

pERCCOM
asmus undus

L
LULEÅ
TEKNISKA
UNIVERSITET



M7017E Multimedia Systems

Lab 2 – Media Distribution Over Internet

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.....	8
SYSTEMS DESCRIPTION	9
Internal design and structure of the application.....	9
PIPELINE DESCRIPTION	11
Sending pipeline	11
Receiving pipeline.....	11
PROBLEMS AND REFLECTIONS	13
Figures	15
References.....	16

INTRODUCTION

This report is about LAB2 of the course M7017E Multimedia Systems, proposed by Karl Anderson for Luleå University of Technology (LUT).

The objective of this Lab is sending and receiving media information on the network by using the GStreamer Framework [1]. To achieve this, an audio conference tool has been 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 distributed 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 follows the recommendation described 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 application with a graphic user interface that permit a communication between at least three simultaneous users. GStreamer has to be used.



Figure 1 – Audio conference in Adobe Connect.

Coming from this requirement, we had to learn more about network communication with GStreamer, understand how it works and how we can implement it. Also, we had to learn more about unicast and multicast communication over a network.

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 two weeks to realise, document and provide our solution in order to stay on schedule for following lab and the coming seminar work. 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 audio conference tool is a quite familiar tool even for a basic user. Nowadays, everybody had the opportunity to use an audio/video tool like msn, skype or Adobe Connect for professional or personal purpose. Therefore, we decided to skip some text button and implement some icon buttons in our interface.

Also, we assumed that a user can handle only one call at the same time. For example, if the user receive a private call during an audio chat room conversation, he will leave the Room before to accept the new incoming call. However, we will see later on that the application is developed in order to provide high scalability and can easily handle multiple calls with very few modifications.

Presentation of extra features implemented

Remember, the basic feature is to establish a communication between at least three simultaneous users.

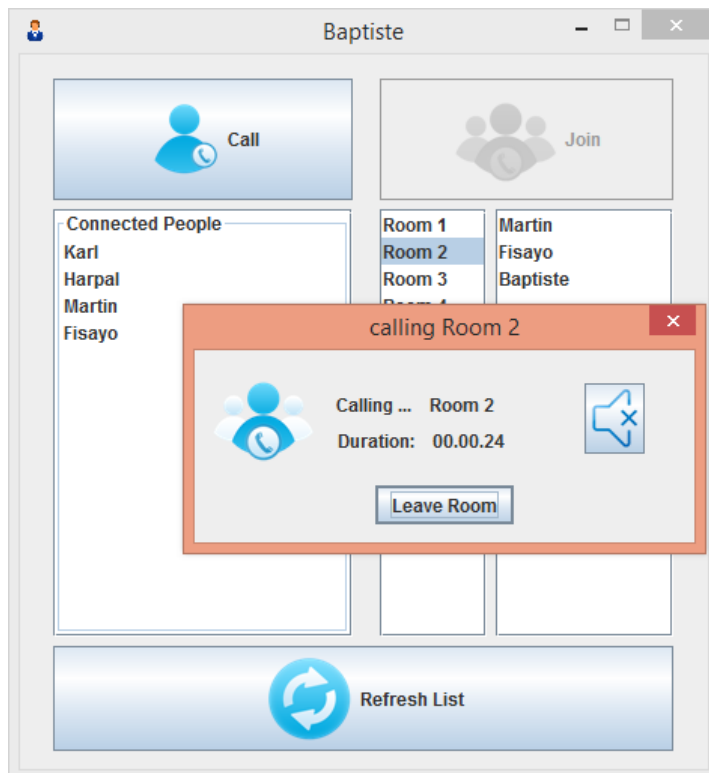


Figure 4 – Conversation between Martin, Fisayo and Baptiste.

We thought that the following extra features improve the application and the user experience:

- The server can be launch on different machine so the IP address of the server should not be defined in hard on the client code. So, when the user launch the application, a first window asks the IP address of the server. At the same time, the user can enter his name/nickname for the audio chat.

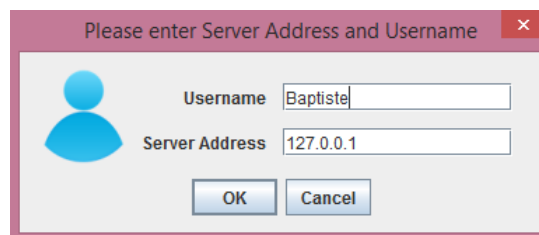


Figure 5 - First window when a client open the application.

- The user should not have the possibility to call himself. If the user join a room, his name can appear in the Room_X list participant because he is actually a participant (see Figure 4). However, his name should not appear on his list “connected people”.

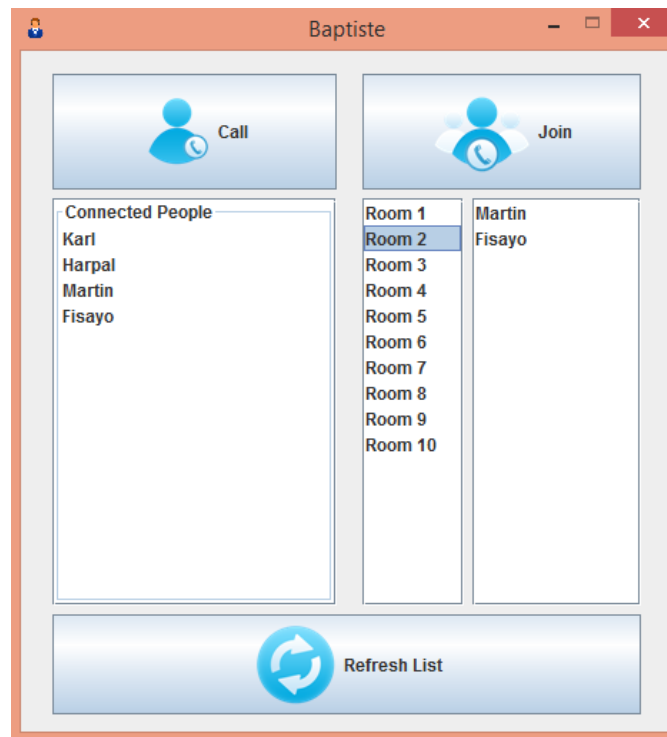


Figure 6 - The user cannot see his name in the "Connected People" list.

- The user can make private call in order to speak privately to one person. Other users are not aware of this communication.

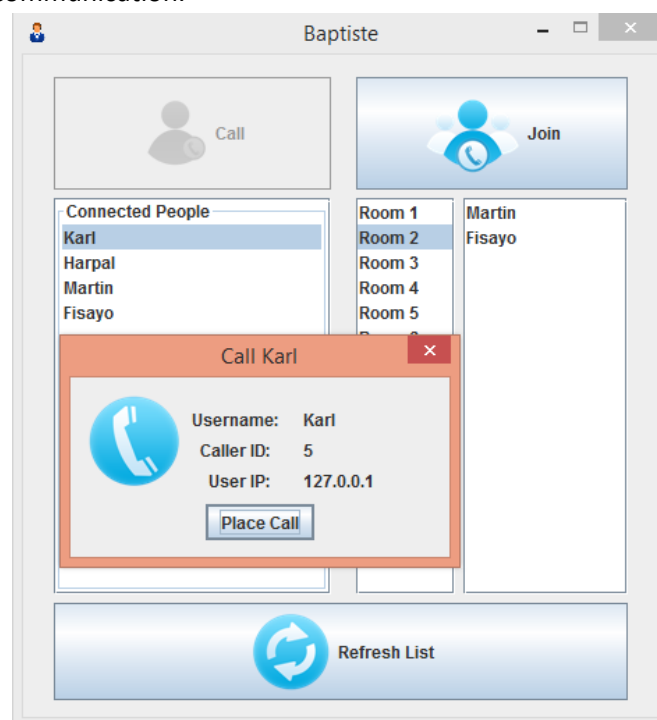


Figure 7- Private call between Baptiste and Karl.

- During a chat conversation, we can be not concerned by a part of the discussion during a certain moment. Thus, the user have the possibility to mute the audio of the application and unmute it when needed.

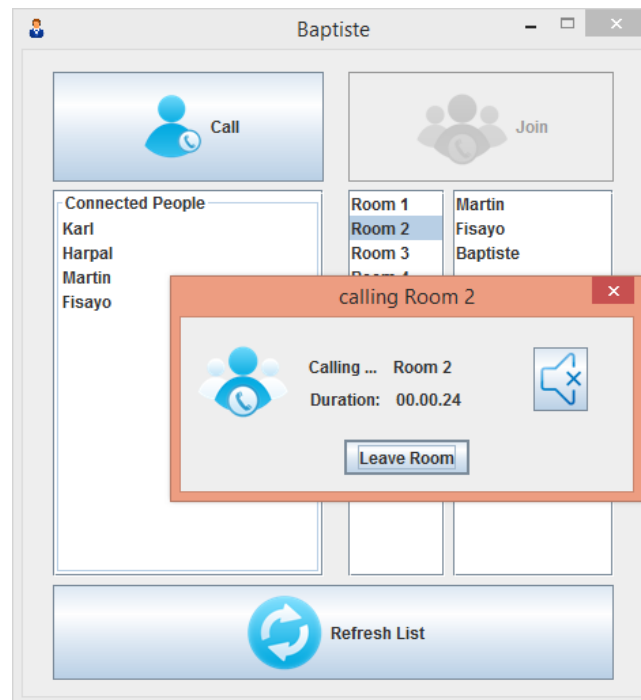


Figure 8 - Mute and Unmute audio of the conference tool.

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. Past years, students had some issues with the development on Windows. However, Windows is the most used Operating System so we wanted to make our application working on this OS.

Also, we were advised to develop in C++ instead of JAVA to avoid some compatibilities issues during our coding part. However, our skills in JAVA are much more developed than for C++ and our LAB1 has been developed in JAVA. We decided to develop our knowledge in Java rather than learning a new programming language leading us to a slower development phase. In the same time, we were able to reuse some part of the code from Lab1.

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

Name	Date modified	Type	Size
build	02/12/2014 18:30	File folder	
dist	02/12/2014 18:30	File folder	
doc	02/12/2014 18:33	File folder	
libs	02/12/2014 18:00	File folder	
nbproject	02/12/2014 18:00	File folder	
src	02/12/2014 18:18	File folder	
test	11/11/2014 08:44	File folder	
build.xml	01/12/2014 16:42	XML Document	4 KB
manifest.mf	11/11/2014 08:36	MF File	1 KB

Figure 9 - 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 ;
- **libs**: contains different libraries used;
- **nbproject**: contains the project's metadata;
- **src**: contains the source code and images used for the GUI;
- **test**: contains eventually some Unit Tests for the project.

Documentation files:

- **M7017E_LAB2report.pdf**: report of the whole audio conference tool Lab2 project;
- **Javadoc_AudioConferenceTool.Ink**: shortcut to Javadoc created for the project;
- **GUI.PNG**: screenshot of the application.

All the code is available on GitHub https://github.com/Udacity2048/M7017E_LAB2 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 .

SYSTEMS DESCRIPTION

Internal design and structure of the application

The application is composed of 6 packages and 17 java classes. This architectural clearly define the client part and the server part, as well as the core application and the graphical user interface.

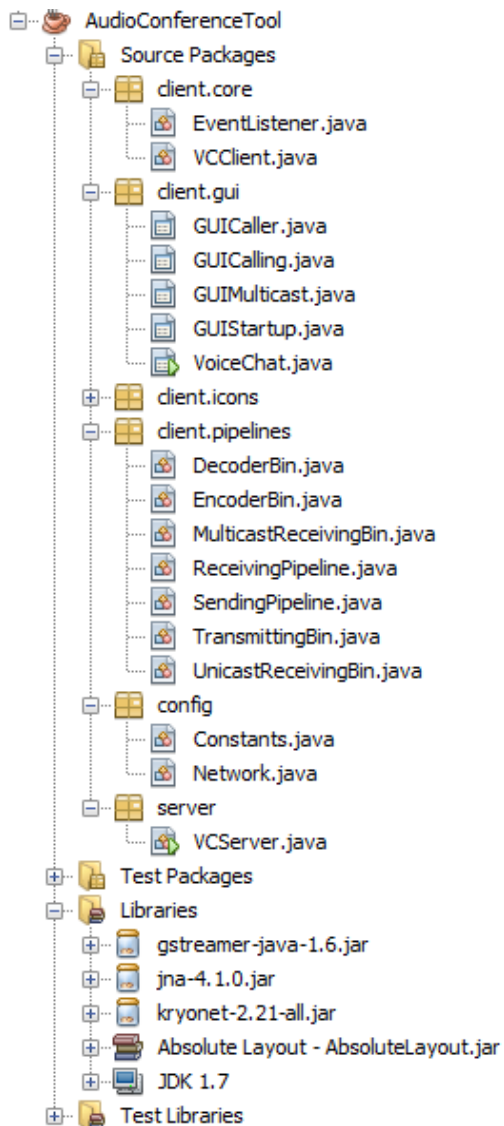


Figure 10 – Classes and packages hierarchy in AudioConferenceTool project.

Here a short and clear description of each packages:

- **client.core**: gather the core application on the client part;
- **client.gui**: gather our personal gui elements;
- **client.icons**: gather icons used in the client gui;
- **client.pipelines**: gather low level pipelines abstraction;
- **config**: gather configurations needed on both client and server side;
- **server**: gather the core application on the server part;

Here a short and clear description of each classes:

- *EventListener*: listener of events coming from the server;
- *VCClient*: client core functionalities;
- *GUICaller*: gui used before to place a private call;
- *CUICalling*: gui used during a private call;
- *GUIMulticast*: gui used after joining a Room;
- *GUIStartup*: gui used at the beginning, to gather server IP address and username;
- *VoiceChat*: principal gui of the application with main functionalities;
- *DecoderBin*: bin to decode arriving rtp packets;
- *EncoderBin*: bin to encode leaving rtp packets;
- *MulticastReceivingBin*: bin to receive udp packets concerning a multicast conversation;
- *ReceivingPipeline*: pipeline to receive audio communications;
- *SendingPipeline*: pipeline to send audio communications;
- *TransmittingBin*: bin to transmit udp packets;
- *UnicastReceivingBin*: bin to receive udp packets concerning a unicast conversation;
- *Constants*: gather all the Constants variables used by client and/or server;
- *Network*: configuration classes for network communication needed on client and server side;
- *VCServer*: server core functionalities.

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

PIPELINE DESCRIPTION

Lab2 requires a deep understanding of the pipeline abstraction used in GStreamer [2]. Past years implementations help us a lot to understand the GStreamer logic. We constructed the pipelines piece by piece with some Bin pre-constructed by GStreamer and some made by our own.

We have to send and receive audio, so we need a sending pipeline and a receiving pipeline.

The following section detail clearly our pipeline construction in plain text. However, I advise you to have first a look on Figure 11 (next page).

Sending pipeline

First element, we get the sound from the micro with `“autoaudiosrc”`. Then, we encode this audio in our `“EncoderBin”` using a mu law encoder. Then we send packets using rtp/udp layers model.

The `“tee”` element enable scalability. In fact, thank to this part of the pipeline, we can send audio to unicast and multicast at the same time.

Our application can in theory handle this situation but our graphic user interface is not optimized to handle it. Because if the user can manage more than one conversation at the same time, we have to implement a mute audio and a mute micro for each specific communication and we have to give the user the possibility to switch from one windows to another – we started to implement these `“version2”` of our application, but we did not have a stable version in the time we had.

Then, do not forget that we have multicast (Room) and unicast (private call) situations. However, looking on the sending side, the difference is only the destination IP address. In the first case, we are using the specific recipient IP address (Example: 10.0.0.7) and in the second case we are using the group multicast IP address (Example: 244.1.1.3 associated with the chat room number 3 – each chat room has its own multicast IP address). Consequently, the design of the pipeline is the same.

Receiving pipeline

Looking on the receiving side, it is more complex. We need a dynamically way to handle communications because in a multicast room, a user can arrive or leave at any moment. Also, you can receive a private call while you are connected to a chat room. The element `“liveadder”` handles this complexity for us.

Also, we need to sort rtp packets arriving on the client. Each communication has to be decode with `“DecoderBin”` separately before to be gathered. The `“gst RTPBin”` pre-built Bin handles this complexity for us. Then, in order to play the sound, we use `“directsoundsink”` element which is an abstraction of the computer speakers for Windows environment.

Again, do not forget that we have multicast (Room) and unicast (private call) situations. Looking on the receiving side, the difference is that multicast need to be filter according to the SSRC number (ID different for each participant of the multicast section) in order to dynamically handle clients' pipelines and to filter our own echo voice with a `“fakesink”`.

For any more details information, the code is well commented and an API documentation is provided for the whole application [2]. Then all elements are described in the GStreamer online documentation available here [3].

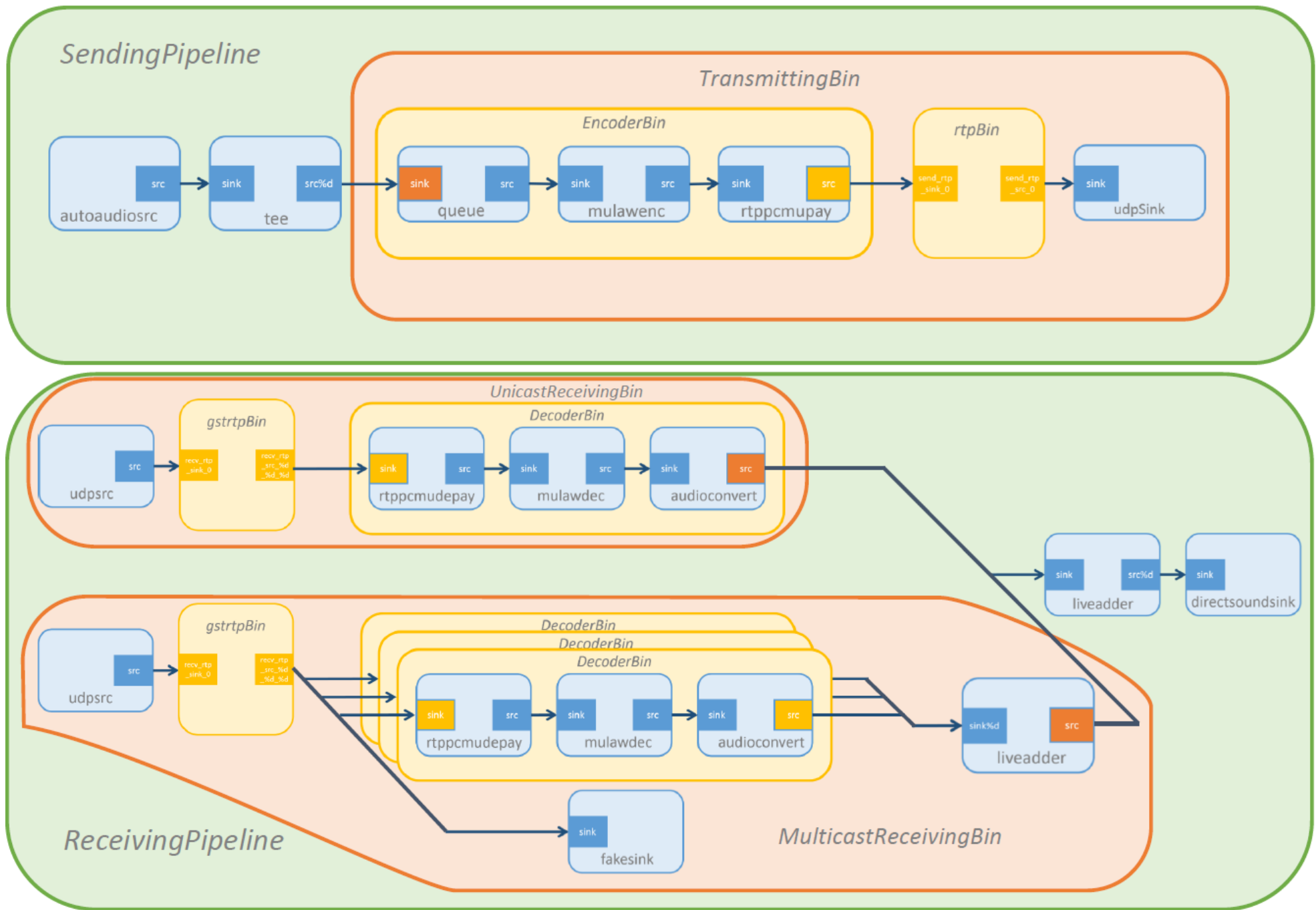
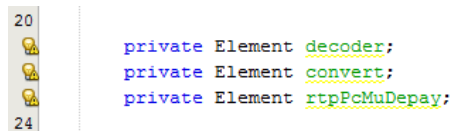


Figure 11- Pipelines modelisation.

PROBLEMS AND REFLECTIONS

GStreamer is an interesting framework for developing media-handling components. However, the pipeline abstraction is really complex when you are modifying it dynamically. Thanks to [4] [5] [6] we understood the main operation of dynamic pipelines. However, we faced many issues which are described in the following section.

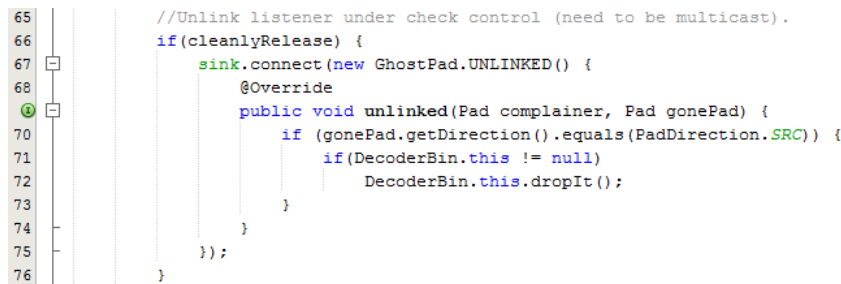
The pipeline abstraction is also fragile. For example, some variables have to be final and some others not. Be care full to not follow the recommendation of your IDE. We spend quite some time to find this error so if you want to reuse our code, we advise you to not change the variables declarations.



```
20 private Element decoder;
21 private Element convert;
22 private Element rtpPcMuDepay;
24
```

Figure 12- Do not follow IDE recommendations for variables declarations.

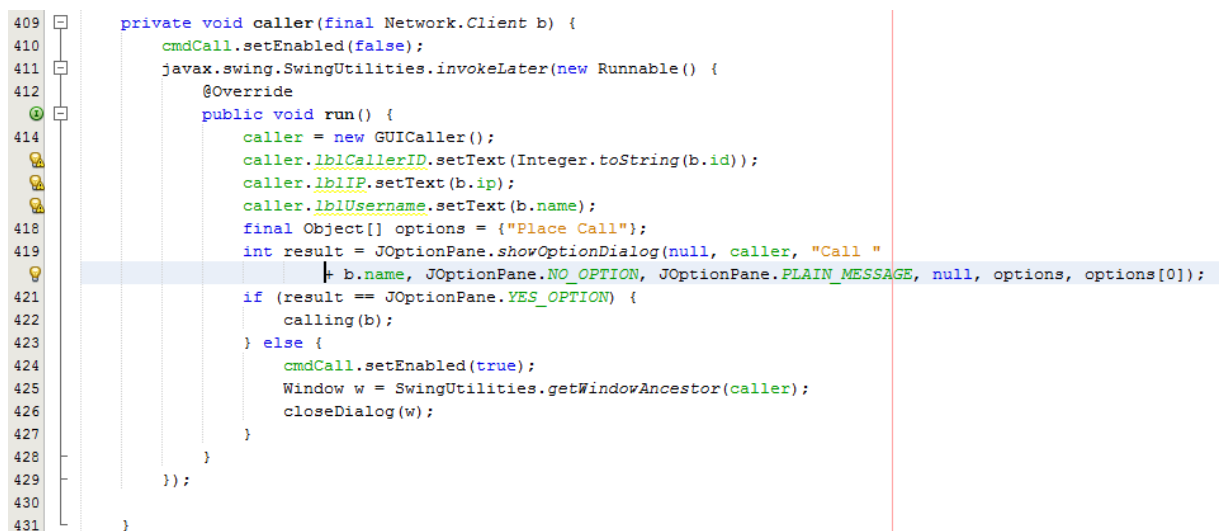
Also, we have to handle some specific situation which can freeze the application. For example, when a call is finished, we must drop the pipeline create specifically for the communication. But, in some case, this 'drop' causes a freeze on the graphical user interface and the application does not respond anymore to any user input.



```
65 //Unlink listener under check control (need to be multicast).
66 if(cleanlyRelease) {
67     sink.connect(new GhostPad.UNLINKED() {
68         @Override
69         public void unlinked(Pad complainer, Pad gonePad) {
70             if (gonePad.getDirection().equals(PadDirection.SRC)) {
71                 if(DecoderBin.this != null)
72                     DecoderBin.this.dropIt();
73             }
74         }
75     });
76 }
```

Figure 13 - Check loop to avoid freeze on the gui.

The Graphic User Interface (**GUI**) is often a gulf in the time spend for a project. Our choice has been to use NetBeans drag-and-drop JFrame to obtain a nicer GUI. However, implement all functionalities behind buttons, text fields and area lists was kind of challenging.



```
409 private void caller(final Network.Client b) {
410     cmdCall.setEnabled(false);
411     javax.swing.SwingUtilities.invokeLater(new Runnable() {
412         @Override
413         public void run() {
414             caller = new GUICaller();
415             caller.lblCallerID.setText(Integer.toString(b.id));
416             caller.lblIP.setText(b.ip);
417             caller.lblUsername.setText(b.name);
418             final Object[] options = {"Place Call"};
419             int result = JOptionPane.showOptionDialog(null, caller, "Call "
420                 + b.name, JOptionPane.NO_OPTION, JOptionPane.PLAIN_MESSAGE, null, options, options[0]);
421             if (result == JOptionPane.YES_OPTION) {
422                 calling(b);
423             } else {
424                 cmdCall.setEnabled(true);
425                 Window w = SwingUtilities.getWindowAncestor(caller);
426                 closeDialog(w);
427             }
428         }
429     });
430 }
431 }
```

Figure 14 - Example of complex thinking in the management of icon buttons.

CONTRIBUTION

LAB2 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. However, we can identify some specific responsibilities for each person.

Baptiste Louis was mainly responsible for handling the project, modelling the pipeline abstraction, commenting the code and writing the documentations including this report.

Martin Bumba was mainly responsible for developing the core application code, the server and the network communication.

Fisayo Sangogboye was mainly responsible for developing a beautiful graphic user interface and integrating extra functionalities to improve our application in the time we had.

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 – AUDIO CONFERENCE IN ADOBE CONNECT.....	4
FIGURE 2 - NETBEANS IDE. FIGURE 3 - GSTREAMER FRAMEWORK.	4
FIGURE 4 – CONVERSATION BETWEEN MARTIN, FISAYO AND BAPTISTE.	5
FIGURE 5 - FIRST WINDOW WHEN A CLIENT OPEN THE APPLICATION.....	5
FIGURE 6 - THE USER CANNOT SEE HIS NAME IN THE "CONNECTED PEOPLE" LIST.	6
FIGURE 7- PRIVATE CALL BETWEEN BAPTISTE AND KARL.	6
FIGURE 8 - MUTE AND UNMUTE AUDIO OF THE CONFERENCE TOOL.	7
FIGURE 9 - STRUCTURE OF THE APPLICATION FOLDER.....	8
FIGURE 10 – CLASSES AND PACKAGES HIERARCHY IN AUDIOCONFERENCETOOL PROJECT.....	9
FIGURE 11- PIPELINES MODELISATION.	12
FIGURE 12- DO NOT FOLLOW IDE RECOMMENDATIONS FOR VARIABLES DECLARATIONS.....	13
FIGURE 13 - EXAMPLE OF COMPLEX THINKING IN THE MANAGEMENT OF ICON BUTTONS.....	13

References

- [1] gstreamer, "Home page," 8 November 2014. [Online]. Available: <http://gstreamer.freedesktop.org/>.
- [2] B. Louis, "API documentation NetBeans," private, Lulea, 2014.
- [3] gstreamer, "gstreamer documentation," 2014. [Online]. Available: <http://gstreamer.freedesktop.org/documentation/>. [Accessed 01 December 2014].
- [4] GStreamer, "Bins and pipelines," [Online]. Available: <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/section-intro-basics-bins.html>. [Accessed 01 December 2014].
- [5] C. Notin, "GitHub," 12 December 2012. [Online]. Available: <https://github.com/ClementNotin/audioconferencing>. [Accessed 01 December 2014].
- [6] GStreamer, "Dynamically Changing the pipeline," 2014. [Online]. Available: <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/section-dynamic-pipelines.html>. [Accessed 01 December 2014].
- [7] Wikipedia, "GStreamer," 8 November 2014. [Online]. Available: <http://en.wikipedia.org/wiki/GStreamer>.
- [8] Google Project Hosting, "Project Home page," 8 November 2014. [Online]. Available: <https://code.google.com/p/gstreamer-java/>.
- [9] GTK-Doc, "audioconvert," 11 November 2014. [Online]. Available: <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-base-plugins/html/gst-plugins-base-plugins-audioconvert.html>.
- [10] GTK-Doc, "autoaudiosrc," 11 November 2014. [Online]. Available: <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-good-plugins/html/gst-plugins-good-plugins-autoaudiosrc.html>.
- [11] GTK-Doc, "Lamemp3enc," 11 November 2014. [Online]. Available: <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-ugly-plugins/html/gst-plugins-ugly-plugins-lamemp3enc.html>.
- [12] GTK-Doc, "filesink," 11 November 2014. [Online]. Available: <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer-plugins/html/gstreamer-plugins-filesink.html>.
- [13] GTK-Doc, "Ugly Plugins," 11 November 2014. [Online]. Available: <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-ugly-plugins/html/>.