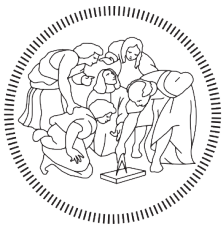

POLITECNICO
MILANO 1863

MSc. Music and Acoustic Engineering

Computer Music - Languages and Systems

Homework 3
Drum Kit

Authors:
10504151
10813836
10426992
10754748
[Group 16]

Link to the repository:
https://github.com/IronZack95/CMLS_Third_Project

MIDI Pedalboard repository:
https://github.com/IronZack95/MIDI_Pedalboard

1 Supercollider and architecture of the project

The goal of this paper is to reconstruct how we have managed to create our project using SuperCollider, a programming language used for many applications in music (playing live by coding, creating different sounds or effects, etc.).
The project was not only limited to the creation of sounds, but also to implement a way to play them without operating on the code. The main goal, therefore, was the creation of a set of sounds and effects which could be played on any platform and any occasion.
In order to allow easier modifications and to obtain a more understandable code, we separated the different project sections in 5 SuperCollider scripts. Each section provides different functions and, in this way, it is more simple to manage all the sounds and the interactions.
For the interaction parts we have connected several external devices:

- **Arduino** via serial connection
- **touchOSC** via MIDI and serial connections
- MIDI **Keyboard**, **LauchPad** and **LauchControl**
- a **DAW** (Ableton Live) for MIDI information exchanges
- **Processing** with a cool animation that varies with the output music

Splitting the code, we created some specific sripts for MIDI, serial and OSC connections. We had to deal with different problems in order to obtain a fluid and easy control of all the external devices and we used also audio and control buffers.

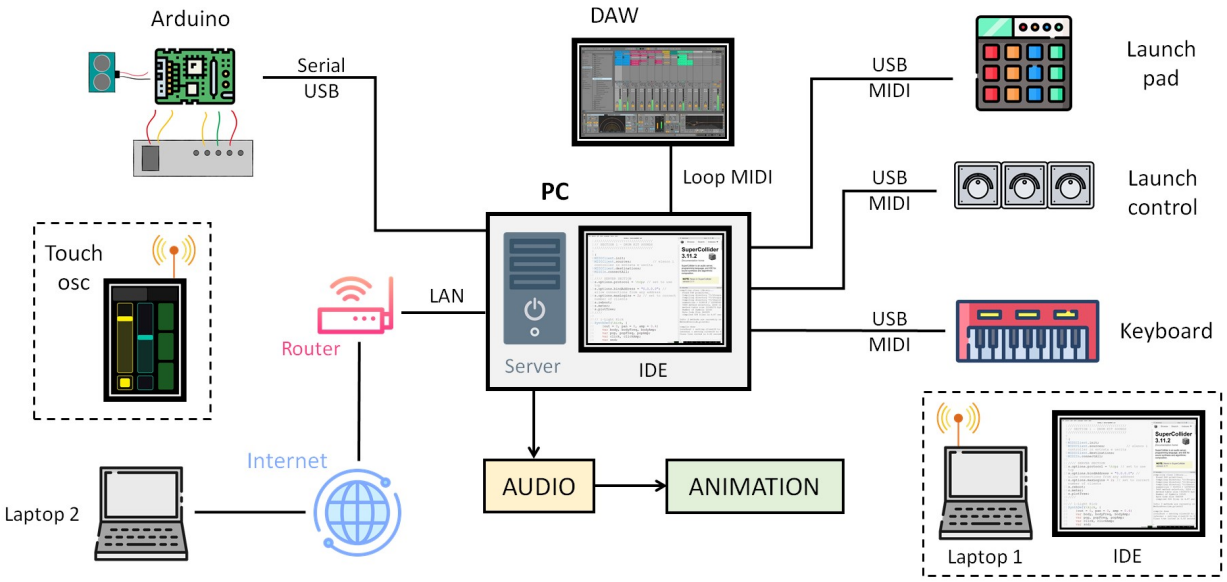

Figure 1: Architecture Schematic

2 The Drum Kit

Our SuperCollider code, as as already mentioned is divided in different scripts:
2 scripts are dedicated to sounds and effects, 2 scripts dedicated to Serial, MIDI and OSC connections and another one that takes as input the audio from a microphone using a specific bus and add an effect on it.

2.1 Sounds

Our project includes 9 Drum Kit sounds and 5 playable synthesizers divided in pads, pluck and synth sound. The MIDI interaction with external devices allows the musician to play and control them separately. We wanted allow the musician to use a MIDI LaunchPad to play the Drum Kit sounds and a MIDI keyboard to to separately play the synthesizers in different notes and chords. Furthermore, the musician can use also touchOSC to control several parameters.
All the sounds are defined trough a *SynthDef* function that includes many parameters and functions such as envelopes and oscillators.

2.2 Effects

We included 4 effects which can be controlled with the external devices:

- **Delay**
- **Reverb**
- **Flanger and Phasing**
- An **Experimental Modulation Effect** that changes with the mouse (or another controller) position.

Also effects are defined trough a *SynthDef* function and include several parameters. Some of these parameters and the Dry/Wet ratio can be controlled by external devices including Arduino. In order to control them, we used specific buffers. We specified the controlling syntax is specified in the MIDI and OSC and in the Serial scripts.

3 Interaction Design

Once every sound have been implemented in Supercollider, one has to implement a way to play them through some sort of device.
Therefore comes into play the interaction design part of the project.
The interaction design (sometimes abbreviated in IxD) is the activity of deciding and implementing the way the user is supposed to interact with the machine.
In particular, three ways of IxD have been implemented in this project, each of which rely on a slightly different standard of communication:

- **MIDI**: Interaction with a MIDI controller, therefore based on the MIDI protocol.
- **OSC**: Interaction with the TuouchOSC app which stands on the OSC standard.
- **Serial Sections**: Serial Sections communication in order to make the Drum Kit able to interact with Arduino.

Let us now analyse them in more depth.

3.1 Interaction with MIDI controllers

The first IxD method which has been chosen was the MIDI standard.
In particular ,thanks to its simplicity of implementation, three MIDI devices (the Launchpad S, a keyboard and the Launch Control) have been connected to Supercollider.
Let us analyse more in depth how MIDI connections are created in Supercollider.

MIDI communication relays on two types of buffers: audio buffers and control buffers. While the formers deal with transmitting digital audio signal from the different devices to the network and then adding them on the main output buffer (in our case a sound-card), the latters take care of transmitting signals to control parameters (an example of control signal could be the MIDI message sent from a knob to Supercollider for controlling a specifi effect).

Now that we have introduce how the MIDI communication works, the attention will be now focused on the most struggling quest before which the project has placed us: making Supercollider remember previously connected devices. In fact, When a MIDI device is connected for the first time to Supercollider, a random identifier number gets assigned to it.
Since this identifier changes every time a new session starts, if one wouldn't save it into some sort of variable, Supercollider wouldn't be able to recognise the device for what it really is.
In order tto solve this problem, a piece of code which fix it has been written.
It works as follow: instead of making Supercollider randomly assign a number to the device, it makes Supercollider read the name (which is a string) of the device and assign to it a specific number stored into a variable.
In this way, each device has its own identifier which doesn't change every time a new session gets started.

Once the MIDI devices are correctly instantiated, the user is able to control sounds and effects via the knobs, the buttons and the keys on the controllers.

3.2 Interaction with TouchOSC

Another IxD method which has been implemented is the Open Sound Control (abbreviated in OSC) protocol.
It is a standard of communication for music data used as an alternative to the MIDI standard when a higher resolution is required.

In the project it was used to communicate with a mobile application called `TouchOSC`.
This app enables the user to create, via a software editor which comes together with the app when purchased, his own graphical interface.
When linked together with Supercollider it works just like a software-based controller.
in particular, it was set to control the same commands of the MIDI controllers plus, by selecting them from the UI, four more synth sounds.

Since the way Supercollider communicate with TouchOSC is pretty similar to the MIDI one, it will not be discussed here.
The attention will be focused, instead, on the most challenging task faced during the development of this project: automatic updating of the UI.
Since the effects (in particular) could be changed via the app or via the MIDI controller, there could be a situation in which, after changing a value of an effect from the controller, the UI of TouchOSC would not be updated to that new value, showing the old one.
In order to solve this problem the `Routine` function has been called.
This function enables the UI to change according to the value which a particular effect is set on. It does so by repeatedly checking the values of the effects and accordingly updating the UI to that new value.

3.3 Serial communication with Arduino

The third and last chosen IxD method was that of serial communications; in particular with `Arduino`.
Thanks to its versatility indeed, one would not find difficult to connect any sort of electronic device in order to control any parameter.
In particular, it has been decided to implement a parameter control via an ultrasound sensor.
If, in the first place, the sensor was set to control the pitch of one of the synthesiser sounds, due to the low efficiency of the device, its purpose has been changed to control the reverb on the TouchOSC app.

4 Interaction with DAWs

As seen in this paper, many ways to interact with this project have been developed in order to promote a more user-friendly usage.
To enhance even more this aspect, it was decided to implement a communication with DAWs (in particular, it has been tested on Ableton Live, but the concept applies to any DAW).
By doing so, even a producer who doesn't know how to use Supercollider could find this project appealing for his purposes.

To do so a third party software has been used: `Loop MIDI`.
Loop MIDI is a program (analogous to `Virtual Audio Cable which will be seen later on`) that tricks Supercollider into recognising Ableton Live as a MIDI device; data from Ableton are sent into Supercollider, processed and then sent back to Ableton.

In this way a user is able to play sounds on supercollider by exploiting a more user-friendly DAW.

5 Network architecture

Another aspect of the project which is worth discussing about is the network architecture. With this concept is intended how wireless communication between a client and a server, both running Supercollider, has been made possible.
Due to the fact that some of the people of the working group were in different cities one from the other (because of COVID-19 or other reasons), it has been thought of a method to play from different places in the world.

At first we wanted a bidirectional communications between laptops running Supercollider, but because of technical difficulties, we opted for a `multi-client` architecture.
Multi-client architecture means that there is only a single server which deals with processing data, while many clients interact with it by deciding what it is supposed to do.
The server is the only one which runs scSynth.exe application, while the clients run scLang.exe program controlled via the IDE.
Communication between them is made possible through OSC standard.

6 Processing and animation

Last but not least, an animation has been coded in order to visualise the sounds.

A specific programming language has been used in order to create an animation capable of changing according to the different sounds of the Drum Kit; its name is `Processing`. Processing is a powerful programming language often used to develop interactive content, games, animation and even generative artwork. In this project it has been used to create an animation which could move according to the different sounds of the Drum Kit.

The animation we used is based on the code of an animation of Samuel Lapointe called *ProcessingCubes* available on Github. This code would allow you to make an animation only on an mp3 track, but we have implemented it in such a way that it works on the real audio buffer, that is, what we process in real time.

We used the *minim* library that allows you to take the sound card's input buffer and use it as system input. We used a third-party driver called *virtual audio cable* which consists of a sort of virtual cable for the audio signal: basically it is as if everything that came out of the sound card or in general from an audio program, was reinserted into the computer as an input to another audio program.