Thanks

I would like firstly to thanks Morten Fjeld, the laboratory director and my tutor, for his welcome, attention, help and trust. The work conditions were really good, it was a pleasure to go to work everyday. The office was well organised and I had all softwares, devices and office functionalities I needed to work on my projects. An other thanks I want to give is for his everyday life help. Thank you for your attention about accommodation, administration problems and atmosphere in and out of the laboratory. Thanks for allowing me to work on a project I had interest with and for making me discover new and really large computer subjects like Java Sound.

Secondly I would like to thank Johan Sandsjö for finding the Bounce Slider Concept, which was a really interesting project, I had loved to work on. It was a real pleasure to meet you and discuss to look for improvements.

I also thank all my teammates for their friendship. Special thanks to Nima Shahrokni for helping me to start using the Slider API and to work with me to improve it, it was a good teamwork. Thanks to Julio Jenaro to solve mechanical problems of the sliders and build the new slider box. A global thanks to Tommaso Piazza, Jonas Frediksson, Fredrik Gustafsson and all the team for the good time we had at the laboratory.

Finally I thank Malin Blomqvist for helping and giving me so much time to solve administration problems.

These six last months were a really good experience for me and I loved to live in Gothenburg. It was so nice to work with foreign students from all Europe and share knowledge and good time together. I learnt so many new things like at work for the way to organise my software development or in my social life, which was so interesting. I met a lot of interesting people and I made so many new friends from all over the world. I can really say that this experience was perfect, I can't find anything I didn't like.

I wish the best to the T2i lab and I hope I will have enough time to keep going on the project and continue this adventure.

Thanks everybody for these 6 six months.

Abstract

This report describes my internship I made in the TableTop Interaction Laboratory of Gothenburg, Sweden. I am going to show you the different steps I made to develop and implement softwares using physical sliders with force feedback. The final goal was to use one dimensional force feedback slider box to replace graphical sliders and help the user to have an easier and faster interaction with the software. Indeed, one of the slider box utility could be to decrease the time spent on a software to use graphical sliders when you need to manipulate many variables. The purpose is to allow the user to move many sliders in same time instead of only one graphical slider moving with the mouse. This device could be used for investments, medical and mathematical softwares for examples.

To accomplish this goal, I had to work step by step to gain more skill about new subjects I never had worked on. This paper will show you three steps or projects I made on the slider to finally make a project using everything I learnt before and make an innovative software.

After describing firstly TableTop Interaction Laboratory and the Force FeedBack Slider (FFSlider), I will make a presentation of a sound modification software I made. This application was made to make me start using the FFSlider API with one slider and gave me a basic knowledge of the Java Sound API. The second software I worked on was InvestView, which is used in investment and taught me how to implement an existing software using a Slider Box. And finally the last software I made was the Bounce Slider which is a psycho-acoustic sounds maker using Slider Box with six FFsliders and all the previous knowledge I had with the last applications. I will finish this presentation describing the different improvement we could do and a range of applications we could do in the future.

Contents

1. Introduction

1.1. TableTop Interaction Laboratory (T2i Lab)

TableTop Interaction Laboratory is located in Chalmers University in Gothenburg, Sweden and is directed by Prof. Morten Fjeld who created it around 2004.
The research interest is Human-Computer Interaction and mostly Tangible User Interfaces and New Interaction Technologies. The Laboratory is a single room located in Computer Engineering building of Chalmers and provide computers, softwares and devices needed for the researchers. The laboratory can welcome five or six students per semester. Students who work there are foreigner students and local student wanting to make master thesis, internship or just participating as a course to have more work experience. The workspace is divided in two sections with electronic and hardware part to build new devices and a software development part. The two parts work in close collaboration to correct and improve the new systems.
The work produced in the lab can be software development, new interaction devices, scientific publications, presentations to find new sponsors and conferences to show the new results.

1.2. T2i Lab Project Examples

1.2.1. Augmented Chemistry

Augmented Chemistry is a low cost application that uses a tangible user interface to teach molecule chemistry at school. This device can be used with a simple screen and a webcam to manipulate chemistry molecules in a virtual world.

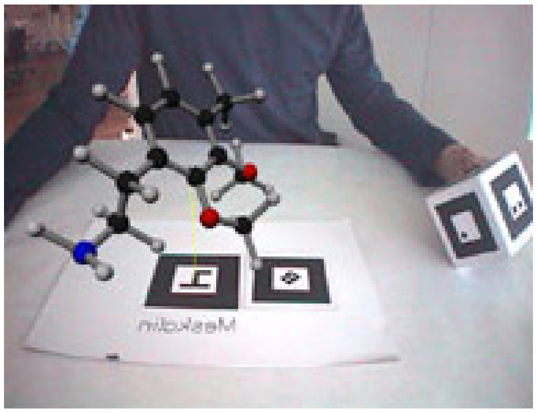


FIGURE 1.1: AUGMENTED CHEMISTRY

Using a graphic interface, you can choose into a library to visualise a molecule, which will appear on a scare pattern shown to the webcam. A pattern is a black scare with special signs printed on paper that the software will recognize. Moving this pattern, you can see on the screen the molecule following and if you use and move a cube pattern, you can make rotations of it and

play in a real 3 dimensions world. This application can be used with many types of patterns to interact with the computer without using any mouse and allow you to change a lot of features.
To use this patters, you only have to show them to the webcam and then, it will erase the molecule on the screen, show an other molecule or apply some filters like electro-negativity. As an exercise, the student can use a book to choose a special pattern to select an atom and build his own molecule and by this way, he will learn the specifications of every atom.
This project use the concept of "ARToolKit API" which provide a library to detect shapes and pattern using a simple home webcam.

At the beginning of my internship, I started to use AR ToolKit API and I tried this application so I was used to make presentations of it for some customers or potential sponsors.

1.2.2. Ortholumen by Tommaso Piazza

Ortholumen is a light pen based tabletop interaction system that can employ all the pen's spatial degrees of freedom (DOF).
The screen is projected on a transparent table from under and a webcam record the movement of the pen following the light spot. Every picture is analysed to detect the light position and its shape to know the orientation.

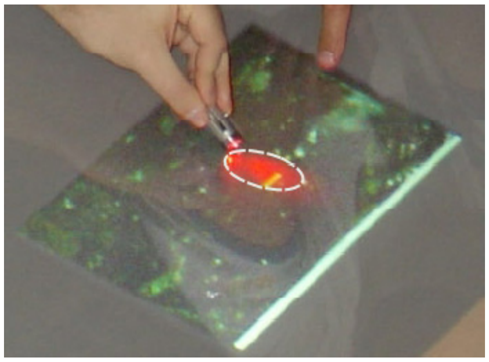


FIGURE 1.2: ORTHOLUMEN AND MAP APPLICATION

This system can be used for example to draw without using the mouse and to behave like a brush. When you come closer to the table, the light shape become smaller and so the line you are drawing become finer. Another example is to use the application "Google earth" to move into the map. Using the light pen and moving it on the table will interaction with the computer. This one will analyse light shape properties, detect the pen orientation and start to move into the map to the direction you want to. You can also make a zoom in or out depending on the distant between the pen and the screen and so the size of the light spot.

2. Force FeedBack Slider

2.1. Definition

The Force FeedBack Slider is a one dimensional, physical and motorized slider, which use a motor to produce tangible interaction in position and force as input and output parameters.

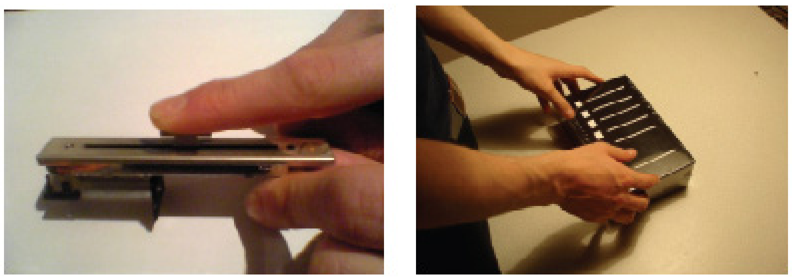


FIGURE 2.1: MOTORIZED SLIDER (LEFT) AND SLIDER BOX (RIGHT)

This device is made of a sliding button (the handle), which can move to 256 different positions and a force from 0 to 250. The new version of the FFSlider is based on a digital platform and has an additional toggle button on its base. This system allow you to make different kind of interactions using a variation of position used in a previous application called "FeelTheBeat" and shown the sound envelope variations. Another kind of interaction use the force variations as it was made in the software "Catapult" to simulate an elastic tension. All this example will be discussed demonstrate in the next paragraph.
The new step of the Force Feed Back slider is to be associated as a Slider Box using a group of 5 sliders and toggle buttons and another one generally used as a master slider to control global features. In the goal to connect many sliders, an additional board was made to connect at the maximum 16 devices per board (slider or board) and connect them to the computer via an USB cable.

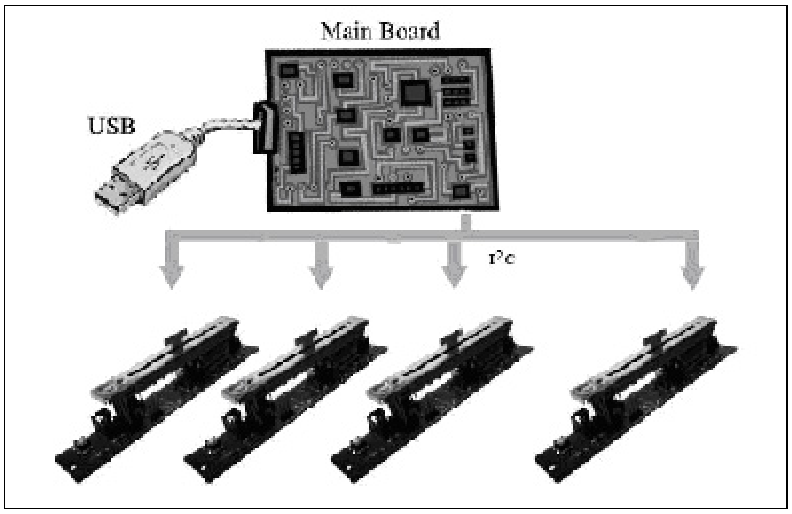


FIGURE 2.2: MAIN BOARD AND MULTI SLIDER CONFIGURATION

2.2. Previous Work

2.2.1. Catapult Application

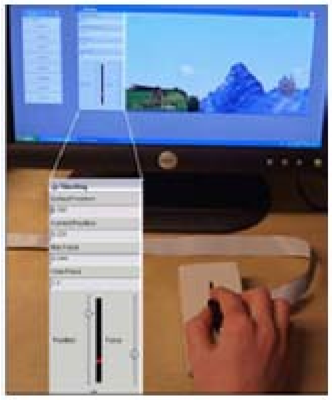


FIGURE 2.3: CATAPULT

The Catapult is an educative application giving a tactile and visual feedback of physics laws by manipulating a catapult (the slider handle). Using this software, you can change the catapult force and other features to try to touch a far away castle dropping a rock. When you start this game, the handle is up and you have to pull it if you want to shoot. During this action a force will be apply and increase depending of the distance from the beginning point.
This example gives the sensation of a real elastic behaviour.

2.2.2. Feel The Beat

This application allows you to make a direct manipulation of sound during the playback. During the playback, the slider handle moves following the amplitude envelope. If you hold it and start to move it, you will modify the amplitude of the sound at the position of the playback.
This application was written in C++ using the analogue slider, which is the old version of the FFSlider and was directly connected to a sound card. In the next chapter, we will see the first part of my internship which was to adapt this software to the new digital slider and make a demo of this new one.

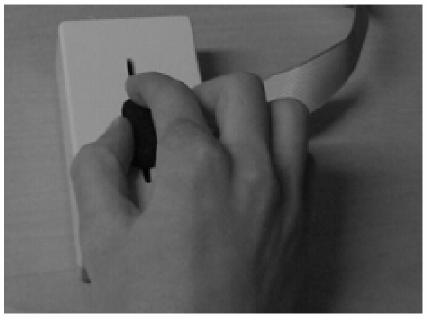


FIGURE 2.4: FEEL THE BEET

2.2.3. Remote Communication

This software is the first application using the new digital version of the slider. Moreover, it is also the first software using two sliders connected via a board and interacts together. The goal of this new device is to create a communication with two sliders to give physical feedback between two persons. Indeed, if the first user moves his slider and apply a certain force, the second user will see the slider moving to the position wanted and if he holds it, he will feel the force of the first user. It could be used with communication software like Msn or Skype to give more tactile feelings to users. Like hold the hand.

2.3.FFSlider API

The slider API is the Java library, which allows you to configure and manipulate the boards and sliders. Before starting with any project, I had to read it, understand it and try it to get a basic knowledge to start working.

The API is made of four classes:

- **FFBoard.class**: Open and configure the board, which communicate with the sliders. This class use another API named Jd2xx to communicate via USB, send and receive information from the board chip.
- **FFSlider.class:** Use the FFSlider to provide the same services than a JSlider and other functions to set the slider force, slider position and add listeners.
- **SliderAPI.class:** To define all the slider parameters and make the link between the board and the slider.
- **Text.class:** Launch and test all the services and a small application with graphical interface.

The main thing which change from a classical use of a graphical slider is the different new interactions you can use with the physical slider.

2.3.1. FFSlider Haptic Modes

An Haptic mode is a kind of interaction or feeling you can give to the user using different behaviours of the slider. These behaviours are defined in the FFSlider class and provide you five interactions:

- **Position**: The slider is moved in the specific position and stop there with no force.
- **Elasticity**: The handle is fixed in a position and when you will try to come over, a force will increase trying to come back in the first position.
- **Friction**: A force is applied to the slider and when moving and after the release, this one stay in place.
- **Detents**: Step by step mode where you can feel a small force which stop after the defined position. This mode is also called tick mode like it is used with a classic JSlider.
- **Texture**: Which provide a vibration to make you feel a rough surface.
- **Oscillation**: After a damped sine movement, the handle comes to a rest.
- **Usermode**: This mode is the most used because it gives more freedom to make listener and interact with every properties. The concept of Usermode is to send events every time the user moves something. This information is built in 2 data. The first one is the property name and the second the value. The kind of properties can be the user force which is applied, the new position or if the button is pushed.

2.3.2. *FFSlider Declaration in an application*

This is an example of how to declare and configure the FFSlider.

```
private FFBoard boards[];                 //Boards 1 to 16 boards connected
private FFSlider Fsliders[];              //Sliders up to 16
try {
      SliderAPI api = new SliderAPI();    //Open the device
      boards = api.getBoards();           //try to get a board


      //try to get the number of sliders connected
      int numberOfFSliders = boards[0].getComponentCount();

      }
catch (Exception e)
      {
      System.out.println("No Board,Slider detected /");
      }
```

```
////// ------Configuration of the FFSliders ------////////////////

Fsliders = new FFSlider[numberOfFSliders];


for (int i = 0; i < numberOfFSliders; i++) //for every slider
                  {
                  Fsliders[i] = (FFSlider) boards[0].getComponent(i);
                                                //get the slider i
                  Fsliders[i].setMode(SliderAPI.USER_MODE);  //Mode
                  Fsliders[i].setSliderForce(200);    //Force (0-250)

                  //////----Classic Jslider properties

                  Fsliders[i].setPaintTicks(false);
                  Fsliders[i].setPaintLabels(false);
                  Fsliders[i].setOrientation(1);
                  Fsliders[i].setMaximum(500);
                  Fsliders[i].setValue(0);
                  }
```

Except for the declaration, FFSlider can be used with the same functions than a graphical slider. A particular use is made of the Listeners on the FFSlider, we will discuss of it later.

2.3.3. FFSlider API Improvements

As all new devices, some bugs appear during creation of new applications asking for some functions that could make the use easier. That's why I had to make some improvements in collaboration with Nima Shahrokni, an old student who worked on the API.

Maximum and Minimum:

During the use of a financial application, it appeared that the FFSlider was made to always have a minimum of 0 and a maximum of 255. When you use a classic slider, sometimes you have to change the range start at 1 or more instead of 0 and so we had to implement some functions. In fact, it was allowed to set the graphical slider min but it didn't change on the physical slider.
It was the same for the maximum because you needed to make a conversion when you wanted to get or set a value. For this reasons, we added this conversions into the API. When you ask to receive the position, the reference position or the value, the conversion made of a proportion between the graphical and physical sliders :

```
value=255*(referencePosition-getMinimum())/(getMaximum()-getMinimum())
```

255 :	Max Physical Slider
referencePosition:	Physical slider Value
getMinimum:	Graphical slider minimum
getMaximum :	Graphical slider maximum

Sliders Panel :

At the beginning, the FFBoard.class was a JPanel containing all the sliders connected to the board in one time. The problem was, there was only one way to make an interface, which means all the sliders next to each other. This system was not enough flexible if you wanted to add some features next to one slider like a label to show the value.
Because of this, when you wanted to take only one slider and put it in a different panel, a new problem appeared. When the user updated a value, the board changed the slider corresponding in its panel. In fact, this slider was not shown in the interface because of the board creation, the Jslider was recreated in the Board panel and this Jslider is not updated.

To improve this part, we added in the FFBoard.class one tab to store one Jpanel per Slider and one variable to have the number of slider. Indeed, now the board is not made of one panel containing all the sliders but as many panels as sliders. To use this new option, we also had a function to get a Panel or a graphical slider to build an interface in the way you want.

```
Example: Panel.add(boards[0].getPanelSlider(NumSlider));
```

3. Sound Modification Demo Development

3.1. Presentation

This sound application is made about the previous software named "Feel The Beat" and allows the user to make some modifications during the playback of a sample. This small project was a beginning to start working with Java Sound API and took me **two months**. I rewrote it at all in Java without the original code of "Feel the beat". For all my projects I going to describe, I developed them with 'Eclipse'.

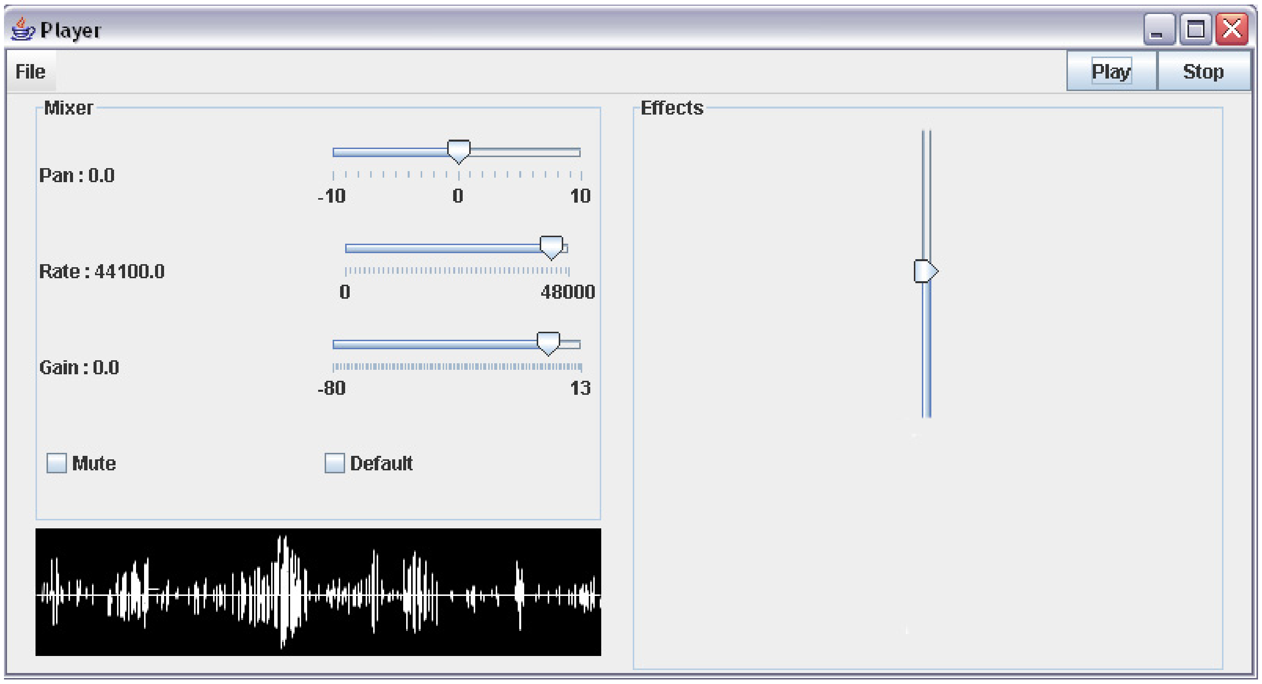


FIGURE 3.1: SOUND DEMO INTERFACE PLAYING A SAMPLE

The purpose is to allow the user to choose a music sample into his library and play it in loop. During this loop, the user can change some basic features. The different enable controls are pan to put more volume in the left or right speaker. The other one is rate to change the speed or the frequency of the sample and the last one is gain to control the volume.
Using the FFSlider, you will see it moving with the music rhythm and showing the amplitude envelope. If you move it during this time, you will start a record of your slider movements, which will be repeated, in the next loop. At the end of this loop, a red graphic will appear on the interface and hide the previous sample graph. At this moment the music playing is modified and you can hear that the part you are hiding with the last slider movements are not anymore played.

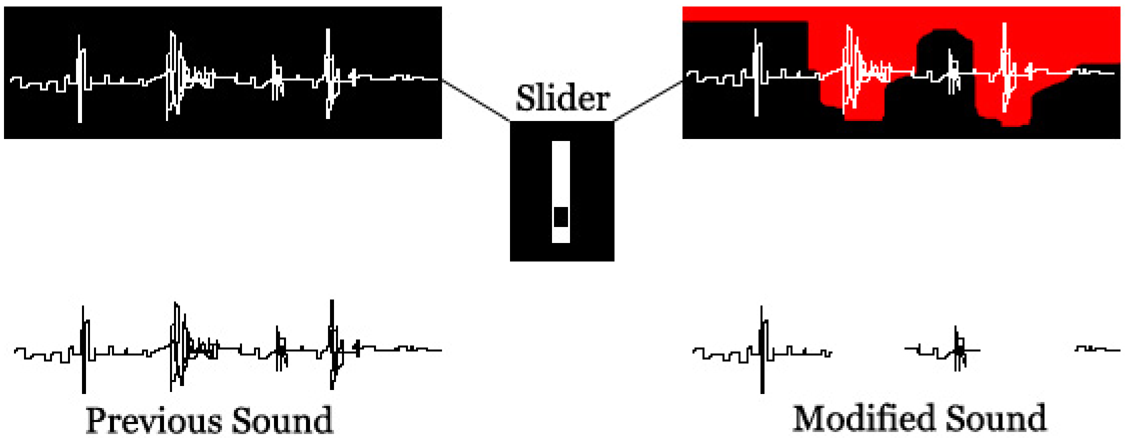


FIGURE 3.2: MODIFICATION OF A SOUND

3.2. Technical Aspect

3.2.1. Java Sound API

The main work of this application was probably about Java Sound API and more particularly the first part concerning Sampled Package. Indeed, Java sound API is divided in two packages containing Sampled and Midi part.
To write an application using Java Sound API, there are no many problems until you know what to use and for this we need a precious knowledge about what is a Line.

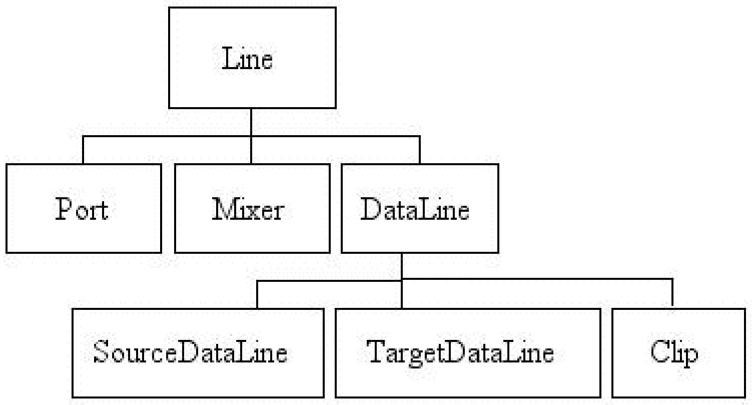


FIGURE 3.3: LINE HIERARCHY

A Line is the most important feature of Java Sound API and gives access to all the different lines. It's an Interface that provides services and controls such as gain, pan and rate.

Built of Line we find after three important objects like Port to access physical sound devices for example a cd-rom. A part we are interested about is the mixer, which provides the services pan, gain and rate we used for this application.
The last object is the most important and used with this API, the DataLine, which allows you to access to media with SourceDataLine and Clip and to get an extern source, like a microphone with TargetDataLine.

For this application, the goal is to playback a wave sample so we had to used SourceDataLine or Clip. The SourceDataLine characteristics are to be able to play a really big file but using a buffer, which is a longer to use and it is really difficult to make a loop. Because of theses properties, I chose to use Clip to easily make loops coupled with a mixer for controls.

3.2.2. Sound sample graph

At the beginning, the interface was really simple with the less options possible but after a discussion, it seemed to be easier to have a visual help added to acoustic feedback when you use the slider. Indeed, it's better to follow the current position and really needed when you want to see if the modification you wanted to do on the sample is well done.

The graphic is made of a Thread, which reads a first time the sample and stores the data into a tab. Depending on the black scare dimensions, a function pick up every pixel, one by one and look into the sample in the corresponding place and take the value. After this, the graph is stored and when the function repaint is called, the entire graph is redrawn. I added a vertical line corresponding to the playing position and the red graph corresponding to the last slider movements. This kind of graphic was found in the Java Sound API Demo, I had first to understand the code before modifying and using it.

3.2.3. FFSlider and amplitude envelope

To make the Slider following the sound, we had to use an alternative system. In fact, when the clip is charged after selection of the sample, this one is read a first time to draw the graph and in same time to create a tab containing highest points. The problem is the frequency of point variations, positive then negative. A sound file graph makes too many variations of amplitude to be followed by the slider. To solve this, when I read the graph, I translate every data in positive value and finally I make a new tab containing the envelope.
That's the variations the FFSlider will show.

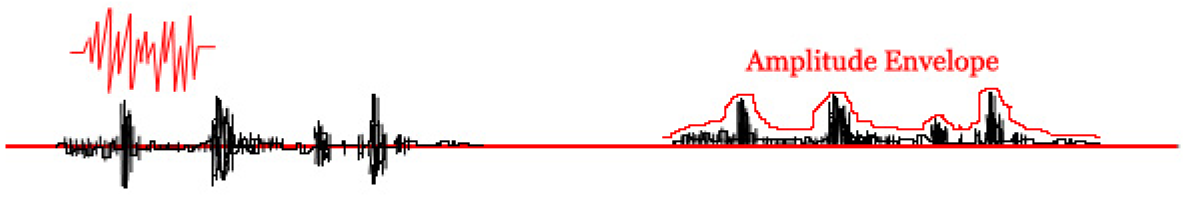


FIGURE 3.4: AMPLITUDE ENVELOPE

3.2.4. FFSlider and Listener

There is one problem that occurred when you send a lot of data to the slider like to make it following the beat. Because of the constant move of it, it's difficult to know if the user is moving something. To solve this technical problem, I decided to use a constant recording on the slider moves. When the play starts, the slider follows the first amplitude envelope of the sound. During this same play, I record the slider movements and replace the old tab containing the amplitude envelope. Because of speed variations, the new values token are a bit different than the previous tab. The slider is already moving to another position that why we can see on the graph that the red part is always moving and loose accuracy.
At the end of every loop, the new tab containing the slider values is sent to the mixer. (Song.class) During the playback, the mixer compares this tab to the sample and if the slider point is under the sample point, this part won't be played.

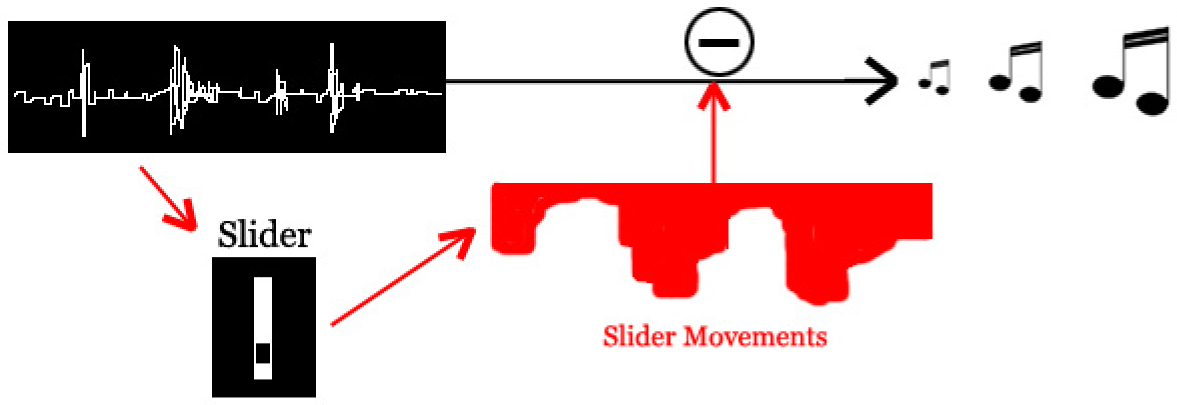


FIGURE 3.5: SOUND MODIFICATION

Another system to represent the beat variations was added after this. In fact, using the amplitude envelope was a start and the result made an effect not enough fluent and the feedback was not enough accurate. The slider still moved to fast and it was difficult to make a link between the music and what the slider was showing. We needed an additional filter. The system I used was to select only the highest point of the graph and realise a king of fall effect for the slider.
In other words, when the music starts, the slider goes up with the first beat. After this, it will fall down in a constant and defined speed but if the new beat is higher than the current position, it will go up again.
Let's take an example. Let's say the first beat is the highest point of the sample and the next beat is zero. If I only play it on the slider with the amplitude envelope, it will be too fast to see on the FFSlider because it goes directly from the highest to zero. With this filter, the FFSlider won't follow the beat and it will go down slower giving enough time to feel it.
This effect is really simple to follow and give a good feedback because the user is already used to this representation of the beat. This concept is taken from a simple equalizer like the one you have with 'Winamp'. Take a look on, you will see it doesn't really follow the beat and show a delay where it goes down slowly.

3.3. Class organisation

This is a simplified visualisation of class organisation to give an overview of how the project works. As you will see, some class groups have many options in same time like Listener Mixer, which provide Pan, Gain and Rate. It has been done in this way because it all has the same behaviour.

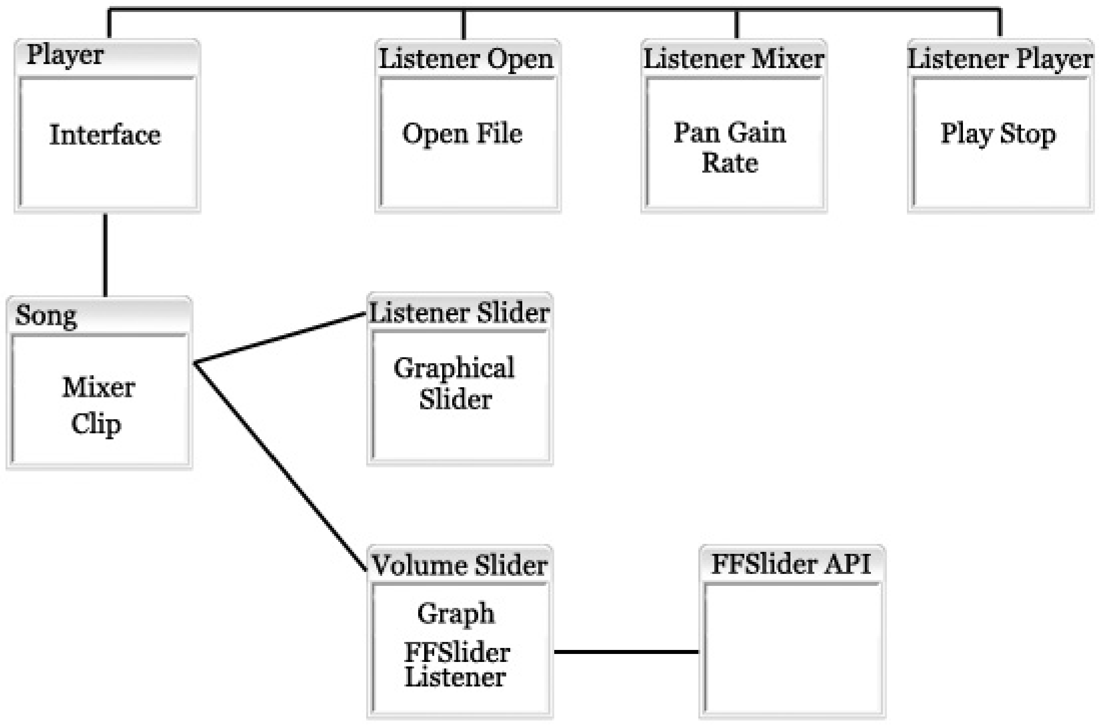


FIGURE 3.6: SOUNDDEMO CLASS ORGANISATION

3.4. Difficulties

3.4.1. JPanel and FFSlider

The first difficulty I had was to start using the FFSlider API. In fact, I started to create a specifically interface including the graph and all other features but one problem appeared. When I added the FFSlider into a special JPanel, I had troubles to see it working. The physical part worked but the graphical slider representing the FFSlider on the interface didn't.
Actually, it was the FFSlider API problem I explained earlier with the copy problem between the creation of a FFSlider and the use of the Panel Board. (See 2.3.3 Sliders Panel)
So I had to work first on the API before keep going on this project. It was a good thing to understand better how the API works and to get use to it.

3.4.2. Volume Slider Class

The second problem I had was more about the optimisation of the application. As you can see on the class organisation, "VolumeSlider.class" is made of two different subjects, which are Graph and FFSlider Listener. Indeed, at the beginning, I had two different classes separating these parts. Both had to be stated in a thread. Redrawing the current sample position in the 'graph.class' and simulate the slider handle fall in 'SliderListener.class'. Moreover, because of the link made between the graphic, the sample and the slider to analyse, modify the sound, show the slider position in the graph, it was easier to make them work together.

3.4.3. Graphical part of the Slider

Another problem occurred to create an alternative interaction using the graphical slider this time. In fact, the Jslider is always moving, following the sound and its value is also always changing. The problem here, is to catch the handle because the normal behaviour of a Jslider with the event OnClick is not to come under the mouse selection as we could think. Even if we hold the click, it will only come tick by tick.

To solve this, I overwrote the event 'OnPressed' to make the Slider comes under the mouse selection. To do this, I had to use the mouse position, which is given from the main Panel and minus the difference with the Slider Panel. After this, you just have to minus slider height with the mouse position respecting the proportions.
Finally, you just have to repeat this operation during the mouse dragged and the slider will follow the mouse.

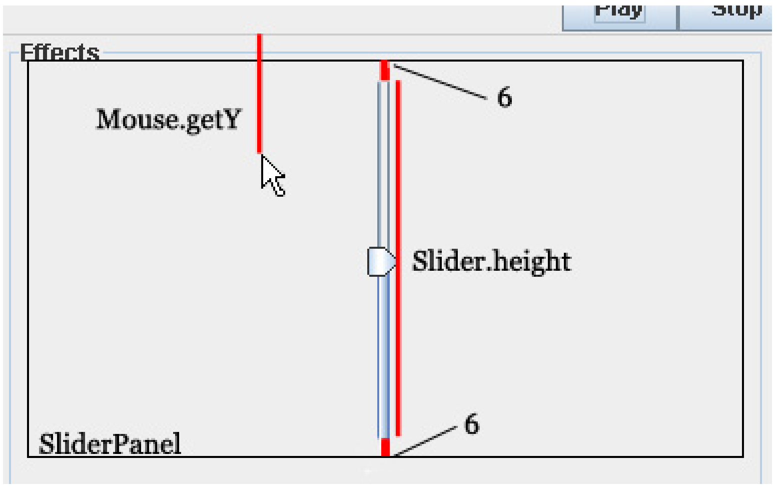
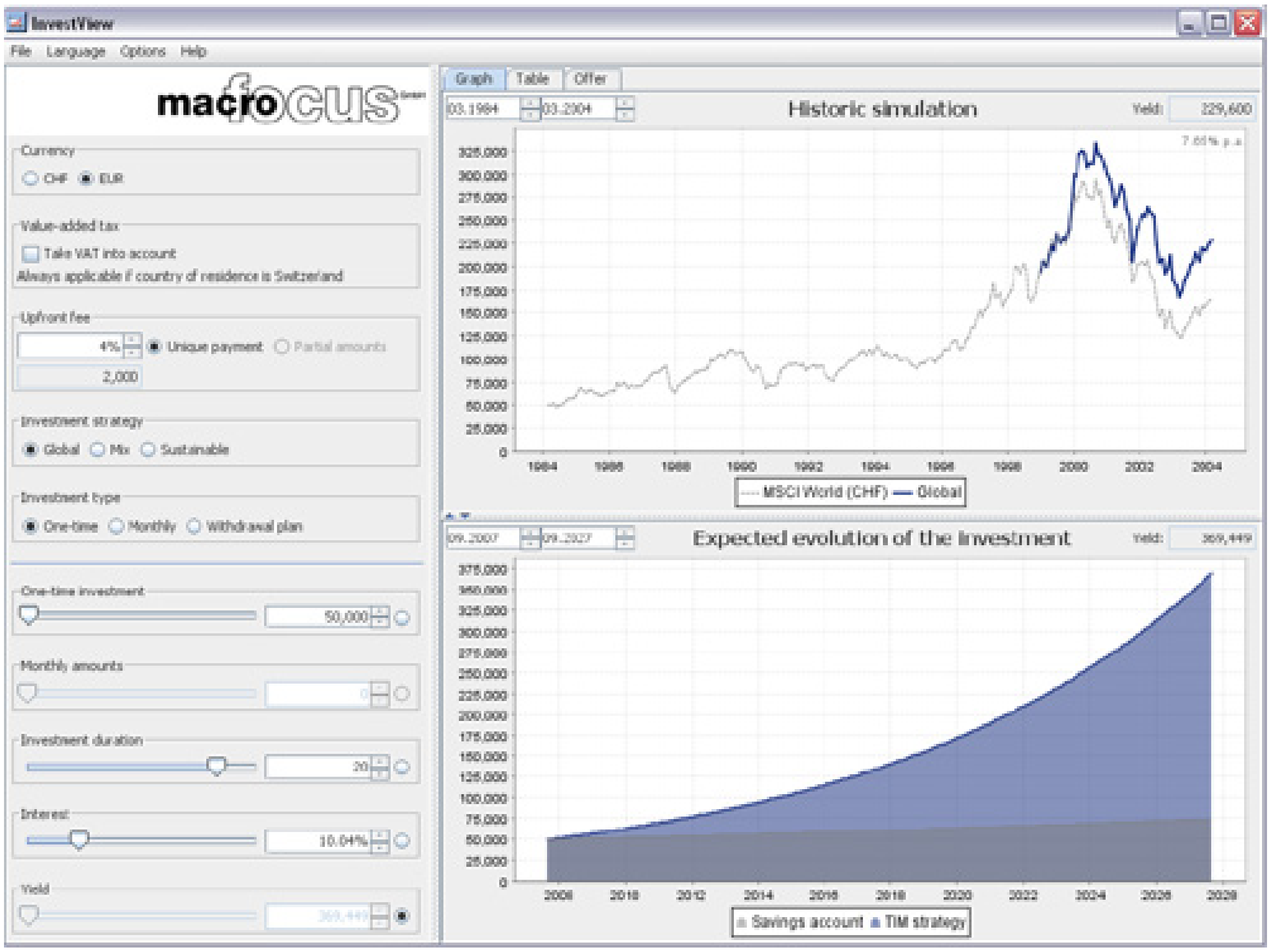


FIGURE 3.7: MOUSE POSITION AND SLIDER POSITION CORRESPONDING

4. InvestView Implementation

4.1. Presentation



Currency: ( ) CHF [✓ EUR]

Value-added tax: [ ] Take VAT into account
Always applicable if country of residence is Switzerland

Upfront fee: 4% [✓ Unique payment] ( ) Partial amounts
2,000

Investment strategy: [✓ Global] ( ) Mix ( ) Sustainable

Investment type: [✓ One-time] ( ) Monthly ( ) Withdrawal plan

One-time investment: 50,000

Monthly amounts: 0

Investment duration: 20

Interest: 10.04%

Yield: 369,449

Historic simulation: 03.1984 – 03.2004, Yield: 229,600

Expected evolution of the investment: 09.2007 – 09.2027, Yield: 369,449

FIGURE 4.1: INVESTVIEW INTERFACE

InvestView is a financial calculator created by MacroFocus in Java and used to simulate savings plan, capital amounts or simply tell you when you will be millionaire.
Using graphical sliders such as One time investment, Monthly amounts, Investment duration, Interest and Yield, you can simulate your own plan and have a view of the expected evolution of the investment.
Because of this use of many sliders, we wanted to try to implement this application replacing the graphical sliders by a FFSlider Box. The goal was to know how difficult is it to replace a JSlider in a project already finished. Indeed, the FFSlider box has a commercial potential for saving time using many sliders instead of one, that’s why it has to be easy to implement in any external project. This exercise was a good second step in my internship and taught me how to use several sliders in same time.
It took me **one month**.

4.2. Use, Launch and compile the application

InvestView that is a commercial application couldn't be given with all the sources and so I only had the classes using Jslider. Because of this problem, it was not possible to launch it from a compiler like Eclipse, classes were separated into different .jar and the classes I was working on were also supposed to be in a .jar to be launch with all classes needed.
This jar document was named InvestView.jar and was also supposed to contain the additional library needed by the FFSlider such as SliderAPI and Jd2xx. It was really time consuming because I had to develop using Eclipse to compile. After compiled, I had to add the new files to investview.jar in the folder named Control and launch it with commands.

<u>InvestView.jar :</u>

- Jd2xx (Drivers needed by FFSlider)
- SliderAPI (FFSlider Library)
- Com
    - Macrofocus
        - Application
            - Invest
                - Compiler
                - Data
                - View
                    - Timvest
                        - Control (Jslider classes)

To work on this project and change the Jsliders, we only needed to work in one class called 'SimpleControlView.class', which contained all Jslider implementations. I also added a new class for the FFSlider listeners.

4.3. Implementation

To implement this application, the first idea could have been to simply replace the Jslider class by FFSlider.class but it's not possible because it's built in a different way and has to be declared with the FFBoard. In other words, a FFSlider has to be declared depending of the board.
The first step was to declare the FFBoard and FFsliders as it is done in the presentation of Slider API. Then I defined the FFSlider in the same way it was done with the first development. The Jsldiers used in this application use a 'BoundedRangeModel ' which is an object made to define maximum, minimum and value for graphical objects. It can be used for Slider, Spinner or Scrollbar. The advantage of using this kind of object is that it allows you to change the value of them using it instead of the slider. Like in this project, when you declare for example one Jslider and one Jspinner with the same BoundedRangeModel, the value of these objects will be both changed when you change the value of the model.

After this step, I did the interaction type declaration. For this application, which is used in financial and only use Jslider as a value selection, the best would be to make a “tick mode”. This mode makes you feel a resistance every x values like a radio button to select a range of options like switching between the cdrom, radio and off.
Finally, you need to call the Slider Listener to change the Bounded Model when the slider is touched and add the Slider to the interface.

4.4. Difficulties

This project was not supposed to be difficult or time consuming but many problems appeared during the development.

4.4.1. Tick Mode

As I told before, tick mode is a step-by-step mode where the user can feel small resistances when he tries to push the slider. To use this mode, you have first to fill a tab with intervals you want to apply a force. Then you just have to define the slider selecting the force you want when the user push the handle.
For this project, steps were defined every twenty points that it means only fourteen step on the slider.
The problem here is that when you select more steps, the slider start to have some difficulties and at the end you don’t feel anything. In other words, if you use more than twenty steps, this mode will not work. If we want to improve this mode, augmenting the number of step would not be the good solution. In fact, this feedback is really not accurate. I mean the slider can only have 250 different positions. When a force is applied on a tick, it needs an other tick to say: if the user is on 166, apply the force to go back on 165. In conclusion, the maximum step mode could have 125 steps at the maximum. If we want to use this mode, the slider would have to be changed to have more accuracy. For the moment, it’s not possible to use it for an investment application but still can be shown as an interaction demo.

4.4.2. FFSlider Listener

After implementation, the Slider Listener worked from the first time and it looked like it’s really easy to add the FFSlider to any application. But finally after some test, the slider seemed to work for a while and after it stopped by itself or the application completely crashed.
The error which occurs is an “Out of bounds error” that means we are trying to put a value out of a tab. The FFSlider and Jd2xx classes were in the error list at the end and were on a Slider.setValue().
The first idea was it came from the Bounded Model, something I was trying to set out of Bounce.
I checked the ‘Bounded Range Model’ but it was not my code so I tried after to check the maximum and minimum I defined for the slider. Trying to check this, I discovered that the slider didn’t crash with any value but after and x value and only for some sliders, not all.

At this point I was quite stuck, I decided to check the Slider API to see if some strange values were sometimes sent. So I made a lot of tests, implement the entire use with out messages and change some functions but it didn’t change anything.
The next step was to check the hardware because it could have been a kind of data overflow coming from the slider, which could stop everything. After discussion with the hardware builder, this problem was supposed to be managed.
Finally, I tried to change a bit the Slider Listener to make it slower and only use one value of ten to have less work on every loop. After this changes, the slider worked much better and it was possible to make some demos to customers but it still crashed after a while.
Having no more idea to solve this problem, I wanted to try if this problem occurred with the basic application which work without the FFSlider. I launched it with command and I realised that the same error occurred on Jslider but without a crashed. Obviously, this error didn’t come from my part as I thought and so I contacted the developer of MacroFocus and explained the problem.
He told me than this problem was fixed now and he gave me an update.
After modification with FFSlider, this application work properly now.

Conclusion, we can say that the FFSlider is easy to implement on a finished project.

5. Bounce Slider Development

This new project is the first version of a sound producer application and the main part of my internship. It took me three months, I worked alone on it and it needed a previous knowledge on Java Sound and FFSlider to use a slider box.

5.1. Concept

Johan Sandsjö, an interaction designer, in a company named Hidden based in Gothenburg, found this concept and we worked on it as a collaboration. My work was to develop the bounce slider application following this advices:

Bouncebox (option to change) is an application for the FFSBox force feedback input hardware. The purpose of the application and research is to provide a tool for exploring perceived physical characteristics of sound ('objects').

Object: Explore Physics (of sound), Music theory, Apparatus for Exploration and Expression.

Related areas: Sound, physics, bilateral manipulation, tabletop software, multimodal interaction, force feedback, movement, psychoacoustics, composition.

The software produces audible sound objects created by stating object mass (timbre) and the gravity (1 = earth gravity) in the program environment. Each objects is then assigned to a physical slider to represent the object in the physical world. The weight and size of the object sets the force and the timbre (from low bass to high pitched treble) of the sound objects. After assigning a sound object to a slider you can either drop or bounce the slider button downwards. The sound objects become audible as the slider button touches the base of its cut.

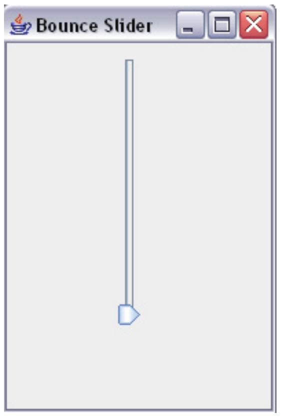
Modulating the first of the grouped sliders generates a low bass drum sound and sets the loop length of the session. Experimenting with the other sliders creates new layers with new sounds, but still within the sat loop length. The stand-alone slider sets the overall tempo of the composition (stretching the loop length). The gravity parameter in the program does also controls the pace the object bounce, and the descending height and intensity of the bounce.

The software should provide basic characteristics of the box, to be suitable as a demo of its basic functions. As the device mimics nature there is a lot of programming features to control, as making the sound change in pitch for each bounce etc.

Johan Sandsjö

5.2. First Model

5.2.1. Presentation



The first model was made of a single graphical slider to give an overview of the concept and check if the idea I had of it was the same than the concept. When you hold the handle and release it from the top, this one will start falling and playing an instrument note until it touch the bottom. The handle is like a ball, which fall in a constant speed named gravity, and the instrument note is the sound of the fall. When the ball touches the ground, it makes a bounce and the ball go up again in a position x depending of the bounce value. The bounce value is a percentage of lost between the last and the new start position. In other words, the new position is a percentage of the last one. What we can hear is a note playing faster and faster like the sound of a balancing spring.

FIGURE 5.1: BASIC SLIDER

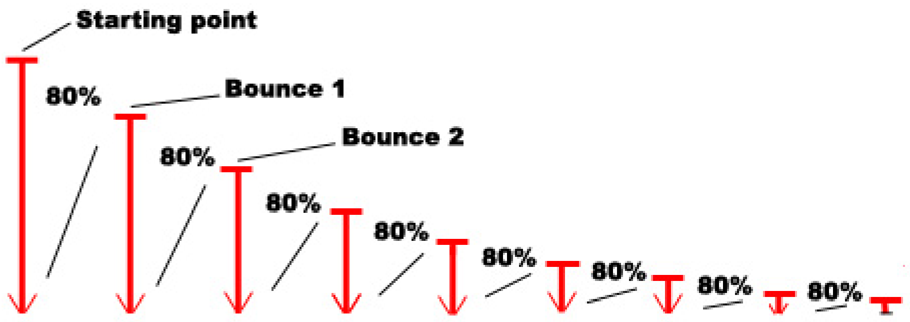


FIGURE 5.2: BOUNCES PERCENTAGE

5.2.2. Java Sound and Midi

Midi (Musical Instrument Digital Interface) is a Java interface which simulation instruments. Theses instruments are loaded from a sound bank where you can find 127 different instruments and a plenty of drums. This system works with a synthesiser, which loads the sound bank and allows playing them into a channel. One synthesizer has 16 channels and can play multiple notes in same time. It’s called multi-timbre. When you want to play an instrument with a special note, you first have to load this instrument and after to create an event “NoteOn”, saying which note and velocity you want to play. It’s the same to stop playing, using an event “NoteOff” saying which note.

```
while (thread != null) {
                  if (play) {channel.noteOn(note, velocity);
                        }
                  thread.sleep(duration); //thread for duration of the note
                  channel.noteOff(note);
```

Every channel has a special used depending of the timbre of the instruments. For example, drums are used on the channel number 9 because it's a special device, which can't be played in the same way.
Synthesizer also has a lot of controls to change the reverb, pan, vibrato and a lot of other filters, which makes a lot of possibilities.

5.2.3. Playing Style

Many ways and variations of playing the note were found and there is a plenty of interesting effects to do. Three different possibilities were pre selected and shown to the customer.
The first style was the first I presented before, which is a simple play of the sound with no effect.
The second was a quite interesting effect built on the pitch, which is a variation of the frequency. Every bounces selected a special pitch of the all bounce, depending on the start position. It sounded like an acceleration in the sound of the first spring concept. (1)
The last one was a variation of the pitch from the maximum to the minimum for every bounces.
This third one was really interesting to hear and it made a good psycho acoustic sounds. (2)

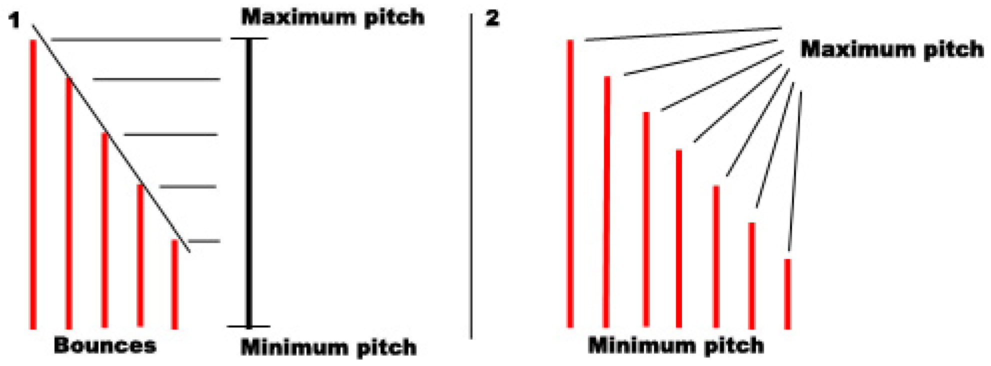


FIGURE 5.3: VARIATION OF PITCH

Finally, after concerting with Johan, we decided that this kind of variation was really interesting but it would be difficult to make a sound maker using a basic play like this. So we chose the simple play but not even the first I described. In fact, the sound of the ball falling was wrong and it was supposed to be the sound of the bounce.
So instead of starting the sound when the handle is released, I modified it to play the sound when the position is zero and now the duration of the sound doesn't depend of the start position, it stops a bit before the next bounce.

5.3. Prototype

5.3.1. Presentation

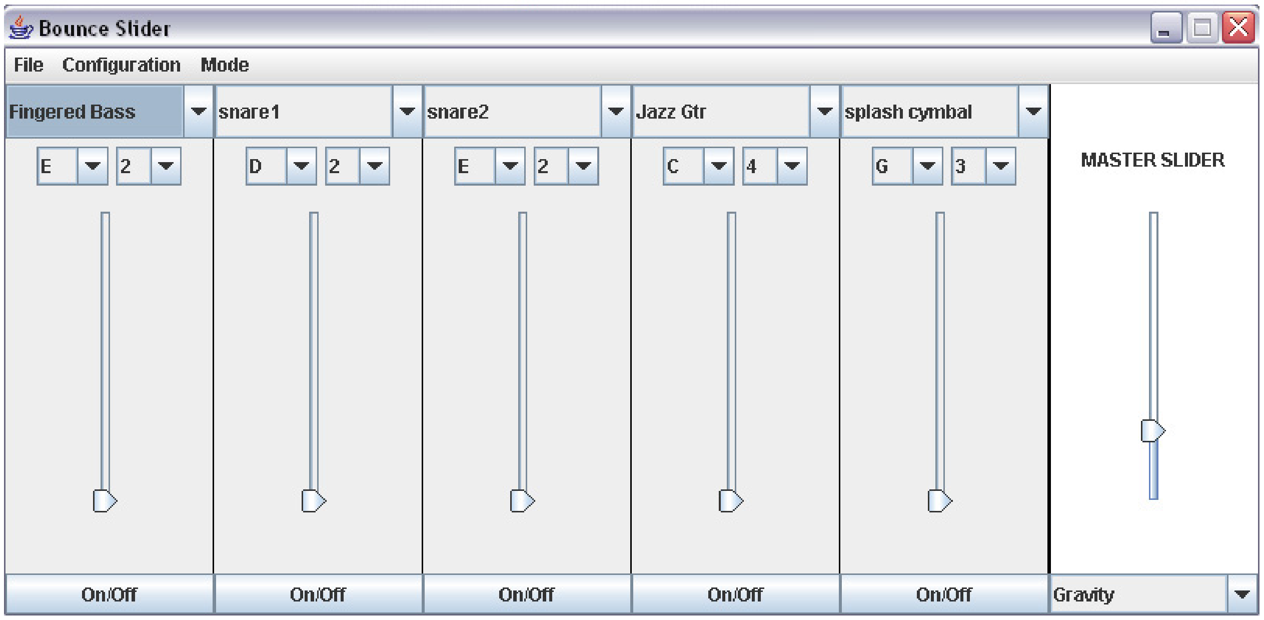


FIGURE 5.4: BOUNCE SLIDER PROTOTYPE

This first version is made of five sliders to play sounds. You can select the instrument, note and octave you want to play and turn the mute on. An additional master slider is built to select two different controls using a multi box. The Gravity, which is the speed of bounces and the Bounce, which is the percentage of lost from one bounce to another.
There is also a menu to save and load a session, which allows you to replay your composition. A configuration save and load to only make a selection on the instruments you want to start with.
And a play mode to make automatic loops at the end of bounces.

5.3.2. Instrument Choice

The instrument list is built with the synthesizer, which gives information about all instruments and helps to build the multi box to choose them. One particularity with instrument list is that it's made of 127 instruments like piano, guitar, bass, violin and strange sounds. Drums are a special kind of instruments with Midi. 57 different drums are provided and when you select one, it's not played on a classical channel.
So when you select it, the synthesizer changes the channel to the channel 9 and the selection of notes is disable but still visible because it's not possible to change a drum note with midi. To show this difference, I made a change in the presentation of instruments in the multi box. Drums are shown at the end and in major letters.

5.3.3. *Master Slider*

Master slider is a global slider, which changes features for all the sound sliders. To make a difference, it has been separated of the other ones with a thicker line and putted in white colour. About the position of this one, the choice was to respect sound mixer tables and it's used to be on the right. A multi box allows the user to change two features such as gravity and bounce. When you change a feature, the changes are made in Synthe.class, which control all the sound sliders working with ChannelNote.class.

5.3.4. *Usability improvements*

After the first tests, one problem appeared about the master slider. User didn't really see the possibility to change between gravity and bounce using the multi box. The fact is, they don't associate the multi box with the master slider. In the goal of improve this part, I created three different visual master sliders and I made user tests to choose the best one. The fourth is the final choice resulting from the user test.

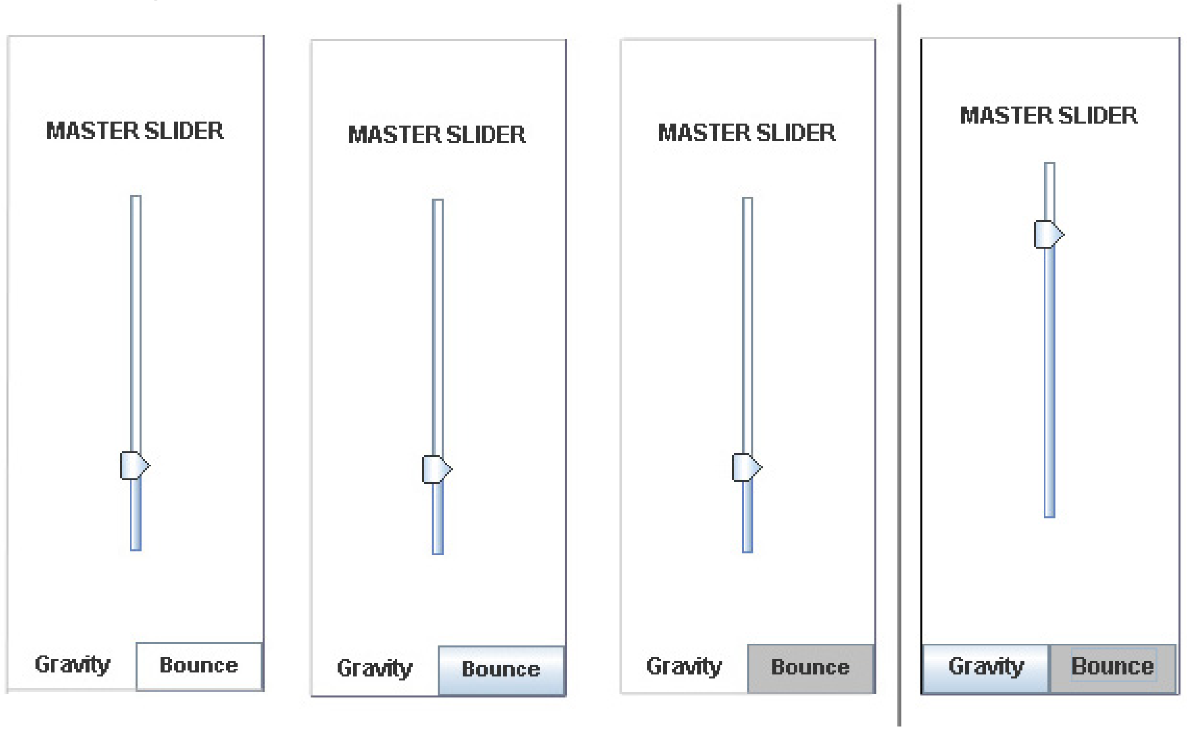
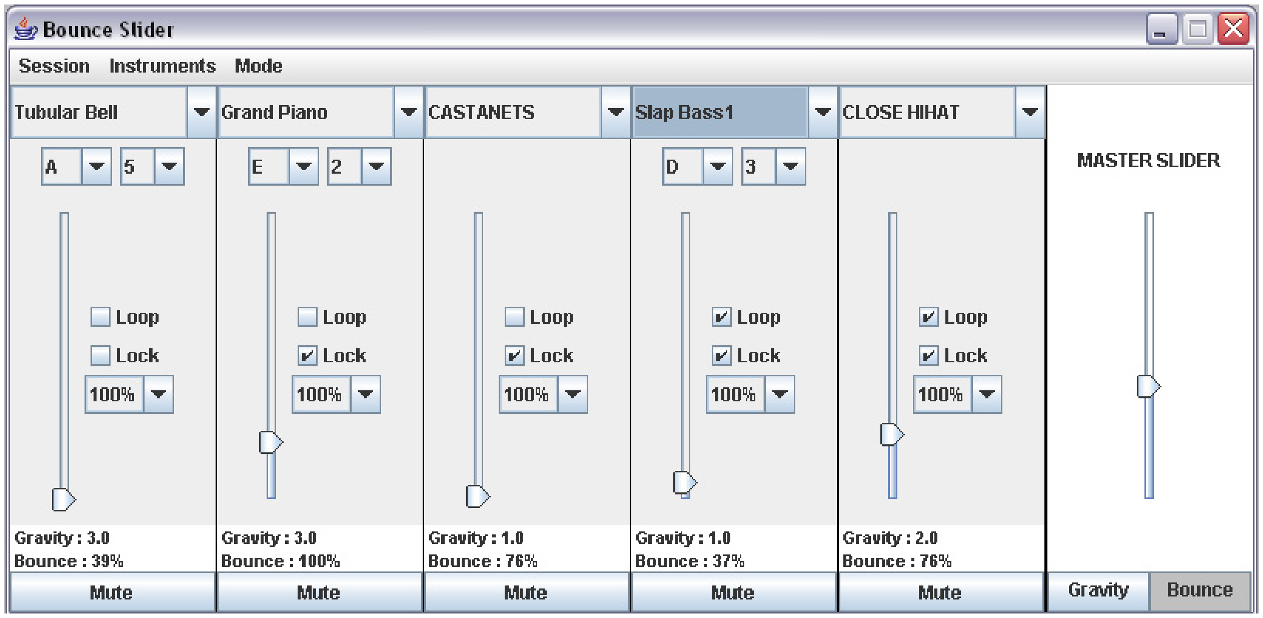


FIGURE 5.5: MASTER SLIDER STUDIES

To group the master slider with the selected button, I wanted to make like a TabbedPane and show them in the same colour. After tested them on users, it shown that it was still difficult to know which one was selected because in any case, user thinks the selected button is not the white. White is not a good colour to show something selected. That's why the last one is not made in white because I had to leave this idea. Finally, I chose to do the same than the mute buttons, a blue button to select and grey colour when it's selected. By this way, user will make a link about the behaviour of all buttons and try to do the same with the master slider.

5.4. Version 1.0

5.4.1. Presentation



Tubular Bell, A, 5, ☐ Loop, ☐ Lock, 100%, Gravity : 3.0, Bounce : 39%

Grand Piano, E, 2, ☐ Loop, ✓ Lock, 100%, Gravity : 3.0, Bounce : 100%

CASTANETS, ☐ Loop, ✓ Lock, 100%, Gravity : 1.0, Bounce : 76%

Slap Bass1, D, 3, ✓ Loop, ✓ Lock, 100%, Gravity : 1.0, Bounce : 37%

CLOSE HIHAT, ✓ Loop, ✓ Lock, 100%, Gravity : 2.0, Bounce : 76%

FIGURE 5.6: FINAL INTERFACE

At this moment, the application already works and the goal now is to make some improvements and add some features to have a better plenty of options to produce great sounds. As you can see on the interface, there is quite a lot of new buttons to make it more flexible. The major improvement is the addition of special features if you want to custom only one slider to have more possibilities of new sounds. When you use a slider, changing the bounce and the gravity with the master slider, you can now access to more options. If you check the Lock mode, this slider will keep the gravity and bounce values and by this way you will be able to create five different sound effects playing in same time. Before this version, it was only possible to create a loop on all sliders in the same time. Now, using the Loop mode check box, you can create a special loop on only one slider and keep going with manual bounce effects on the other ones. This use of special features per slider needed more visual indications and that's why I added two labels on the bottom to show which gravity and bounce values are actually used by this slider.
Another change was made during user test about instrument selection. As I said before, when the user choose a drum, it's not possible to select a note or an octave. To show this, I put note and octave box as disable but still visible. The problem was that the user still tries to change the note and don't really know what to change to make it enable again. To fix this, I decided to not show these boxes when a drum is selected. That's why you can see on the interface, there is not always a note choice and it's also more visible because of the major letters to show a special use with drum.
The latest innovation is about the multi box you can find next to every slider to select a percentage, which is the value of the gain or volume you want for the sound you're playing.
That's an important part if you want to realise advanced music samples.

5.4.2. Class Organisation

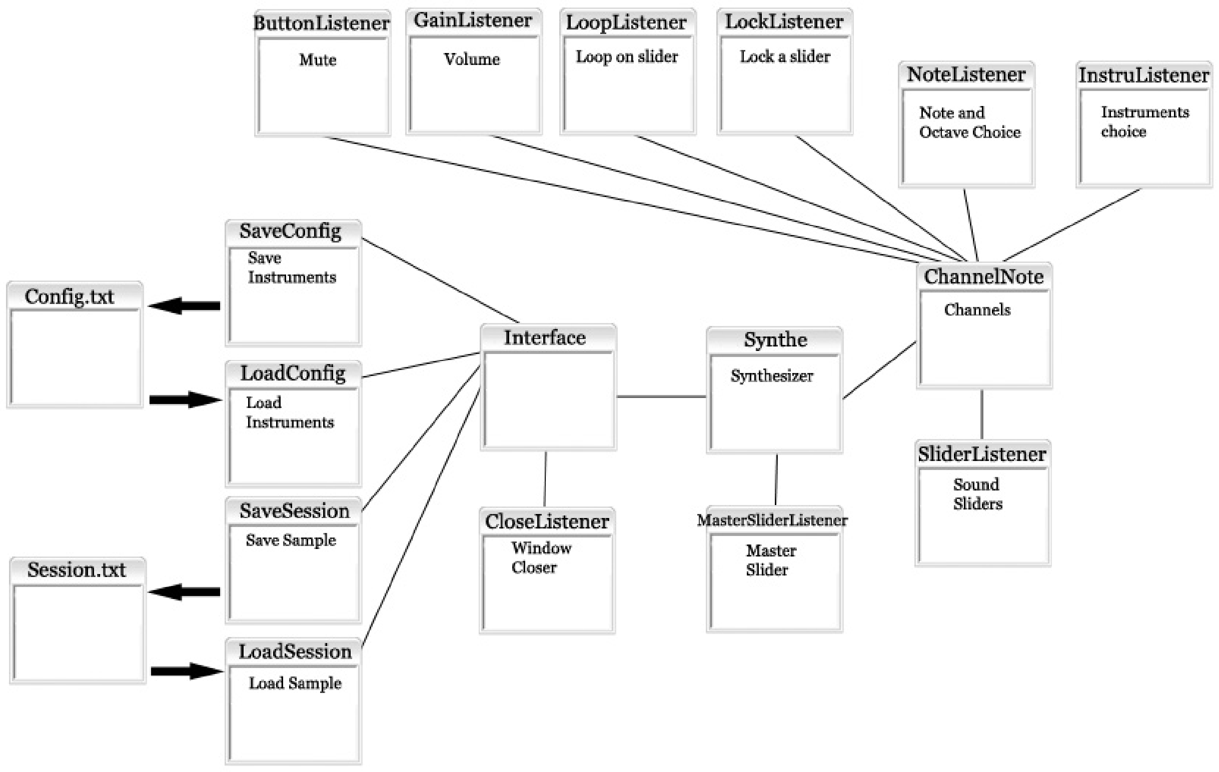


FIGURE 5.7: CLASS ORGANISATION AND DESCRIPTION

As you can see on this picture, the main part of the organisation is about ChannelNote.class, which provide all services needed to manipulate sounds. All option classes are linked with this one and FFSlider are directly connected to it. For the master slider, which has to make chances on all sliders, this one is connected to the synthesizer and will change all features in same time.
Another part of this graph concerns the backup side to help the user in his compositions.

5.4.3. Load and Save

This application allows you to save two kinds of information. You can save information about instruments, notes and octave as a configuration if you want only to have basic features and start to play quickly. You can also save the session if you want to open a previous composition you did the last time and play it directly.
To save and load features, I used a simple text document where I start with "BounceSliderSession:" or "BounceSliderConfig:" to check if it's the good type of file. After have checked this comparing letters, I create a list of number representing for every slider the values of instrument, note, octave, mute, loop, lock and current slider values about gravity and bounce. For the master slider, I store the global value of gravity and bounce.

<u>File Backup example:</u>

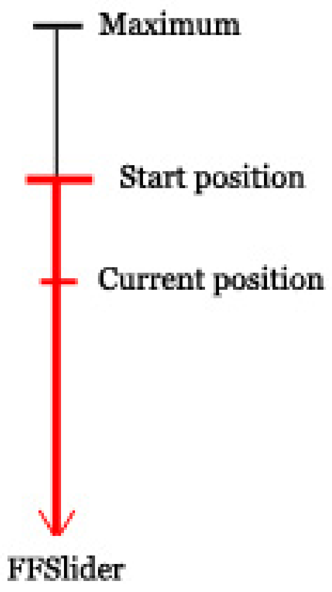
```
BounceSliderSession:          //File Type Session
9                             //note
3                             //octave
146                           //instrument
0                             //origin
10                            //gravity (0-30)
80                            //bounce (0-100)          Slider 1
0                             //loop (boolean)
0                             //lock (boolean)
0                             //gain (0-200)
1                             //mute (boolean)
0                             //playing?
4                             //Slider 2
3
…
…                             //Slider X
…
…                             //Master Slider
10                            //Master gravity
80                            //Master bounce
```

To load a file you just need to make loops and read information for every slider.

5.4.4. FFSlider interactions

A new particularity comparing with the two last applications is the possibility to launch it with any FFSlider connected. It means a more flexible interface using a double development for FFSlider and Jslider. And because the FFSlider box was not finished, I worked on two sliders and so the software detect how many FFSliders are connected and build the interface in consequences.
As I said in the first part about the sound modification software, it's difficult to detect when the user is moving the handle if the slider is already moving. Nevertheless, two types of interactions was found depending of the use you want to do of it.

The first and simpler interaction is with the normal slider behaviour. When you release the handle, the sound starts and the slider goes down until it touch the ground and will go up again depending on the bounce value. At the end, the slider stop and you can restart it by holding the handle again and release it in x position. Because of the fact the slider always go down, the user would tries to push it up and that's what we are trying to detect.



When the user release for the first time the FFSlidier, this start position is stored and the slider starts to go down. That's the current position.
To change the loop and make a new start position, the user has to push up the slider again. A slider listener looks if the new current position is upper than the start position and only if it's upper, it will stop the current loop and wait. During this time, the slider is waiting for the current position to be fixed somewhere. In fact, it is storing the maximum value of the current position until the slider start to go down again.
When the current value is downer than the maximum found, start position becomes maximum value found and the loop starts.

FIGURE 5.8: FFSLIDER CHARACTERISTICS

This method is good in almost every case but there is no fact. If you select a position, you have to wait until the loop start in a lower position than where you want to start. It means, you can select a position lower than the previous. This case is really a problem especially if the bounce value is 100% because it means the loop will be constant and the start position will change only for a bigger value. If the start position becomes the maximum, the slider is stuck.
To correct this exception, a new behaviour was found. There is a timer looking at the slider and during the play, if the value stay the same for more than one second, the start position changes and the loop restarts here.

For the Jslider the listener is easier to configure, you just need a click one the slider and the loop will stop. You just have to select the new start position.

5.4.5. Usability improvements

Some changes have been made during the development concerning the usability.
At the beginning, when user pushed the mute button, this one changed its label to indicate the new value of the button to select mute off. This title could be confusing with the state of the button so I changed this for a button with a simple label mute and a grey colour when it's pushed to show that mute is activated. Before this, an additional visual effect disabled the Jslider because if mute is on it's not needed to control it but it was a bad idea. The user tries to use it when it's disabled.
With this problem, we can add what I said before about the drum selection, which disappears now when selected and the problem with the master slider and its buttons.

5.5. Difficulties

At the end of the development, we found a problem about the synchronisation of the sounds that sometimes didn’t follow at all.
When the speed slider selected started to be quite fast, some sounds were sometimes not played or stopped. This problem comes from Midi event because when the loop restart in a too small segment, ChannelNote, which started the «NoteOn», is still in sleep when a second “NoteOn” comes and so the first sound doesn’t stop at all.
To solve this problem, I reduced the sleep time for ChannelNote, which wait quite a lot when it’s not playing. The sounds can be played a bit faster now but it’s resource consuming.

5.6. Future work

This project is the first version of the bounce slider, which works properly and can produce really interesting music. There is a plenty of possibilities to try and make the music is quite easy. Indeed, the use of the FFSlider gives good feedbacks and its use could be applied for many other concepts in music.
One modification could be the possibility to record the session as a sample in MP3. I tried to do it but I didn’t have enough time left so I tried to do it recording directly from the speaker port but it was no possible to select it as a SourceDataLine.
The major improvement should be to not use anymore midi sound but music sample for a better flexibility and a bigger range of possible sounds. It could be great also to have the possibility to use external devices like a drum box we could control via Midi event system.
As I said about the playing style and the way to modify the pitch during the play, I shown that there is a lot of new effect we could add to make psycho acoustic music and it could be great to add a new part in the interface to select some filters and create some amazing sounds.

The most difficult future work I think will be to modify and adapt this application to the new slider box, which doesn’t use the same chipset and so doesn’t work like the previous one. Some problems appeared at the beginning of the conception and maybe, the Slider API will have to be modify one more time. Anyway, there is still a lot of work on this application, some parts can be optimised like the play of sounds or the slider listener. Indeed, one slider listener is quite resource consuming and I am not sure the actual system can follow with six FFslider.

6. Internship Conclusion

This internship was supposed at first to be based on C/C++ development but finally Morten Fjeld told me about the FFSlider based on Java, a language I prefer and I would like to work with in the future. Because of the new digital version of the FFSlider, new applications were going to be developed and it was better for me to take part of a new project. So I started this project since the beginning and I had a good overview of the development steps of an application. Moreover, I learnt how to organise myself and organise the application architecture to be the faster possible. Working on Bounce Slider taught me how to work when you have to develop an application for a customer. I learnt how to meet him, present the new features and discuss about the changes or improvement to do. In the same domain, the laboratory often makes presentations for potential customers or sponsors, and these moments were also a good training in the social way and really good to improve my English.

I discovered an entire new domain I never worked on, Java sound and I think it was really interesting and I manipulated a lot of new concepts, I had fun.

I also worked on InvestView, which was a project already finished and I had to take part of it, use and understand code that was not mine as it's made in a company.

Another important part is the teamwork side. Even if I worked alone on my project as everybody did in the laboratory. There was a really good teamwork and we discussed a lot to share and solve different problems together. That's an important part of my internship. The social way was the main goal of it because it has been made in a foreign country and I learnt so much.

So as a conclusion, I can sincerely say that this internship has been a so interesting experience in everyway. I worked on many domains and I improved everything I wanted to. My English level is now enough good if I want to work in an English company. My Java skill is much better and I have a good working experience on. Bounce Slider is a great new project and I will follow the next development steps and maybe write a scientific publication about.

Content Of The Attached CD

1. Java Development Classes
    1.1 Sound Modification Demo
    1.2 Invest View
    1.3 Bounce Slider
2. Final Version of Bounce Slider
3. Videos of applications and slider behaviours
4. Pictures
6. Report
5. Needed drivers to install

These applications are shown only to give a code example. Sound Modification Demo and InvestView projects can’t be launched without FFSlider connected.

References

[1] Ali Shahrokni, Julio Jenaro, Tomas Gustafsson, Andreas Vinnberg, Johan Sandsj and Morten Fjeld. One-dimensional force feedback slider: going from an analogue to a digital platform.

[2] Huber R. Kretz, A. and M. Fjeld. Force feedback slider: An interactive device for learning dynamic system behaviour.

[3] Joakim Almgren, Richard Carlsson, Henrik Erkkonen, Jonas Fredriksson, Sanne Møller, Henrik Rydgård, Mattias Österberg and Morten Fjeld. Tangible User Interface for Chemistry Education: Portability, Database, and Visualization

[4] Andersen, T.H.; Huber, R.; Kretz, A.; Fjeld, Morten**.** Feel the beat: direct manipulation of sound during playback.
[5] Julio Jenaro Rodrigues, Ali Shahrokni, Martin Scrittenloher, Morten Fjeld. One-Dimensional Force Feedback Slider: Digital Platform.

[6] Martin Scrittenloher, Selina Schwager and Morten Fjeld. Haptic Support for Intimate Remote Control Communication.

[7] Piazza, Tommaso; Fjeld, Morten: Ortholumen (in press, 2007): Using Light for Direct Tabletop Input.

[8] ***ARToolKit developer homepage***. ‘http://artoolkit.sourceforge.net/’.

[9] The java sun api for the jslider. ‘http://java.sun.com/javase/6/docs/api/javax/swing/JSlider.html’.

[10] Java Sound Programmer Guide.‘http://java.sun.com/j2se/1.5.0/docs/guide/sound/programmer_guide/contents.html’

[11] Java Sound Ressources and examples ‘http://www.jsresources.org/’

[12] Java Midi List of Notes‘http://www.harmony-central.com/MIDI/Doc/table2.html’

[13] MIDI channels, voices, timbres and Modes. ‘http://www.philrees.co.uk/articles/midimode.htm’

[14] Ftdi necessary driver installation. ‘http://www.ftdichip.com/Drivers/CDM/Win2000/CDM_Setup.exe’.

[15] Eclipse software development. ‘http://www.eclipse.org/’.

[16] jd2xx development project. ‘https://jd2xx.dev.java.net/’.

[17] Java Resources and tutorials. ‘http://java.sun.com’

List Of Figures