Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Creative Robot Composer

Semester Thesis

Manuel Lippuner

lmanuel@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Manuel Eichelberger
Prof. Dr. Roger Wattenhofer

January 2, 2019

Acknowledgements

I would like to thank my family, for all the valuable inputs regarding the usability and simplicity of the graphical user interface. This advice helped me to make the GUI more intuitive.

Abstract

This thesis builds upon the existing robot composer framework, which provides an algorithm to produce music. This framework has been ported to c++, restructured and extended.
Additionally, a vst-plugin has been developed, which acts as a user interface for the robot composer framework. For the development of the plugin the JUCE framework has been used.
The plugin integrates with existing audio producing software and allows the user to not only generate music, but also to adjust or exchange parts of the composition afterwards.

Contents

Chapter 1

Introduction

This thesis is a successor to different projects that have lead to a Robot Composer framework with a simple GUI and a different robot composer from each project [1, 2, 3]. In this thesis a tool is developed, which lets the user select one of the different composers to compose a piece of music and allows him afterwards more interaction possibilities with the generated music. The tool should allow for a composer work flow. The user is able to modify the generated music and recompose certain parts of it. Such that the tool could be used as a source of inspiration for professional music composers.
To make the tool more accessible to the music producing community, a new front end in form of a plugin for digital audio workstations is developed. That allows, that the user can benefit from the functionality, that is already available in the digital audio workstation. The user can further process, mix or add effects to the generated music.
There are many different types of plugins for digital audio workstations. For this project the vst-plugin has been selected and for this the framework has to be rewritten in c++. This is opportunity to also better separate the structure from the generation functionality. Additionally, the structure can be adapted to support changing the music piece, after it has already been generated.

1.1 Related Work

The Robot composer project is a framework to generate music based on music theory and not machine learning [3].
The framework generates a piece of music with a block structure, such that it supports many popular song structures.
The framework generates general Midi files, which can be replayed with a compatible media player.[1]
There is a simple GUI, which allows to set the input parameters for the Robot composer and it displays the structure of the generated piece.

Chapter 2

Background

2.1 MIDI

Midi is a communication protocol for audio devices. The devices are connected by a midi cable or by a virtual midi bus. On each MIDI bus there are 16 channels and a MIDI device can be set up to only listen to one channel. This protocol allows to automate instruments and control them for example with a MIDI keyboard. The MIDI messages contain information about the pitch, pitch bend, velocity of the key strike, the volume or timing information, that is used to synchronize all the machines connected to the MIDI bus. MIDI instruments are synthesizers that transform the digital midi messages into sound. [4]

2.1.1 General Midi

General Midi is a standard based on the MIDI protocol. A General MIDI instrument is capable of emulating 128 instruments, called programs. On each of the 16 channels one can select a different instrument by sending a program change message on the channel [1] [5].
The robot composer framework uses the General Midi standard. Each instrument that is selected gets its own channel the the composer writes the scores for this instrument to it.

2.1.2 Midi Files

A MIDI file is a digital representation of a music piece. This representation is quite small since it only saves the MIDI events and no audio samples, but it contains all the information needed to be able to generate a score sheet from it [6].
To safe a sequence of MIDI messages to a file, one has to add a time stamp to each message, which can be calculated with the the number of MIDI ticks per quarter note, which is specified in the MIDI file header. The time stamps are

Figure 2.1: Picture of a setup with a software DAW [8].

calculated as follows:

$$timestamp = (quarterBeatsSinceStart) * (numberOfTicksPerQuarterBeat) \tag{2.1}$$

For each tempo change in the piece, one has to add a MIDI message, that specifies the duration of a quarter note in microseconds [7].

$$MPQN = MICROSECONDS_PER_MINUTE/BPM \tag{2.2}$$

2.2 DAW's

In a music studio, there are many different instruments and loudspeakers. The purpose of a Digital Audio Workstation is to act as the central relay station. It records the music from the different inputs and sends it to effect machines or tone generators and finally mixes the different channels down to the output channels. The MIDI output is sent to the tone generating devices, such as synthesizers or sample machines. Audio samples are sent to loudspeakers [9].
In the early days of digital music producing, DAW's have been actual hardware devices as can be seen in figure 2.2. Nowadays most of the times a software program on a recording computer is used instead, as in figure 2.1. Some examples for software DAW's are Cubase, Fruitloop Studio or Ableton Live.

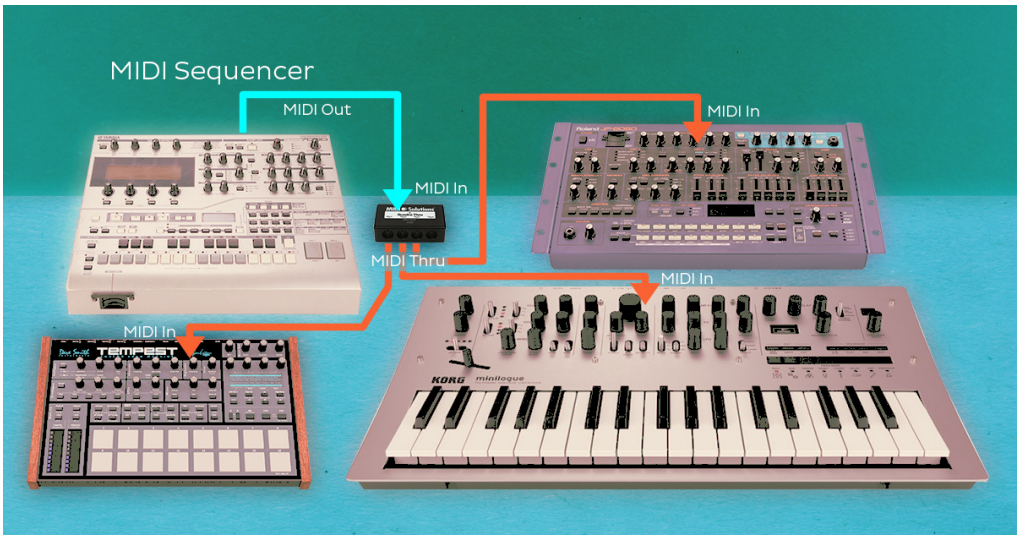

Figure 2.2: Picture of a setup with a hardware DAW [8].

2.3 Plugins

Instead of connecting hardware audio devices, one can add plugins to a DAW that emulate the behaviour of the hardware. The DAW acts as plugin host and sends audio and/or MIDI data to a plugin to process. The processing has to be done in real time and the outputs of the plugins can be connected to other plugins, a recording track or directly to the output/loudspeaker.
There are different types of plugins. A virtual instrument takes MIDI input and creates audio data from it. An example would be a virtual synthesizer or a sample machine, which plays prerecorded samples on command. An other type are effect plugins which processes an audio input to an audio output. There are MIDI effect plugins, which do the same for MIDI data.
Nowadays, there is a huge amount of commercial and free plugins available, such that whole music producing process can be done on a single computer, without need for the expensive specialized hardware [10].
The purpose of the plugin developed in this thesis is to automate the generation of MIDI input data. Therefore, instead of connecting the MIDI inputs of other plugins or recording tracks with an external MIDI keyboard, they can be connected to the output of the RobotComposer plugin.

2.3.1 VST Plugins

Virtual Studio Technology is a plugin standard developed from the German software company Steinberg, which also develops a software DAW called cubase. The VST standard has first been released in 1996 and has since become one of the

most used plugin formats nowadays.
For developers, Steinberg released freely available interface specifications and a c++ sdk, such that everybody can create his own vst-plugins [11].

2.3.2 JUCE

JUCE is a c++ framework , which offers support to create different plugin formats. One can create multiple plugins from the same code base. For example one can create a VST-plugin and a Audio Unit plugin (the Audio Plugin format from Apple) with the same functionality.
The framework not only offers support for multiple format and multiple platform audio plugin creation, but also a GUI framework. Such, that the plugin has the same appearance in all the different formats.
JUCE is a commercial framework with a free educational version[12].

Chapter 3

Theory

3.1 Music Theory

The music theory used has been transferred from the previous robot composer project [1]. The compositions of the robot composer are based on western music theory and therefore its chord progressions and song structure.

3.1.1 Instrument range

The framework has been extended to take the limited range of instruments into account [13, 14]. For each channel one has to select an instrument and a function, such as lead, accompaniment or bass. One has to make sure that the range of the selected instrument is compatible with the function it has to full fill. Therefore, lists of instrument for each functions have been introduced.

3.2 Rhythm Tree

A rhythm algorithm creates a rhythmic figure with a length, that is specified by the input. The root represents the whole duration, and the duration of each child is calculated as the parent duration divided by the number of siblings. The tree is constructed, by randomly deciding for each node, whether to add a random amount of children or to leave it as a leaf node. During the construction one has to make sure that there are no nodes with a duration below a minimal duration. The rhythm can be generated by parsing the leaf nodes and adding their duration. To make it more interesting, one can merge a few leaf nodes together and randomly select some leaf nodes to be rests.
In 1 there is an example algorithm to produce rhythm with the rhythm tree method. This algorithm has been inspired by [15].

Data: groundrhythm, splitProbability, mergeProbability, restProbability, dividers, smallestDuration
Result: Rhythm
for *beat in groundrhythm* **:**
    choose divider from dividers
    if *beat / divider > smallestDuration and splitprobability* **:**
        splitRhythm add recursiveSplit(beat/divider, splitprobability, smallestDuration)
for *beat in splitRhythm* **:**
    while *mergeProbability* **:**
        beat += next beat
    if *restProbability* **:**
        Rhythm addRest(beat)
    else:
        Rhythm addBeat(beat)
return Rhythm

Algorithm 1: Rhythm generation with the rhythm tree method.

Chapter 4

Implementation

The application is divided into 4 parts. There is the Plugin Processor, which is responsible to playback the generated Midi data and runs on its own thread, because the processing is required to be in real time.
Then, there's the Structure, which holds the meta information, such as the time signature or bpm, but also the actual music. It provides functionality to modify the music, generate a midi file, serialize and restore it's state and to display the piece of music on the GUI.
Next there is the Robot Composer part, which does the actual composing. It offers functionality to create a piece and fill in all the meta information but also to alter an existing piece.
Finally, the plugin editor acts as the GUI. It should provide access to all the functionality but also allow an intuitive work flow for the user.

4.1 Piece

The Piece class is a container to hold the song structure the meta information of each block and the piece itself and the actual music.

4.1.1 Instruments

The Piece class maintains a list of instruments and saves the channel, program change and the functionality of that instrument.
There can be several types of instruments: lead, accompaniment, bass, ambient or drums. This information is used by the Robot Composer to compose music that is tailored to the functionality of the instrument.

4.1.2 Structure

A piece consists of different parts. Each part has its function, such as intro, verse, chorus, down or outro. This has been transferred from the previous Robot

Composer project [1]. Now each block additionally saves the chord progression and the ground beat of the music. This additional information is used by the Robot Composer to ensure or make it more probable, that the scores for the different instruments fit together. Such that there is a bit of coordination between the instruments.

4.1.3 Meta information

The meta information is a struct to save musical parameters such as volume, bpm, scale or time signature.
Each Block has its own meta information, that means that each block is played with its own tempo, volume and scale. This adds some variety to the piece and makes the different block more distinguishable.

4.1.4 Serialization

The state of a music piece can be serialized, such that one can save the progress on a composition and continue the work another time. [16]

4.2 Composers

The purpose of a composer is to provide functionality to create a new music piece and to recompose specified parts of an existing piece. The Robot Composer base class defines the composing procedure and asks the child composer how they would execute the tasks in this procedure, by calling virtual functions. An example for such a task could be selecting the instruments or to generate the block structure. A new composer can override those functions and when this composer is selected on the GUI the code that overrides the base functionality will be called during the composing.

4.2.1 Conventions

To create a new composer, one has to follow a few conventions and guidelines:

- The duration of a note or a chord is always measured in a fraction of quarter beats.
- The pitch of a note is relative to the ground note of the block it is played in.
- The volume of a note is measured in percentage of the block volume.

- The Robot composers use seeded randomness, such that a composition can be recreated with the same seed. As source for the randomness a Mersenne Twister Engine from the c++ standard library [17] is used. The UniformRandomness header file provides a selection of frequently used random functions.
- Overrife the virtual functions from the RobotComposer base class

4.2.2 Robot Composer

The "RobotComposer" class is the base class for all the robot composers and implements the definitions for the public functions. Those are the functions, that are called by the GUI.
It also provides a basic implementation for the virtual functions. Those can be grouped into three categories:

Virtual Functions

The functions to create a new piece. They add/generate the meta information for the Piece and select the instruments by using the instrument ranges header file.
The functions to create the block structure and fill in the block meta data have been inspired by [1]. They have been adapted to slightly change the meta information for block.
The last group are the functions that actually compose the music. Each instrument type has its own composing function and the composer has access to the chord progression and the ground rhythm of the block it is currently composing.

4.2.3 Jazz Composer

The jazz composer has been transferred from the former Robot Composer project [3]. It isn't as general as the base composer, as it doesn't support all the different time signatures. It overrides the instrument selection, the block structure generation and the composing of the drum and accompaniment voices.

4.2.4 Music Theory Header Files

In the former Robot Composer Framework, there was a look up file with both midi specific definitions and concepts from music theory, such as chords and scales. This file has been split up into a general Midi header file in the Structure section and music theory file for the robot composers.

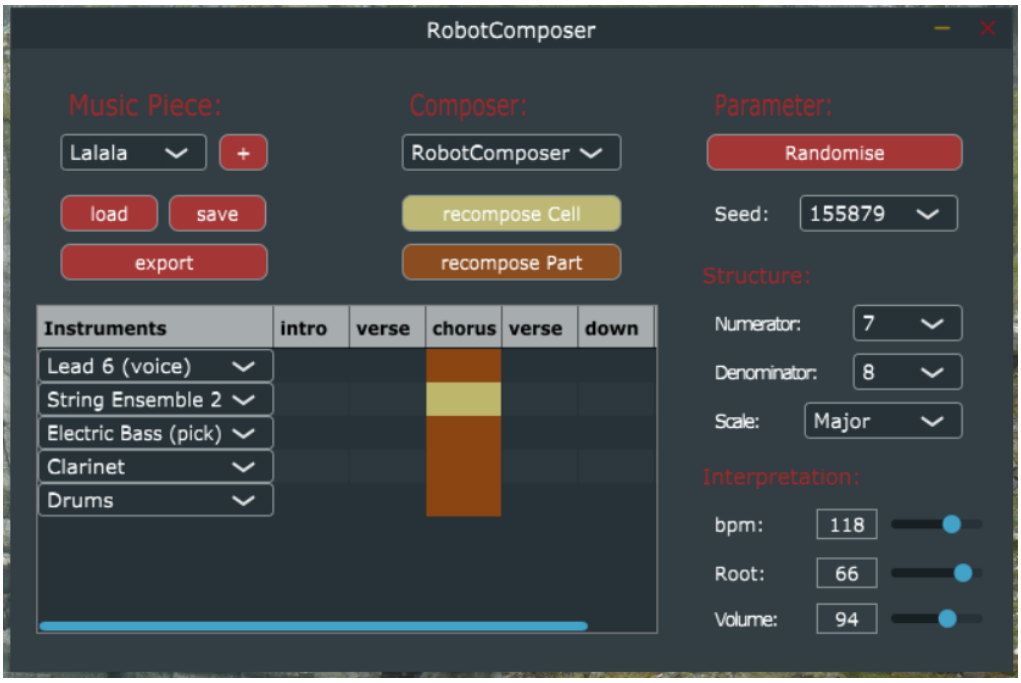

Figure 4.1: Screen shot of the VST Plugin.

There is also a new rhythm header file, which provides some functions to handle the base Rhythm saved by each block and the rhythm Tree function which implements the rhythm tree algorithm 1.

4.3 Editor/GUI

The GUI 4.1 has been developed using the Projucer GUI editor from the JUCE framework and offers the following functionality and interaction possibilities to the user:

- Select one of the Robot Composers from the composers drop down box.
- create a new piece of music, which is composed by the selected composer.
- Save the piece, which serializes the selected piece and writes the data to a binary file.
- Load a previously saved piece.
- In the Piece selection box, the piece can be renamed. If there are currently multiple piece loaded, a different piece can be selected.

- The midi messages for the selected piece are created and loaded into the processor.
- The selected piece is displayed on the GUI in a table representation, where each cell represents the score of the corresponding instrument played in the corresponding block.
- During playback, the block that is currently played is always selected.
- Each cell can be selected and one can tell the selected composer to either recompose only the selected score or to recompose the whole block with new parameter values.
- The Parameters of the currently selected block are displayed in the parameter colon on the right side of the GUI.
- The interpretation parameters can be changed by the user. This triggers a regeneration of the music and the MIDI messages that are loaded in the processor are updated. For example, one can change the volume or bpm of the block, even during playback.
- With the instruments drop down boxes, the user can change the instruments, that are used to play the scores of the corresponding channel.

4.4 Plugin Processor

Most of the plugin processor class has been generated by JUCE and handles the communication with the host. Amongst others, the generated functionality is used to tell the plugin host whether the plugin processes audio, midi or both.
The actual processing is done in the processBlock function. This function gets repeatedly called by the Host with the data to process. Since the RobotComposer doesn't take any inputs but plays the created music it ignores the input data. The midi playback code has been copied from a simple MIDI player [18]. This MIDI player takes the song position info, which the plugin receives from the host, to determine which midi messages have to be sent to the plugin output.
The MIDI player has been extended to figure out, which part/block of the loaded music is playing and to send a notification to the GUI whenever a new block started playing.

Chapter 5

Result

Even though, VST-plugins aren't designed to generate music but rather process the one that is delivered to it by the host, the plugin integrates with the DAW and therefore the music producing process. The user can generate music from different composers and the tool offers the possibility to recreate parts and change the interpretation of the created music.

Chapter 6

Conclusion

6.1 Personal Insights

When I was implementing the robot composer class, I realized, that it is quite difficult to decide what freedoms, in form of random decisions, to give to the composer and where to force it into fixed structures. The perception of music by humans heavily relies on pattern recognition. Therefore, it is important that the Robot Composer makes use of known patterns in rhythm but also in chord progressions or melodies and it should reuse the randomly generated patterns as well. Therefore, the task is to find the balance between random decisions and restricting the robot composer, such that the tool frequently generates decent music pieces.
I had performance issues when composing a new piece or when generating the midi data. This caused a freeze of the GUI, which isn't the best user experience. I had to rewrite the function declarations, such that the arguments are either pointers or they are called by reference. In future it would definitely be worthwhile to start using the calls by reference and pointers from the beginning. Additionally, I had to implement a cache that saves the midi messages of each block and only the messages for blocks that have changed are getting regenerated.

6.2 Future Work

New features can be added to the different parts of the tool.

- One could figure out whether its feasible, that the processor can issue a jump of the song position pointer. That would make it possible, that the plugin could tell the host to jump to the beginning of one of the blocks.
- Create a better composeDrums function for the base RobotComposer. In my opinion, the current one is not really sophisticated and it would be worthwhile to improve it.

- The JazzComposer should be improved, such that it supports all the parameters that can be configured on the GUI.
- The structure can be extended to support additional midi features. It is possible to tell the midi instruments how a tone should be expressed, with messages that change the note down speed, the after touch or the pitch wheel characteristics.

- One could implement the functionality to increase or decrease the tempo and the volume within a block.
- On the GUI, one could add the possibility to add and delete blocks and instruments.

Obviously, new composers can be developed and one could try to get community feedback on the quality of the composers and the usability of the tool.

Bibliography

[1] N. Studach, "Robot Composer Framework,"
https://pub.tik.ee.ethz.ch/students/2017-HS/SA-2017-95.pdf
, 2018.

[2] C. Zu, "Hip hop robot,"
https://pub.tik.ee.ethz.ch/students/2018-FS/SA-2018-23.pdf
, 2018.

[3] R. Schmid, "Robot composer,"
https://pub.tik.ee.ethz.ch/students/2016-HS/SA-2016-63.pdf
, 2017.

[4] Website, "Midi association,"
https://www.midi.org
.

[5] Website, "General midi,"
https://www.midi.org/specifications-old/item/gm-level-1-sound-set
.

[6] M. Vaidyanathan, "Midi to score sheet,"
http://midisheetmusic.com
.

[7] Website, "Midi file,"
https://segaretro.org/MIDI
.

[8] A. Wegerle, "Was ist midi?"
https://blog.landr.com/de/ist-midi-eine-einfuehrung-das-einflussreichste-tool-das-die-musik-je-g
.

[9] Wikipedia, "Daw,"
https://en.wikipedia.org/wiki/Digital_audio_workstation
.

[10] Wikipedia, "Vst plugins,"
https://en.wikipedia.org/wiki/Virtual_Studio_Technology
.

[11] Steinberg, "Vst sdk,"
https://www.steinberg.net/de/company/developer.html
.

[12] Website, "Juce - multiplatform audio applications development framework,"
http://juce.com
.

[13] Website, "Range of instruments,"
http://www.orchestralibrary.com/reftables/rang.html
.

[14] J. Champion, "Midi note ranges of instruments,"
https://soundprogramming.net/file-formats/midi-note-ranges-of-orchestral-instruments/
.

[15] N. McBride, "Constructing rhythm trees in openmusic,"
http://www.nmcbride.com/constructing-rhythm-trees-in-openmusic/
.

[16] W. S. Grant and R. Voorhies, "cereal - a c++11 library for serialization,"
http://uscilab.github.io/cereal/
, 2017.

[17] c++ std library, "Mersenne twister engine,"
https://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine
.

[18] IvanC, "Playing a midi file,"
https://forum.juce.com/t/playing-a-midi-file-revisited/26646
.

Appendix A

Instrument Ranges

List of Orchestral instruments and their MIDI note range [14].

- **Strings**
- Violin 55-103
- Viola 48-91
- Cello 36-76
- Double Bass 28-67
- Bass Guitar 28-67
- Acoustic Guitar 40-88
- **Brass**
- Tuba 28-58
- Bass Trombone 34-67
- French Horn 34-77
- Trombone 40-72
- Trumpet 55-82
- **Woodwinds**
- Piccolo 74-102
- Flute 60-96
- Oboe 58-91
- Alto Flute 55-91
- Cor Anglais (English Horn) 52-81
- Clarinet 50-94
- Bass Clarinet 38-77
- Bassoon 34-75
- Contrabassoon 22-53
- Soprano Recorder 72-98
- Alto Recorder 65-91
- Tenor Recorder 60-86
- Bass Recorder 53-79
- Baritone Sax 36-69
- Tenor Sax 44-76
- Alto Sax 49-81
- Soprano Sax 56-88
- **Tuned Percussion**
- Glockenspiel 79-108
- Xylophone 65-108
- Vibraphone 53-89
- Marimba 45-96
- Bass Marimba 33-81
- Celeste 60-108
- Tubular Bells 60-77
- Timpani 40-55
- Harpsichord 29-89
- Harp 24-103

Appendix B

How To

README.md

Repository for robot composer framework. The goal of robot composer framework is to provide a framework to algoritmically generate music according to the ruleset given by music theory. This project served as Semester thesis at ETHZ. Written in C++ 17.

User:
1.) Install and open a Digital Audio Workstation that supports MIDI and VST plugins, such as Cubase, Ableton or Acid.
2.) copy the RobotComposer.dll to the vst plugin directory of the DAW and update the plugin list
3.) Create a project with a midi track
4.) Add the RobotComposer as a virtual instrument to the project
5.) set the RobotComposer as the Input of the MIDI track
6.) set the output to a instrument that supports the General Midi standard. For example the Microsoft GS Wavetable Synth preinstalled in Windows

If there is no DAW available on the computer, one can open the standalone plugin RobotComposer.exe This application comes without replay functionality. Therefore, one has to export the created music and listen to the midi file. In windows the windows media player can be used to play midi files.

Developer:
1.) Download the vst sdk (https://www.steinberg.net/de/company/developer.html)
2.) Download the Juce framework (https://shop.juce.com/get-juce)
3.) open the projucer application and set the global paths under file -> global paths... .
4.)open the RobotComposer/RobotComposer.jucer file
5.) In the settings of the project one can select the plugin type and other pluginparameters.
6.) The GUI can be modified, by opening the PluginEditor.cpp in the File Explorer of the Projucer.
7.) Make sure, that the Modules have the correct path to the JUCE/modules directory.
8.) In the exporters tab one can create exporters for the desired development environment and architecture.
9.) Finally one can open the code in the development environment and start coding.

Figure B.1: README from the git lab repository of this thesis.