M7017E Lab 2
Audio Conferencing

Team

- Flore Diallo
- Hervé Loeffel
- Clément Notin

Table of Contents

1. Problem Specification

The main goal of this lab is to send and receive audio stream on a network by using IP unicast or multicast. We decided to create an audio conferencing tool able to manage more than three people. We want them to be able to talk in real time. In order to do that, we have access to three libraries of *Gstreamer* : Good, Base and Core, which are needed to manage the audio flow transfer between the participants.

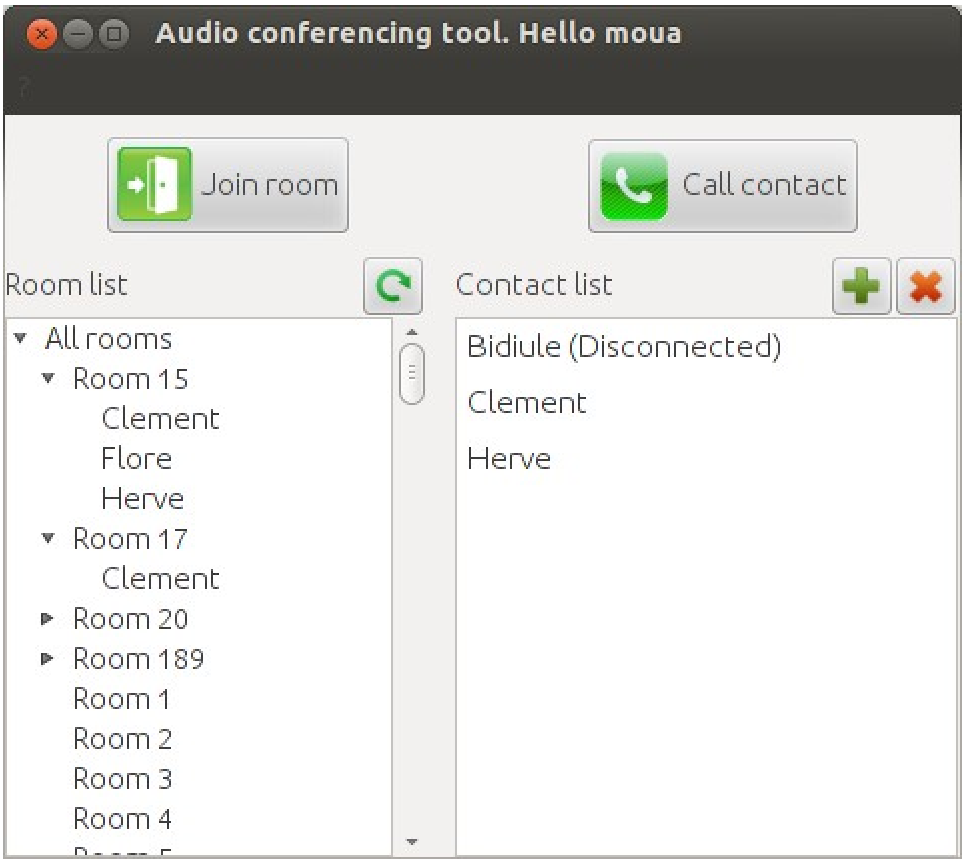
In order to make the use of this tool more intuitive and easy for the user, we decided to implement a simple but complete interface which provides some extra features. First, a personal contact list where the user can save, delete and see his own favorite contacts and if they are connected or not. He can call only one of his connected contacts if he decides to reach someone through that list.

The user also has access to a room list. He can see the list of people currently in each room and decide to join one or several rooms to hear and be heard in it. When he interacts with his contacts (or when he saves or deletes them in his list), the user only need to know his name. In order to keep it simple, the software is managing by itself all the internal network settings like IP addresses and the user does not need to take care of that at any moment.

To design the user interface, we supposed that the user has some basic knowledge in this kind of tools and that he knows the principle of discussion rooms. We also supposed that he already knows the usage of a simple contact list.

The application also allows a user to join severals rooms and to have a unicast conversation in the same time.

2. Usage and User’s guide

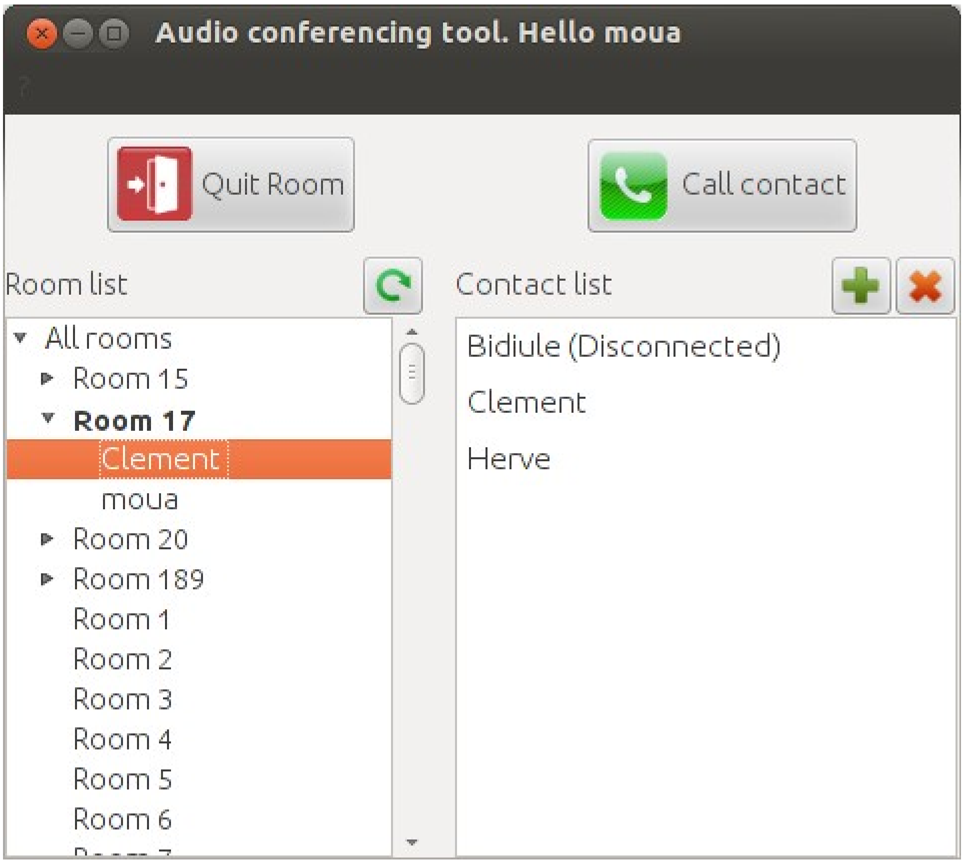


- All rooms
  - Room 15
    - Clement
    - Flore
    - Herve
  - Room 17
    - Clement
  - Room 20
  - Room 189
  - Room 1
  - Room 2
  - Room 3
  - Room 4
  - Room 5

Bidiule (Disconnected)
Clement
Herve

When you start the application for the first time you have to choose a name. This name will enable the other users to recognize you, and to add you in in their contacts list. Then you will see the application as in the previous screenshot.

You are offered two ways to talk with people. It is possible to join a room for group conversation or it is also possible to directly talk in private with one of your contacts. When you open the software, the rooms list is shown on the left. You have access to the complete list: if some rooms have already some audience, they will be on the top of the list and you will see who is in the audience, and if it is empty, the rooms will be on the bottom of the list. You join a room by selecting a room in the list and by clicking the button “Join room”, and the room name will appear in bold. When there is already one or several persons, you are automatically in discussion with them. If you want to leave the room you just have to select the room to leave and to press the same button (with the text “ Leave room”). You can see the name of the persons connected to a room, by clicking on this room.



You can add a contact by clicking on ✚and by entering his name. You can also remove a contact by clicking on✖. The contacts list shows you if the contacts are connected or not: a “Disconnected” mention appears on the disconnected contacts as in the screenshot. You can call a contact by clicking on “☎Call contact”. If a user is called by someone, he will receive a message which indicates the name of the person who call him. Thus he can choose to accept or to decline the call. In both cases the person who asked for a call will be informed of the answer. If the call is accepted, the conversation will start. The conversation finishes when one of the users clicks “☎ Hang up”.

It is possible to join several rooms and also to call one contact in the same time. But be careful, you will hear all the people of all the different conversations, and they will all hear you, it could sometimes become confusing for everyone.

3. Systems Description

Technical stack

Our project works with Java >1.6.

The UI is created with Swing and tries to use the system's native look'n'feel (the screenshots in this document were taken using Ubuntu Unity graphical manager).

The build and the dependences of the project are managed by Maven which is an industry standard for Java development. We advise to use the corresponding IDE plugin (m2eclipse for Eclipse) or the command line tool "mvn". One can generate the executable JAR for the program with all necessary resources with "$ mvn clean compile assembly:single" at the root of the project and find the result in "target" folder.

We also used Lombok to avoid generating getters, setters, default constructors etc. This does not add any runtime overhead but it must be installed in the IDE to remove errors about missing methods. You can see it for example when we use the annotations @Getter, @Setter, @AllArgsConstructor, …

The dependencies (automatically resolved and downloaded by Maven) are:

- gstreamer-java: 1.5
- lombok: 0.11.6

Packages and classes

In the same project we have a client to be used by everyone and a central server to be launched once (more on this in the following). The respective source codes are separated in different packages.

- se.ltu.M7017E.lab2.client: main package for the code specific to the client
  - se.ltu.M7017E.lab2.client.audio: package for audio in the client (mainly *GStreamer* bins, elements and pipelines)
  - se.ltu.M7017E.lab2.client.ui: package for client User Interface
- se.ltu.M7017E.lab2.messages: network protocol between client and server needs messages, they are conveniently (un)serialized through these classes shared between the server and the client
- se.ltu.M7017E.lab2.presenceserver: main package for the code specific to the server

For classes' description, please refer to Javadoc comments.

Code design guideline

We applied common Object Oriented Programming best practices. The code is organized and separated between client and server, however what could have been directly shared (like protocol messages) or through clever inheritance has been done so to prevent code duplication.

Both client and server are multi-threaded through Java standard threading mechanism.

Code is commented at every level and with Javadoc for public methods.

Methods description

The methods are documented with Javadoc comments. Please refer to it.

Data types and structures

Our software does not need specific data types other than the ones already provided by Java.

However we created our own *GStreamer* Elements when needed to enhance reusability and create clean pipelines. The Elements have been created through the means of *GStreamer* Bins and GhostPads.

Version control

We have used Git and Github to share our files within the team and to provide traceability and store development history. Here is our browsable/downloadable repository:
https://github.com/ClementNotin/audioconferencing

4. Algorithm description

There is no specifically clever algorithm in our application to explain. Only a few really simple search algorithm when Java collections were not offering what we needed.

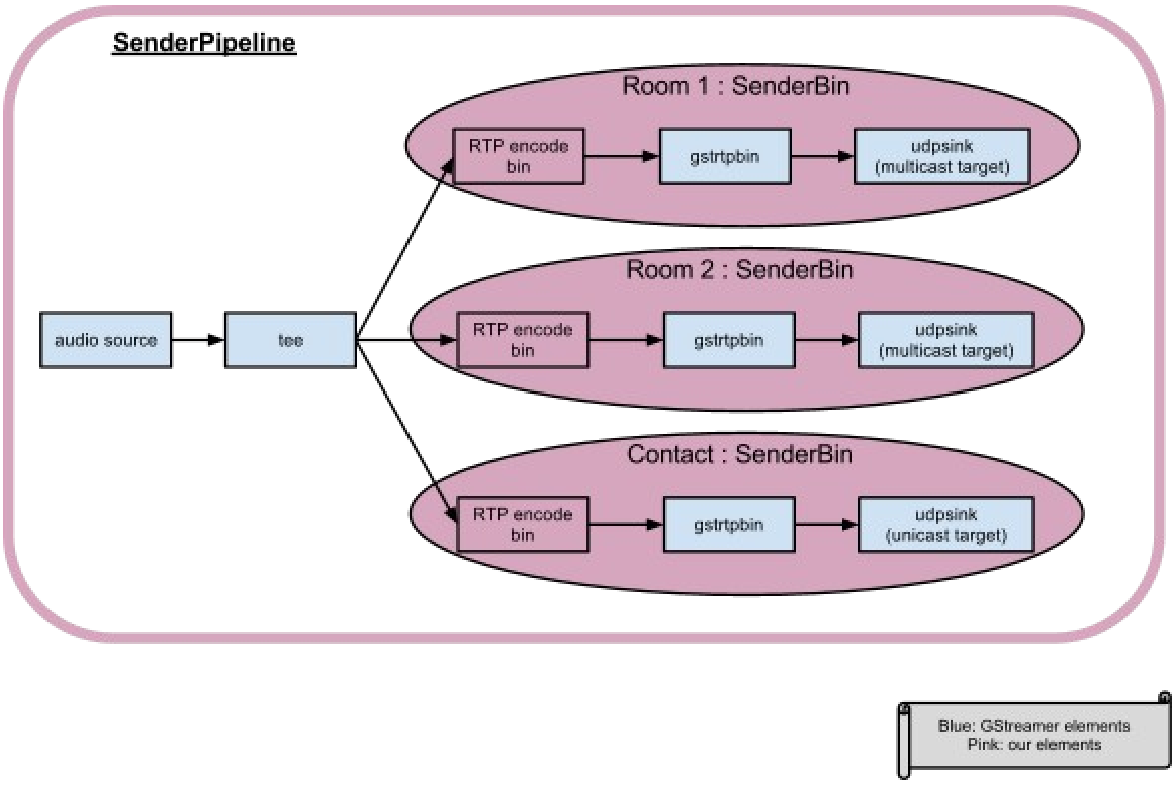
5. Audio description

Architecture

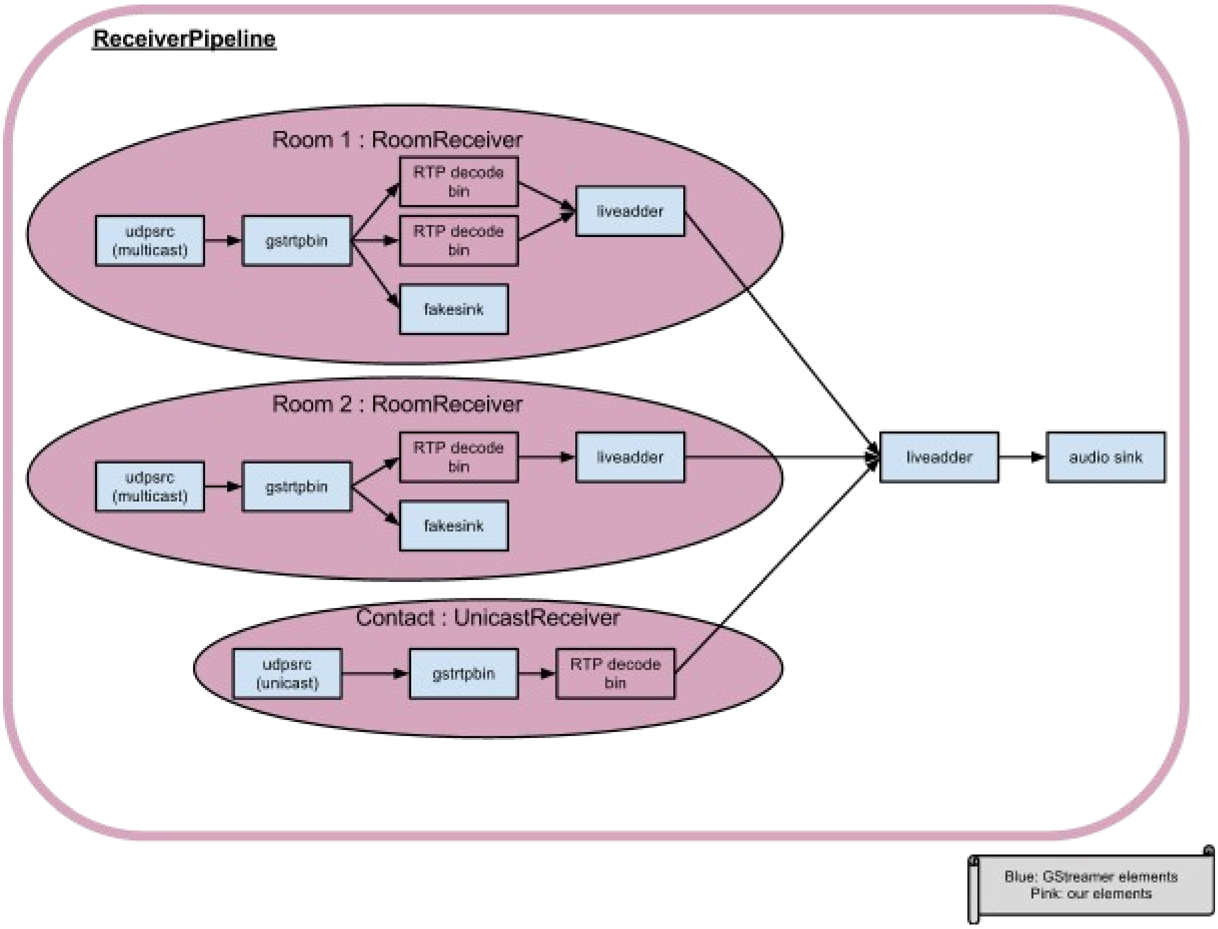
The audio requirements we have set up for ourselves asked to be able to be present and able to talk in several rooms (multicast) and with one contact (unicast) at the same time. Therefore we needed a flexible architecture with possibilities of adding/removing receiving/sending elements without stopping anything at runtime.

Our architecture is splitted in two pipelines: one for receiving (ReceiverPipeline class) and one for sending (SenderPipeline class).

Here is an example of both pipelines when the user is connected to Room 1, Room 2 and is also in a direct communication with a contact. So he sends to and receives audio from those 3.



Sending audio to a room or a contact is very similar, just the destination IP and port differ. When joining a new room it is as simple as creating a new SenderBin "bubble" and connecting its sink to the tee.



However receiving from a room means receiving several multiplexed streams which must be demuxed, hopefully this is done based on the SSRC by gstrtpbin. We also receive our own echo so it is detected and connected to a fakesink to ignore it properly. In the Room 1 for example there are 3 people connected including me, and two people including me in Room 2. One first audio mixing is made for each room (with the liveadder in the RoomReceiver), then the rooms and the direct call are mixed together (with the liveadder in the ReceiverPipeline).

The double mixing could be useful for future features by enabling an independent volume control (or even mute) on every room or participant in the room.

Therefore joining a new room is just creating a new bubble RoomReceiver and connecting it to pipeline's liveadder. Same for leaving a Room.

There is some complexity but it is masked inside our own-made Elements (in pink) that can be added at runtime as needed.

Codec

We needed to compress the audio stream to keep it under a reasonable bandwidth. Based on the assumption that this audioconferencing system will mainly be used for vocal communication (and not music streaming) we have decided to use the codec “speex” which has been specially developed for voice compression and is available in GStreamer.

The bitrate is constant and set with the parameter “quality” which we decided to set at 6 on a scale from 0 to 10 which provides a good tradeoff quality-bandwidth.

This codec offers two parameters to further reduce the bandwidth consumption: the VAD and DTX. We enabled both of them.

According to speex documentation, “voice activity detection (VAD) detects whether the audio being encoded is speech or silence/background noise. [...] Speex detects non-speech periods and encode them with just enough bits to reproduce the background noise” and “Discontinuous transmission (DTX) is an addition to VAD/VBR operation, that allows to stop transmitting completely when the background noise is stationary. [...] only 5 bits are used for such frames (corresponding to 250 bps).”

6. Server and protocol description

The communication protocol between server and clients is done with text messages (String). There is only text transmitted between server and clients, thus the data transiting via the server is very lightweight. These messages contain information about the type of message, and different data depending on the type of message. Each element of information is separated by a comma. This messages offer (un)serialization.

The types of messages are diverse. For example there is the "Hello" message which indicates that someone is connected (ex: "HELLO,Fred", indicates to the server that Fred is connected). It enables the server to know who is connected and then to send a message to other clients to inform them about a new person. When the server receives a Hello or a Bye message it transmits the list of connected people to each connected clients with a ConnectedList message. When a client receives this kind of message it refreshes automatically its contact list to show who is connected/disconnected.

There are also the Call and AnswerCall messages who are used for unicast communications. The Client 1 sends a Call message to the server containing the name of the user, his IP address, the name of the receiver (Client 2). If Client 2 is connected, the server sends him this message, otherwise the server sends an error message to Client 1. When Client 2 answers, it sends an AnswerCall message to the server with Client 1 and Client 2 names, Client 2 IP, and the answer ("yes"/"no"). Then the AnswerCall message is sent to Client 1. The two clients can get all the information required to start a communication in unicast with this protocol.

The server manages also the rooms for multicast communications. There are other useful types of messages like Audience, Join, Left, etc... They enable to know who is connected in which room...

At the beginning we thought about a fully P2P software, that is to say without any central server. Even if it is possible to implement such system, it is more complicated and given the time we had to work on this lab we preferred to keep a traditional central server.

It is important to note that no audio stream is transferred through the server. His role is only to manage presence of users and calls negotiation.

7. Problems and Reflections

Contacts Management

When using this audio conferencing tool, you can add, delete, save and call some contacts. Since the main goal of this work was not to write a contacts book, we focused on the multimedia part of the development and did not address some problems that can appear with the contact list. For example, you freely type the name of the contact you want to add. It is then possible for you to save a contact who does not really exist. User is also asked to choose a name for its first connection, its availability is not verified and collisions could happen with unpredicted consequences. No password is asked and identities could be stolen.

When someone wants to add you in his/her contact list, you do not receive any invitation, you cannot choose to refuse to be added. That could be a privacy problem if we wanted our software to be widely used, but we decided to not spent too much time on this problem since it is not the main goal of this course.

In summary, management of the contact list is very basic and can cause some problems in the software. It must be used with care.

RTP and RTCP

We do not use RTCP but it could have been useful to get statistics about other participants in order to adapt the quality on sender side to better match receiver’s bandwidth. For the audio part of this, it would not be a problem to implement it in the SenderPipeline because the streams for rooms and unicast calls are encoded separately (that is the reason we organized it like this instead of encoding once for all channels).

About the network (multicast and unicast)

We thought about the possible use cases of this audioconferencing system and discovered that it could be useful sometimes to be present in several rooms and in one unicast conversation at the same time. However being in several unicast conversations with different people at the same time did not seem to be useful (same effect could be achieved by everyone joining the same room) therefore this is not supported by our software (it would still be really easy to do it).

Rooms are identified by IDs which, in fact, are directly mapped to multicast IP addresses. Room 1 is at IP 224.1.42.1 and Room 254 at IP 224.1.42.254. For future features, the server could associate a name to each room and probably a dynamic IP selection algorithm and communication to clients.

The rooms rely on IP multicast to work. If it is disabled in the local network or not properly routed (this is the cause on the open internet for example) then it will not work. Therefore this software is particularly suitable for enterprise private networks which are fully managed.

The unicast calls make the assumption that the user is not be behind a NAT router because no NAT traversal mechanism has been implemented.

Protocol implementation security

Origin and format of messages are not checked, they could be voluntarily bogus or spoofed and lead to crashes or identity thefts.

Anyone can stream to a room (by knowing IP and port), without explicitly joining it in the server. This could be mitigated by requiring the server to send to everyone in the room the list of allowed SSRCs (unique sender identifier in RTP standard) in the room. Then clients could simply ignore stream coming from unregistered SSRCs.

As previously, in this early stage of the software we decided to focus on the multimedia, and for the remaining to rely on users’ goodwill and absence of malicious people on the network.

8. Conclusion

Even with all the problems and restrictions we have honestly raised in the previous section, we are very confident in the quality of this early piece of software. The parts we decided to put the focus on are working great. This is a good starting base to write a fully-featured and really user-friendly audioconferencing system. The code is documented and organized, moreover the audio engine is also designed to be ready to scale up for new interesting features.

9. Contribution

Flore Diallo

Developer, worked mainly in the gui development and the room list management.

Hervé Loeffel

Developer, worked mainly on the gui and the unicast functions.

Clément Notin

Technical leader, architect, main developer on server and audio pipelines.

Code of honour

We state that our project is compliant with the code of honour. All the production is our original team work, except for the external libraries (JAR files) and explicitly marked code.