wav
Linphone iPhone-2014-12-11_13-52-42.wav

Saved
Linphone iPhone-2014-12-10_17-33-09.wav
Linphone iPhone-2014-12-10_17-43-20.wav
Linphone iPhone-2014-12-11_13-39-34.wav

Figure 7 - Saved folder to store so voice messages.

- We implement reuse our audio recorder (LAB1) to give the user the possibility to record a new greeting message and to save it in the greeting folder of the application. The SAVE_AS window opens automatically in the Greeting folder. The new greeting is automatically defined as the new greeting for future incoming calls.



newgreeting.mp3
record.mp3
test.mp3

File Name:
Files of Type: All Files

2014-12-10_17-33-09.wav
2014-12-10_17-43-20.wav
2014-12-11_13-39-34.wav

Listen your new greeting message before to save it.

00:03:874

Figure 8 - Integrated recorder to record a new greeting.

- We give the user the possibility to choose at any moment a new greeting from the Greeting folder of the application. This option is available from the **edit** menu bar.

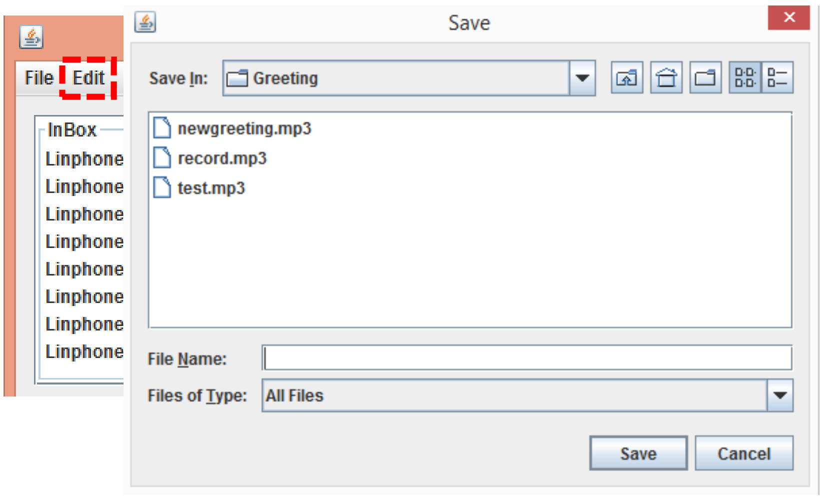
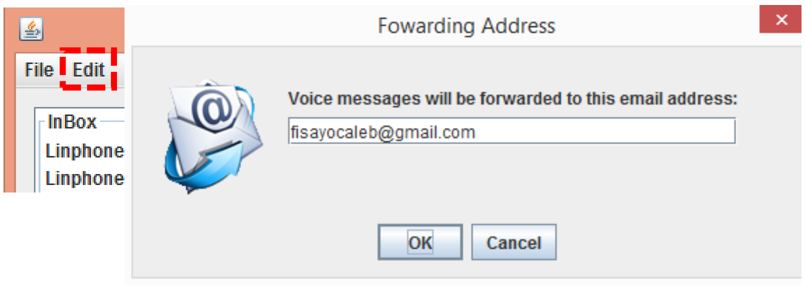


Figure 9 - File chooser to choose a new greeting.

- We implement an email notification capability. When the user receive a new voice message, the user receive an email with the voice message. The user have the possibility to change this forwarding email from the GUI. This option is available from the edit menu bar.

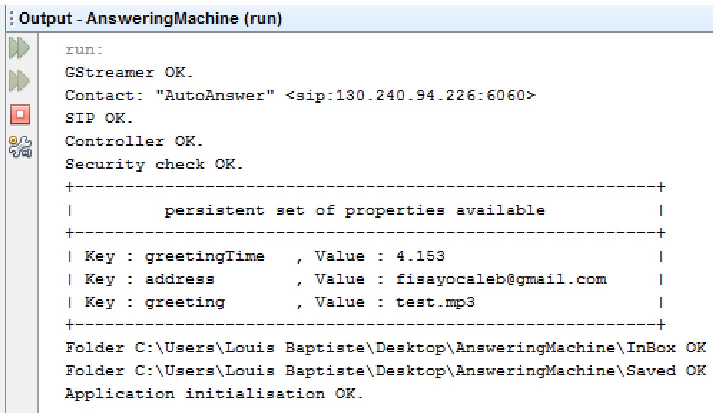


Voice messages will be forwarded to this email address:

fisayocaleb@gmail.com

Figure 10 - Window to change the forwarding email address.

- We implement a clear and detailed console output for debugging purpose but also to follow and understand the program. This output is important to check for example our persistent properties implementation to store some information (like the email address) even after restarting the application.

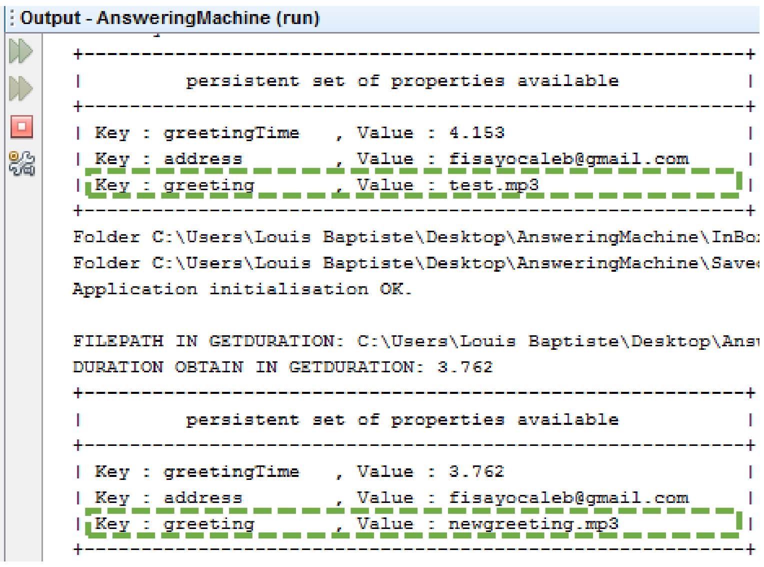


Output - AnsweringMachine (run)

```
run:
GStreamer OK.
Contact: "AutoAnswer" <sip:130.240.94.226:6060>
SIP OK.
Controller OK.
Security check OK.
+-----------------------------------------------------------+
|         persistent set of properties available            |
+-----------------------------------------------------------+
| Key : greetingTime    , Value : 4.153                     |
| Key : address         , Value : fisayocaleb@gmail.com     |
| Key : greeting        , Value : test.mp3                  |
+-----------------------------------------------------------+
Folder C:\Users\Louis Baptiste\Desktop\AnsweringMachine\InBox OK
Folder C:\Users\Louis Baptiste\Desktop\AnsweringMachine\Saved OK
Application initialisation OK.
```

Figure 11 - Output console for the application initialisation.

Also, the console output permit to see that properties are changed **correctly** after a specific manipulation like changing the greeting message.


Output - AnsweringMachine (run)

```
+----------------------------------------------------------+
|          persistent set of properties available          |
+----------------------------------------------------------+
| Key : greetingTime    , Value : 4.153                    |
| Key : address         , Value : fisayocaleb@gmail.com    |
| Key : greeting        , Value : test.mp3                 |
+----------------------------------------------------------+
Folder C:\Users\Louis Baptiste\Desktop\AnsweringMachine\InBo
Folder C:\Users\Louis Baptiste\Desktop\AnsweringMachine\Save
Application initialisation OK.

FILEPATH IN GETDURATION: C:\Users\Louis Baptiste\Desktop\Ans
DURATION OBTAIN IN GETDURATION: 3.762
+----------------------------------------------------------+
|          persistent set of properties available          |
+----------------------------------------------------------+
| Key : greetingTime    , Value : 3.762                    |
| Key : address         , Value : fisayocaleb@gmail.com    |
| Key : greeting        , Value : newgreeting.mp3          |
+----------------------------------------------------------+
```

Figure 12 - The new greeting path have been registered in the properties.

All the code is available on GitHub https://github.com/Udacity2048/M7017E_LAB3 for open purpose. A .jar file is provided but an error occurs during the execution on windows ("gstreamer not found") despite all libraries are linked with the jar file – new: jdk version can be responsible. Instead, please use Netbeans to open our application for the moment.
Do not hesitate to contact us at baplou-4@student.ltu.se .

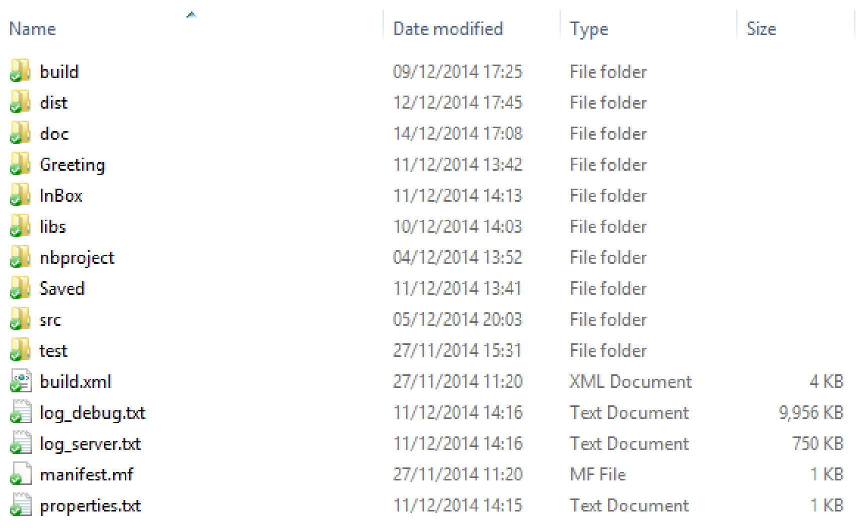
FILES, FOLDERS AND DOCUMENTATION GUIDE

Remember, our application has been developed in JAVA on Windows 8.1 with NetBeans IDE 8.0.

Like the previous LAB, we were advised to develop on Ubuntu machines to avoid some issues. However, Windows is most used so we wanted to make our application working on this OS.

Also, we were advised to develop in C++ instead of JAVA. However, our skills in JAVA are much more developed than for C++. In the same time, we were able to reuse some code parts of LAB1 and Lab2.

This section present folders & files present in our application folder.



Name	Date modified	Type	Size
build	09/12/2014 17:25	File folder	
dist	12/12/2014 17:45	File folder	
doc	14/12/2014 17:08	File folder	
Greeting	11/12/2014 13:42	File folder	
InBox	11/12/2014 14:13	File folder	
libs	10/12/2014 14:03	File folder	
nbproject	04/12/2014 13:52	File folder	
Saved	11/12/2014 13:41	File folder	
src	05/12/2014 20:03	File folder	
test	27/11/2014 15:31	File folder	
build.xml	27/11/2014 11:20	XML Document	4 KB
log_debug.txt	11/12/2014 14:16	Text Document	9,956 KB
log_server.txt	11/12/2014 14:16	Text Document	750 KB
manifest.mf	27/11/2014 11:20	MF File	1 KB
properties.txt	11/12/2014 14:15	Text Document	1 KB

Figure 13 - Structure of the application folder

Application folder contains following folders:

- *build:* contains compiled files;
- *dist:* contains the JAR file and all the java documentation files;
- *doc:* contains documentation files, i.e. a JavaDoc shortcut, GUI screenshot and the report ;
- *Greeting:* contains greeting media files;
- *InBox:* contains new incoming voice messages;
- *lilbs:* contains different libraries used;
- *nbproject:* contains the project's metadata;
- *Saved:* contains saved voice messages;
- *src:* contains the source code and images used for the GUI;
- *test:* contains eventually some Unit Tests for the project.

Properties files:

- *log_debug.txt:* logs for debugging the application;
- *log_server.txt:* logs to understand SIP communications;
- *properties.txt:* persistent properties used by the application.

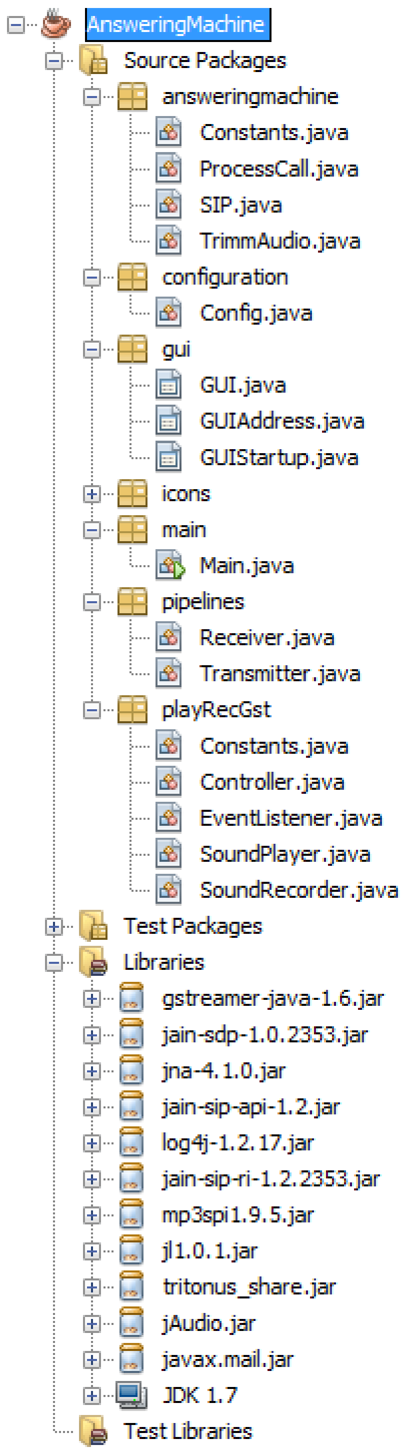
Documentation files:

- *M7017E_LAB3report.pdf:* report of the whole audio conference tool Lab2 project;
- *Javadoc_AnsweringMachine.lnk:* shortcut to Javadoc created for the project;
- *GUI.PNG*: screenshot of the application.

SYSTEMS DESCRIPTION

Internal design and structure of the application

The application is composed of 7 packages and 16 java classes. This architectural clearly define the answering machine core application part, the graphic user interface part, the configuration properties part, the underlay pipeline part and the player&recorder package from LAB1.



```
AnsweringMachine
    Source Packages
        answeringmachine
            Constants.java
            ProcessCall.java
            SIP.java
            TrimmAudio.java
        configuration
            Config.java
        gui
            GUI.java
            GUIAddress.java
            GUIStartup.java
        icons
        main
            Main.java
        pipelines
            Receiver.java
            Transmitter.java
        playRecGst
            Constants.java
            Controller.java
            EventListener.java
            SoundPlayer.java
            SoundRecorder.java
    Test Packages
    Libraries
        gstreamer-java-1.6.jar
        jain-sdp-1.0.2353.jar
        jna-4.1.0.jar
        jain-sip-api-1.2.jar
        log4j-1.2.17.jar
        jain-sip-ri-1.2.2353.jar
        mp3spi1.9.5.jar
        jl1.0.1.jar
        tritonus_share.jar
        jAudio.jar
        javax.mail.jar
        JDK 1.7
    Test Libraries
```

Figure 14 – Classes and packages hierarchy in IPansweringMachine project.

Here a short and clear description of each **packages**:

- *answeringmachine:* gather the core application;
- *configuration:* gather methods to manage persistent properties;
- *gui:* gather JFrames of our GUI;
- *icons:* gather icons used by our GUI;
- *main:* gather the main class;
- *pipelines:* gather the underlay pipelines of the answering machine;
- *playRctGst:* gather our LAB1 GStreamer Player and Recorder.

Here a short and clear description of each **classes**:

- *Constants:* gather constants used but the answering machine part;
- *ProcessCall:* methods which handle a call step by step;
- *SIP:* SIP protocol implementation;
- *TrimmAudio:* handle an issue explained in the section < PROBLEMS AND REFLECTIONS >;

- *Config:* gather methods to handle persistent properties;

- *GUI:* main gui of the application;
- *GUIAddress:* gui used to change the forwarding email address;
- *GUIStartup:* gui used for login;

- *Main:* gather all sub-initialization to launch the application;

- *Receiver:* pipeline to receive the new voice message;
- *Transmitter:* pipeline to transmit the greeting;

- *Constants:* gather constants used by classes in its package (LAB1);
- *Controller:* middleware for the Player and the Recorder (LAB1);
- *EventListener:* listener to handle player events (LAB1);
- *SoundPlayer:* audio player pipeline (LAB1);
- *SoundRecorder:* audio recorder pipeline (LAB1);

Here the purpose of different **libraries**:

- *Gstreamer-java-1.6.jar:* GStreamer;
- *Jain-sdp-1.0.2353.jar:* SIP;
- *Jna-4.1.0.jar:* GStreamer;
- *Jain-sip-api-1.2.jar:* SIP;
- *Log4j-1.2.17.jar:* SIP;
- *Jain-sip-ri-1.2.2353.jar:* SIP;
- *mp3spi1.9.5.jar:* getDuration() Mp3;
- *jl1.0.1.jar:* getDuration() Mp3;
- *tritonus_share.jar:* getDuration() MPp3;
- *jAudio.jar:* TrimmAudio() Wav;
- *javax.mail.jar:* mail notifications.

The details of each classes, their constructors, their methods, are available on the Javadoc provided with the project folder, under *doc* folder [2].

PIPELINE DESCRIPTION

Lab3 do not requires a deep understanding of the pipeline abstraction used in GStreamer [2]. However, SIP implementation was challenging. Past years implementations help us a lot to understand the logic.

Concerning pipelines, we reuse our player pipeline and our recorder pipeline from LAB1. Then we just need a transmitter pipeline to send the greeting and a receiver pipeline to receive the voice message of the caller.

These pipeline are not complex but the following section explain quickly how they work:

Transmitter pipeline

Transmitter pipeline transmit the greeting to the caller. So the first element is "filesrc" which permit to use our greeting files as the source. Then we decode the file with "mad" because our greeting are in "mp3" format. Then we prepare the audio with a convertor, resampler, encoder and we send it using an RTP/UDP layer.

Concerning configuration of these elements, you just need to define the filePath of the greeting source file (which is stored in our case in the persistent properties of the application) and you need to define ip and port of the caller for the udpSink element.

Receiver pipeline

The receiver pipeline is not more complex. It needs to receive the voice from the caller. So, the source is not a file but it is the network and the sink is not the network but a file. The location of the destination file is the root. Then, when the message is finished, we move the new voice message to the InBox folder of our application.

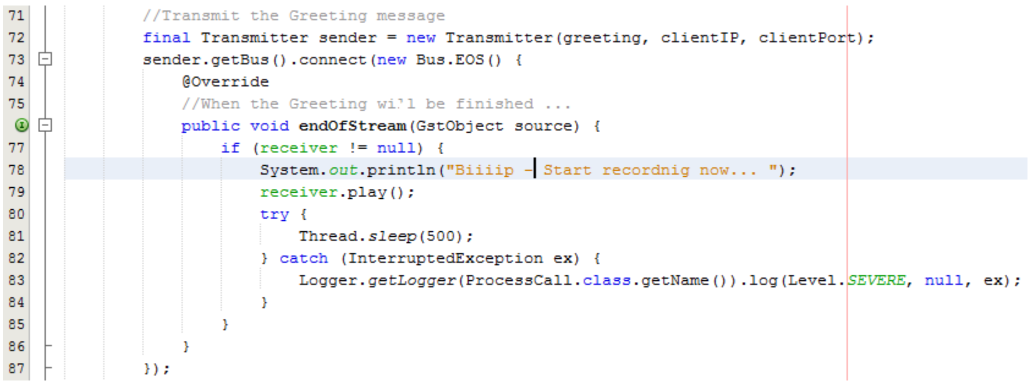
We are using the "wav" format with a "wavenc" encoder element. This format has been used in order to be able to trim the audio by the length of the greeting message. Our TrimmAudio solution was not working for "mp3" files.

PROBLEMS AND REFLECTIONS

After two LABs, we are quite familiar with **GStreamer.** However, the pipeline abstraction is still no at flexible and stable that we would imagine. Also, for this lab we had to focus on SIP and email notification. Thanks to [3] [4] [5] and our personal reflection, we understood the SIP implementation and we found solution to our problems.

The following section speak about some issues that we had. It can be useful for future work on this subject.

- We had again an interesting issue playing with pipeline. To explain it, let's consider a normal scenario for our answering machine: a caller call the user, he listens the greeting and then leave a message. Our issue is that the recorder start to record the voice at the same time that the greeting message start – it should start to record after the greeting.

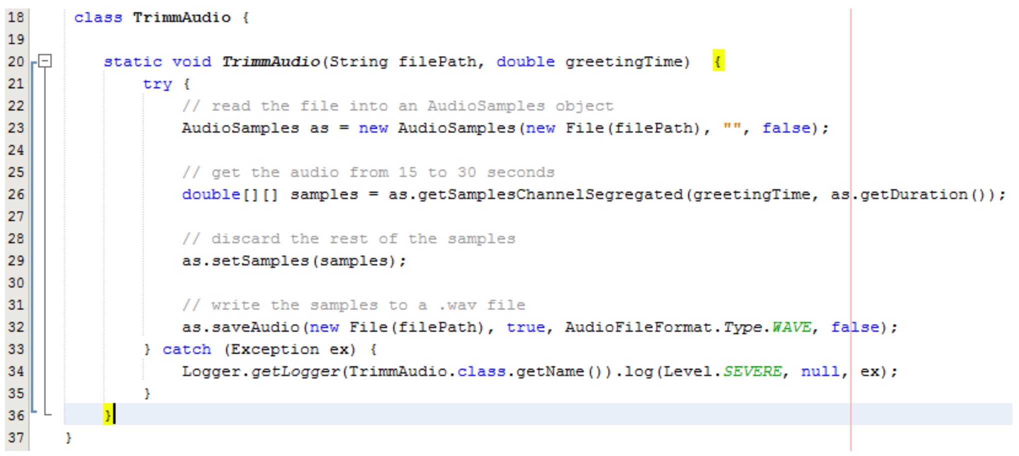


```
//Transmit the Greeting message
final Transmitter sender = new Transmitter(greeting, clientIP, clientPort);
sender.getBus().connect(new Bus.EOS() {
    @Override
    //When the Greeting will be finished ...
    public void endOfStream(GstObject source) {
        if (receiver != null) {
            System.out.println("Biiiip - Start recordnig now... ");
            receiver.play();
            try {
                Thread.sleep(500);
            } catch (InterruptedException ex) {
                Logger.getLogger(ProcessCall.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
});
```

Figure 15 - The recording should start after the greeting according to the code.

We suppose that there is a kind of transmission on the caller side which transmit voice during the greeting to the answering machine, and a kind of buffer in the pipeline we implement on the answering machine.

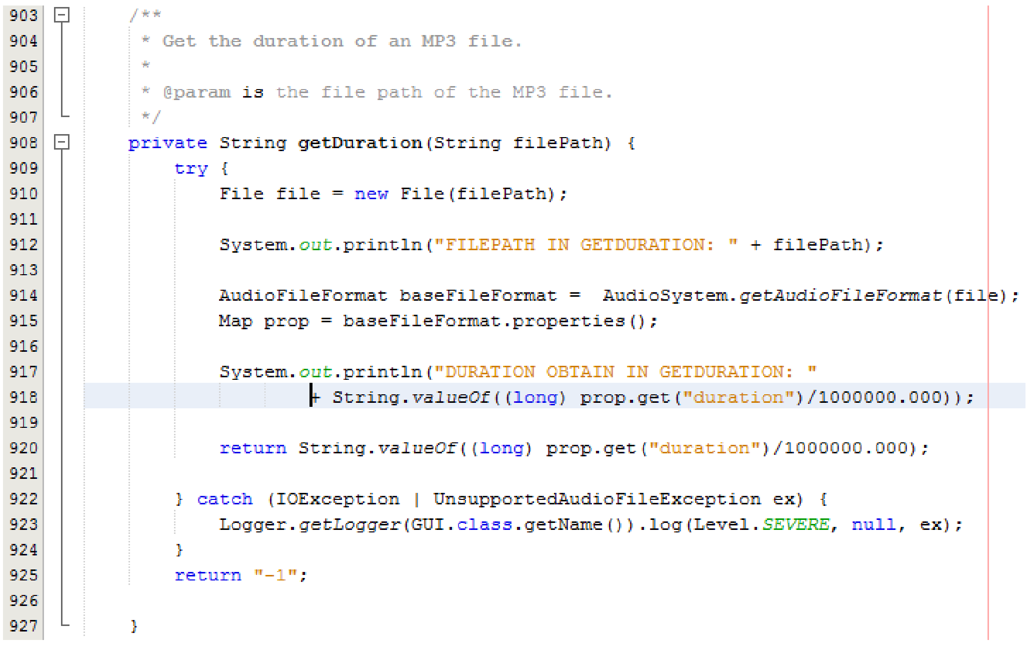
If you are in our situation (no-understanding of the problem) and you want to a solution anyway, you can reuse our TrimmAudio. This class cut the recorded voice message but the length of the greeting message. We are not proud of this "solution" but it is working well.



```
class TrimmAudio {

    static void TrimmAudio(String filePath, double greetingTime)  {
        try {
            // read the file into an AudioSamples object
            AudioSamples as = new AudioSamples(new File(filePath), "", false);

            // get the audio from 15 to 30 seconds
            double[][] samples = as.getSamplesChannelSegregated(greetingTime, as.getDuration());

            // discard the rest of the samples
            as.setSamples(samples);

            // write the samples to a .wav file
            as.saveAudio(new File(filePath), true, AudioFileFormat.Type.WAVE, false);
        } catch (Exception ex) {
            Logger.getLogger(TrimmAudio.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Figure 16 - TrimmAudio to handle the problem of recording starting.

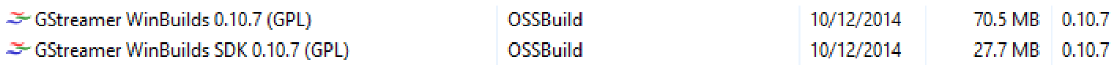
- Also, in order to cut the recorded message, you need to know the exact length of the greeting message. Our first greeting were saved in "wav" format but I appear that the transmitter was not working anymore. Consequently, we record greeting in "mp3" format and we get the duration of the greeting by using jAudio.jar library.



```java
/**
 * Get the duration of an MP3 file.
 *
 * @param is the file path of the MP3 file.
 */
private String getDuration(String filePath) {
    try {
        File file = new File(filePath);

        System.out.println("FILEPATH IN GETDURATION: " + filePath);

        AudioFileFormat baseFileFormat =  AudioSystem.getAudioFileFormat(file);
        Map prop = baseFileFormat.properties();

        System.out.println("DURATION OBTAIN IN GETDURATION: "
                + String.valueOf((long) prop.get("duration")/1000000.000));

        return String.valueOf((long) prop.get("duration")/1000000.000);

    } catch (IOException | UnsupportedAudioFileException ex) {
        Logger.getLogger(GUI.class.getName()).log(Level.SEVERE, null, ex);
    }
    return "-1";

}
```

Figure 17 - getDuration method used to get the length of our greetings.

- Now that we use "mp3" format in our recorder pipeline, we are using "mad" decoder from GStreamer. You can have some problem to access this GStreamer plugin on window. The OSS Build version 0.10.6 can eventually not work whereas the plugin mad is installed as you can see it during the installation process of OSS build. To solve this problem, download the beta version OSS Build 0.10.7 (GPL) [6].



GStreamer WinBuilds 0.10.7 (GPL)	OSSBuild	10/12/2014	70.5 MB	0.10.7
GStreamer WinBuilds SDK 0.10.7 (GPL)	OSSBuild	10/12/2014	27.7 MB	0.10.7

Figure 18 - GStreamer installation with OSS Build for Windows.

CONTRIBUTION

LAB3 project has been realized by three international students working as a team on all aspect of the project and each decisions were discussed and approved by every members, especially the initial reflexion on all different parts of our pipelines and the SIP protocol. However, we can identify some specific responsibilities for each person.

Baptiste Louis was mainly responsible for handling the project, commenting the code, developing the graphic user interface, and writing the documentations including this report.

Martin Bumba was mainly responsible for developing the core application code, the email notification and the SIP protocol.

Fisayo Sangogboye was mainly responsible for solving the issue of our recorder and implementing the audio trimmer.

ACKNOWLEDGEMENT

We, Baptiste LOUIS Martin BUMBA and Fisayo SANGOGBOYE, state that our project is compliant with the Academic ethics / Code of honour defined in the Study Guidance of the course M7017E Multimedia Systems, Luleå University of Technology, Semester1 Quarter2 2014.

Figures

FIGURE 1 - HARDWARE FIXE PHONE WITH INTEGRATING ANSWERING MACHINE. 4
FIGURE 2 - NETBEANS IDE. FIGURE 3 - GSTREAMER FRAMEWORK. 4
FIGURE 4 - GREETING FOLLOWED BY THE CALLER VOICE MESSAGE (WIRESHARK). 5
FIGURE 5 – GRAPHIC USER INTERFACE OF OUR ANSWERING MACHINE. 5
FIGURE 6 - LOGIN WINDOW. 6
FIGURE 7 - SAVED FOLDER TO STORE SO VOICE MESSAGES. 6
FIGURE 8 - INTEGRATED RECORDER TO RECORD A NEW GREETING. 6
FIGURE 9 - FILE CHOOSER TO CHOOSE A NEW GREETING. 7
FIGURE 10 - WINDOW TO CHANGE THE FORWARDING EMAIL ADDRESS. 7
FIGURE 11 - OUTPUT CONSOLE FOR THE APPLICATION INITIALISATION. 7
FIGURE 12 - THE NEW GREETING PATH HAVE BEEN REGISTERED IN THE PROPERTIES. 8
FIGURE 13 - STRUCTURE OF THE APPLICATION FOLDER 9
FIGURE 14 – CLASSES AND PACKAGES HIERARCHY IN IPANSWERINGMACHINE PROJECT. 10
FIGURE 16 - THE RECORDING SHOULD START AFTER THE GREETING ACCORDING TO THE CODE. 13
FIGURE 17 - TRIMMAUDIO TO HANDLE THE PROBLEM OF RECORDING STARTING. 13
FIGURE 18 - GETDURATION METHOD USED TO GET THE LENGTH OF OUR GREETINGS. 14
FIGURE 19 - GSTREAMER INSTALLATION WITH OSS BUILD FOR WINDOWS. 14

References

[1] gstreamer, "Home page," 8 November 2014. [Online]. Available: http://gstreamer.freedesktop.org/.

[2] B. Louis, “API documentation NetBeans,” private, Lulea, 2014.

[3] Wikipedia, “Session Initiation Protocol,” Wikipedia, 09 December 2014. [Online]. Available: http://en.wikipedia.org/wiki/Session_Initiation_Protocol. [Accessed 15 December 2014].

[4] C. Notin, "sip_voicemail," Github, December 2012. [Online]. Available: https://github.com/ClementNotin/sip_voicemail. [Accessed 15 December 2014].

[5] J. Skeet, “package javax.mail,” stackoverflow, 2013. [Online]. Available: http://stackoverflow.com/questions/6606529/package-javax-mail-and-javax-mail-internet-do-not-exist. [Accessed 15 December 2014].

[6] Google, “ossbuild,” google, [Online]. Available: https://code.google.com/p/ossbuild/. [Accessed 15 December 2014].