System Design Document for MelodyGenerator (Group 19)

Version: 1.0
Date: 2014-04-07
Authors: David Oskarsson, Kristofer Yffén, Emma Nyborg

This version overrides all previous versions.

Contents

1 Introduction

1.1 Design goals

Usability
One goal is that the program should be an easy-to-use tool for creating own music or get inspiration for making new tunes with or without any prior knowledge of music theory.

Therefore, the program needs to be simple but functional. There should be some settings with default values, e.g. key, length of tones, signature. A single prominent “done”-button should make it easy for non-advanced users. For experienced users that wants to specify their own requirements there should be a window for settings where they can add regular expressions.

Extensibility
In the future, one can add more pre-defined filters and filter groups to make it easier to set common rules in music.
Another feature that could be added is support for loading previously generated MIDI-files and edit these with new regular expressions and filters.
One can also extend the player to be able to play with different instruments and decide which instrument that should be used within the MIDI-file that is saved.

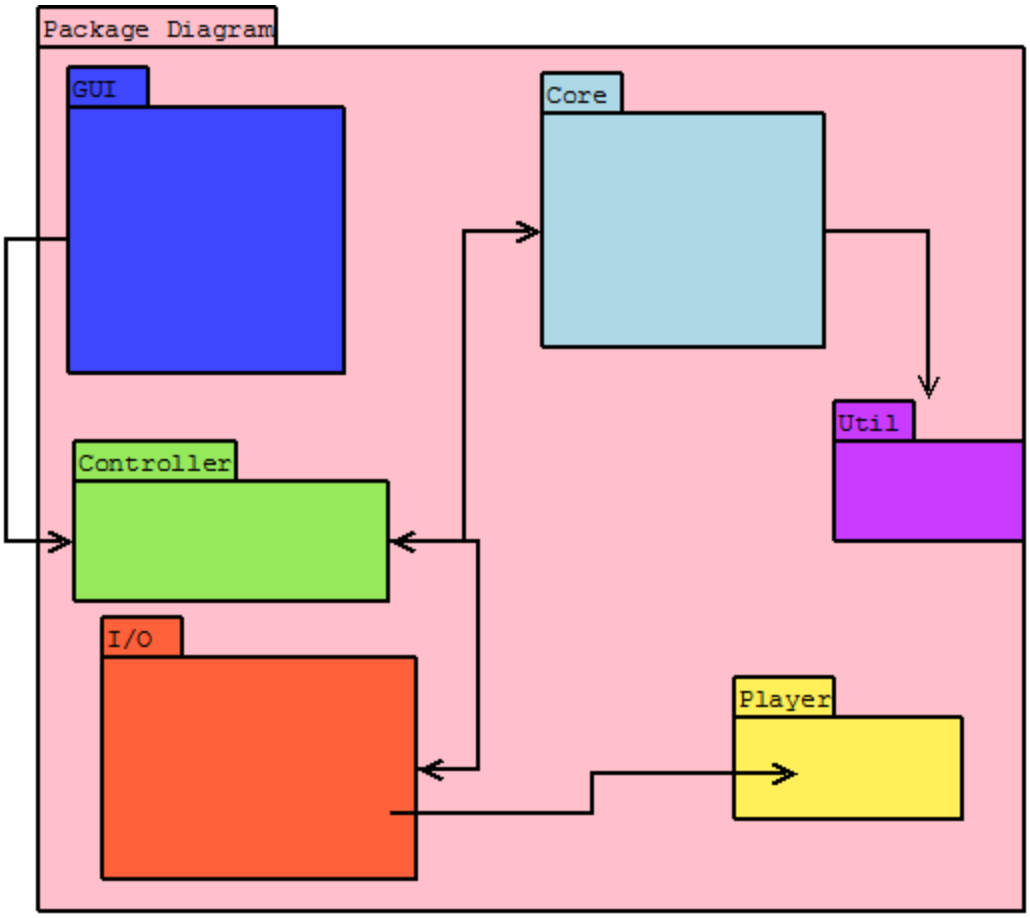
Testability
Some basic testing needs to be done to make sure the program is doing what is expected of the program.
The input needs to be tested so that it contains the correct data, otherwise there will not be any result. The data also needs to be tested with the regular expressions that can be added. A final check is to test the output so that it matches the preferences that you have added (i.e key, note length etc…).

1.2 Definitions, acronyms and abbreviations

ABC = A music coding language that can be parsed to MIDI.
MIDI = Musical Instrument Digital Interface
MVC = Model-View-Controller
GUI = Graphical User Interface

2 System design

2.1 Overview

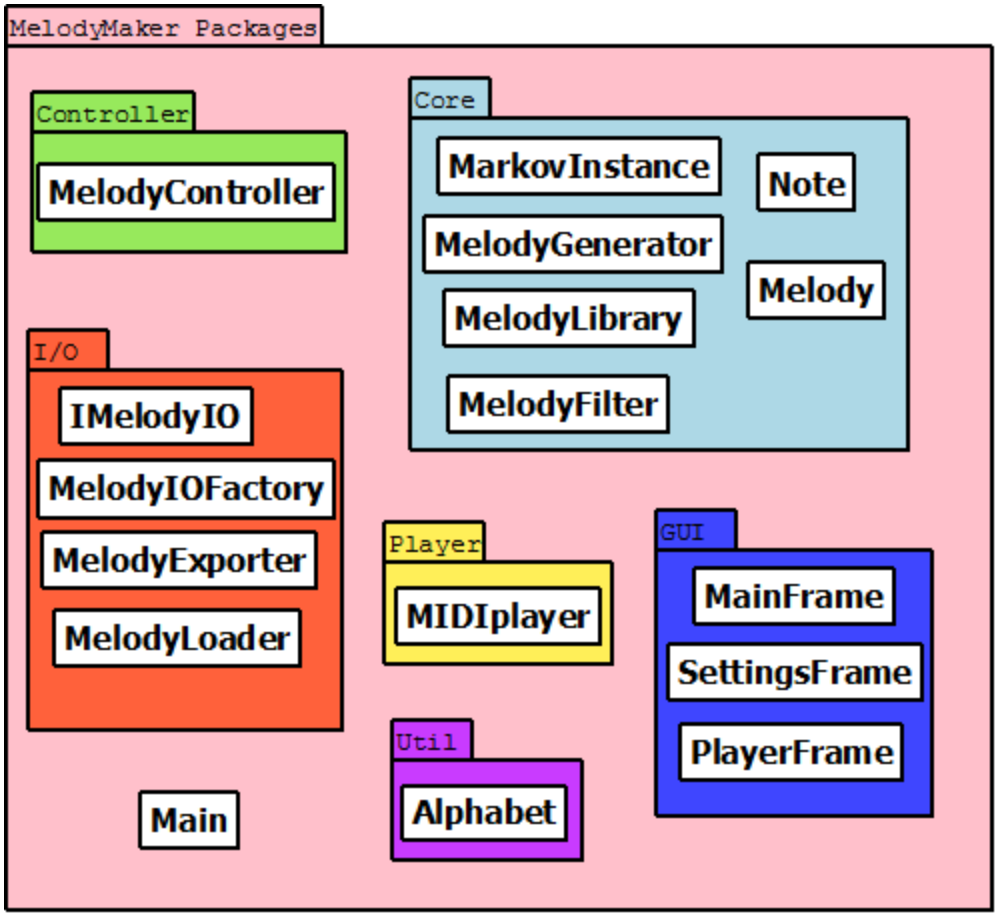


- The controller package is handling the connection between the GUI and the view in the MVC structure
- The I/O package handles all the input and output. The data will be loaded with the Loader class and the Exporter class is used to export data.
- The core package consist of the model
- The player and util packages are supporting packages
- The GUI package is the one creating the graphical interface

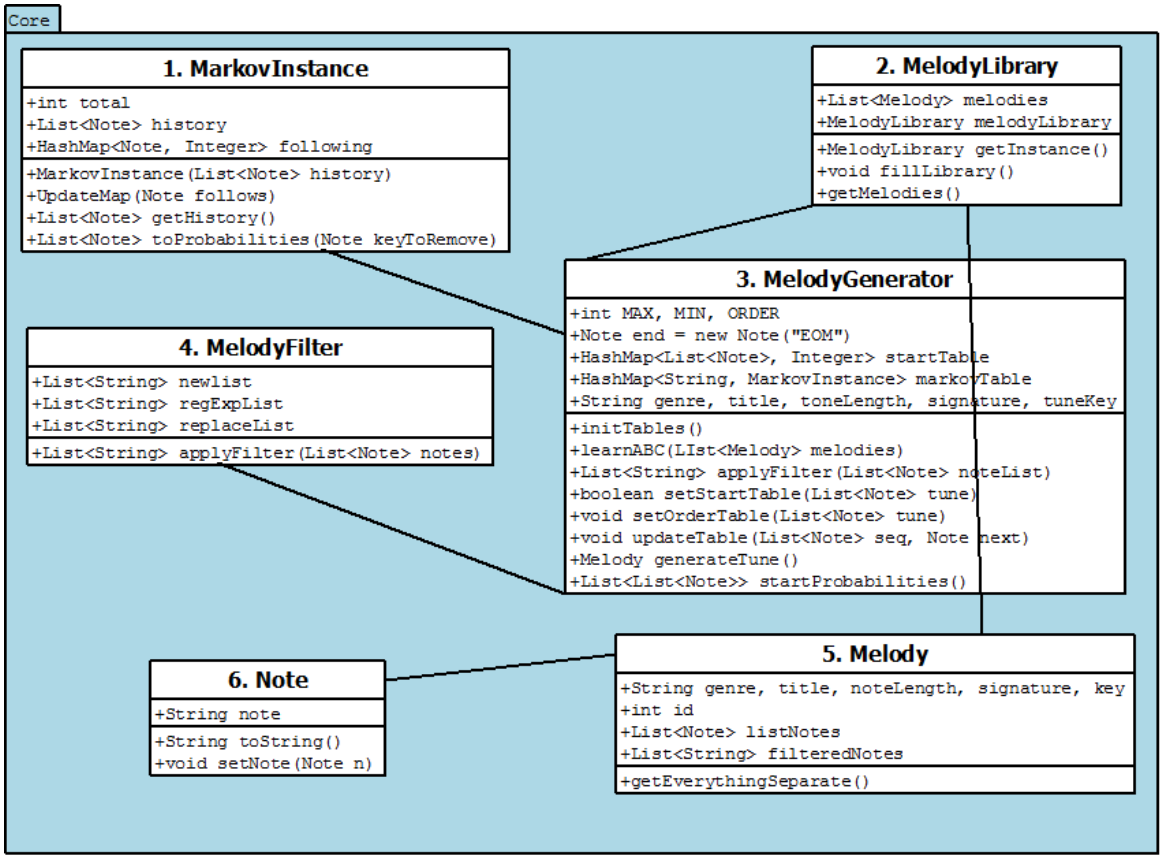
Melody Maker is using the MVC design pattern and the model is represented by the core package, the other package's responsibilities are quite obvious.

2.2 Software decomposition

2.2.1 General



2.2.2 Decomposition into subsystems



1.
* The probability tables in MelodyGenerator is filled with MarkovInstances. An
* object of this class stores information of for a sequence of notes and the
* probabilities for other notes to follow this sequence.

2.
Holds all Melody-objects

3.
* MelodyGenerator can generates new tunes with Markov chains. The probabilities
* is based on the note succession in the melodies in MelodyLibrary.

4.
* A class that makes you generate your tune after your own specifications.
* You can apply this filter on the generated tune

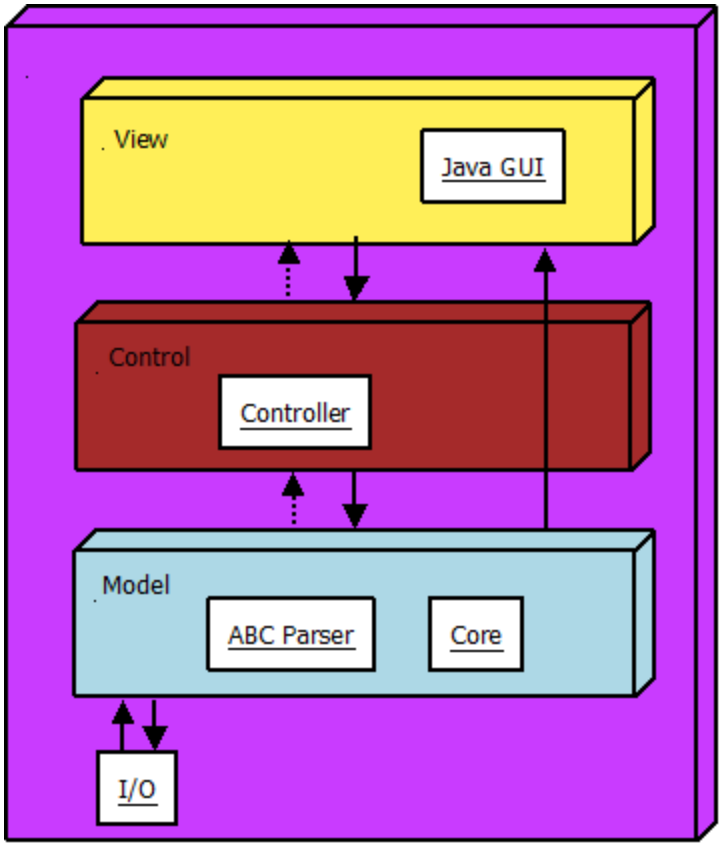
5.
An object of this class stores all information we need for a melody

6.
* A simple wrapper class that holds a note, represented as a String of various
* length

- MarkovInstance, which is a helper class for MelodyGenerator
- MelodyGenerator, which uses a Markov chain to create a melody from the data that is loaded
- The MelodyLibrary stores many Melody objects in a library
- MelodyFilter is a class that applies specific filters on the tune so that you can modify the tune by your own taste.
- Note represents a note, which is specified in our Alphabet class in the util package
- Melody is a class that is representing a melody that is stored in the MelodyLibrary

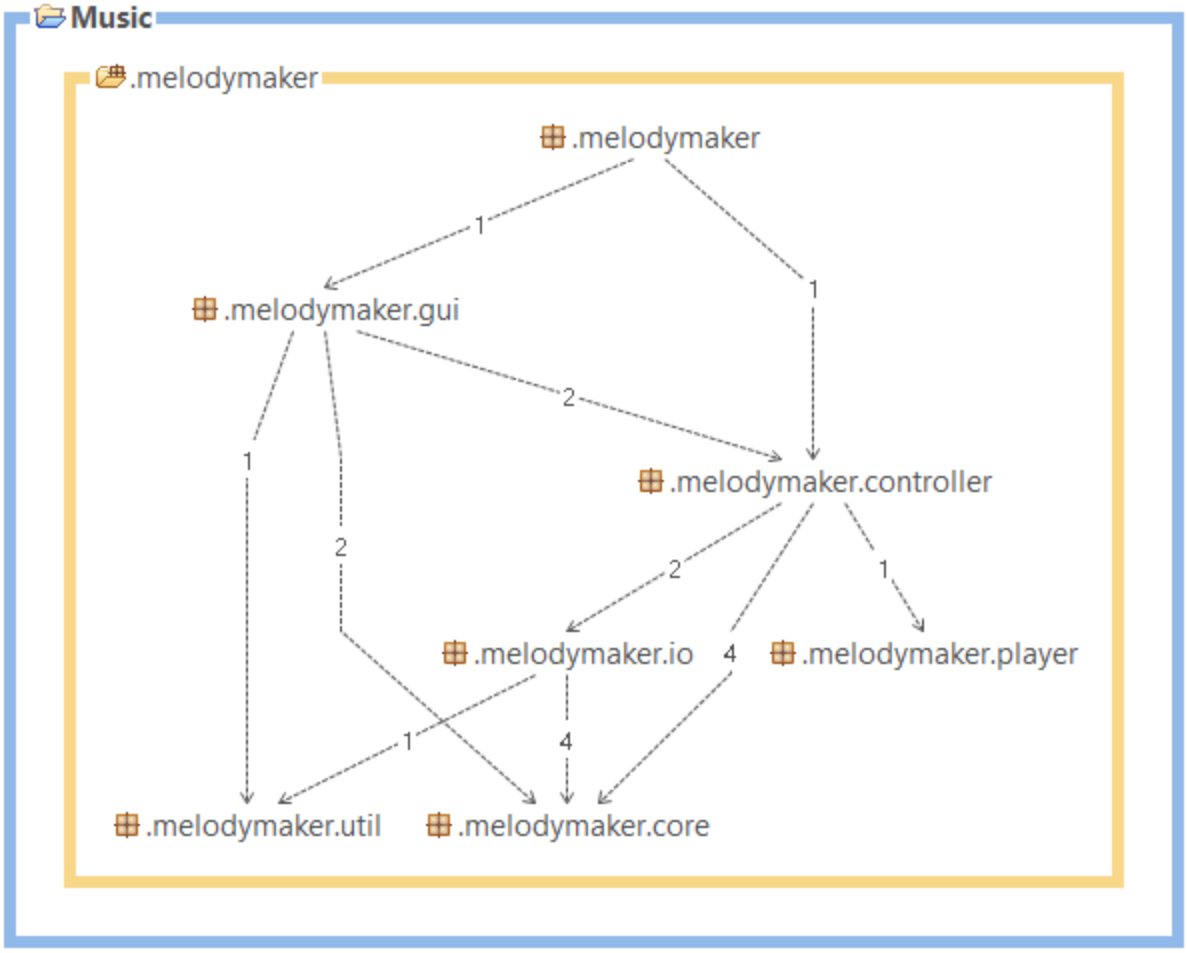
2.2.3 Layering

The layering is as shown in the figure below. Higher layers are at the top of the figure.



2.2.4 Dependency analysis

The Controller package handles the interaction between our model and view, and also between all sub-systems. The figure below shows the dependencies according to STAN.



2.3 Concurrency issues

n/a

2.4 Persistent data management

For the Melody Maker, the storage of data (in ABC format) has been separated from the application logic. It also allows the data to be used in other applications. For the purpose of extensibility, a database system is required.
See appendix A for ABC example.

2.5 Access control and security

n/a

2.6 Boundary conditions

n/a

3 References

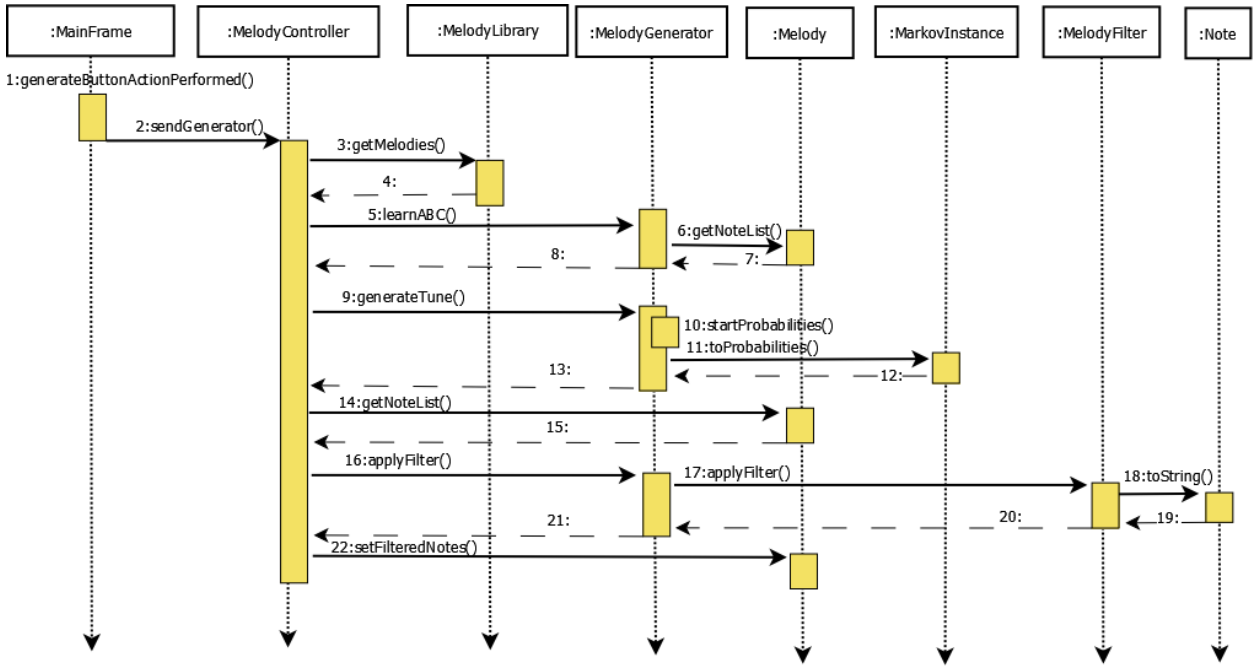
Chris Walshaw, 2014, URL: http://abcnotation.com/software
FolkWiki, 2014, http://www.folkwiki.se/
http://docs.oracle.com/javase/7/docs/api/javax/sound/midi/package-summary.html

APPENDIX A

Example of an ABC code:

```
X: 1
T: Molly Put The Kettle On
R: reel
M: 4/4
L: 1/8
K: Amin
cded cAAc|Bcdc BG G2|ceed cBAG|EDE^G A2 A2|
cded cAAc|Bcdc BG G2|cBcA B2 AG|EDE^G A2 A2||
cde^f g2 fg|aged cA A2|cde^f g2 fg|age^g a2 a2|
cde^f g2 fg|aged cA A2|cded cAAG|EDE^G A2 A2||
```

Sequence diagram for clicking Generate



Sequence diagram for Save Song