Linux Multimedia Studio

Alexandre Hudon 9580433  
Sardar Maninder Singh 6555519  
Simon Symeonidis 5887887  
Pavleen Kaur 9754733   
Amir Hakim 4050711  
  
Due: Feb 25th, 11 marks

# Github

As requested, the assignment code, and documents are all located here:

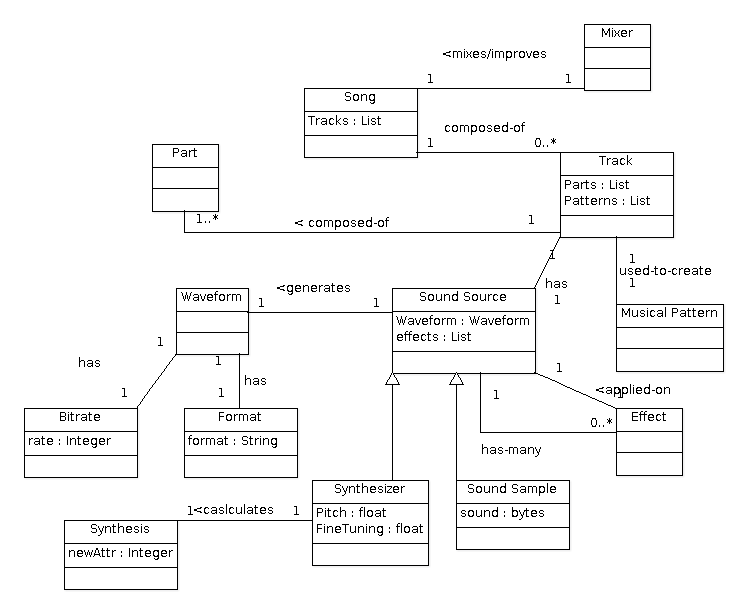
<https://github.com/AHudon/SOEN6471_LMMS>

# Summary of Project

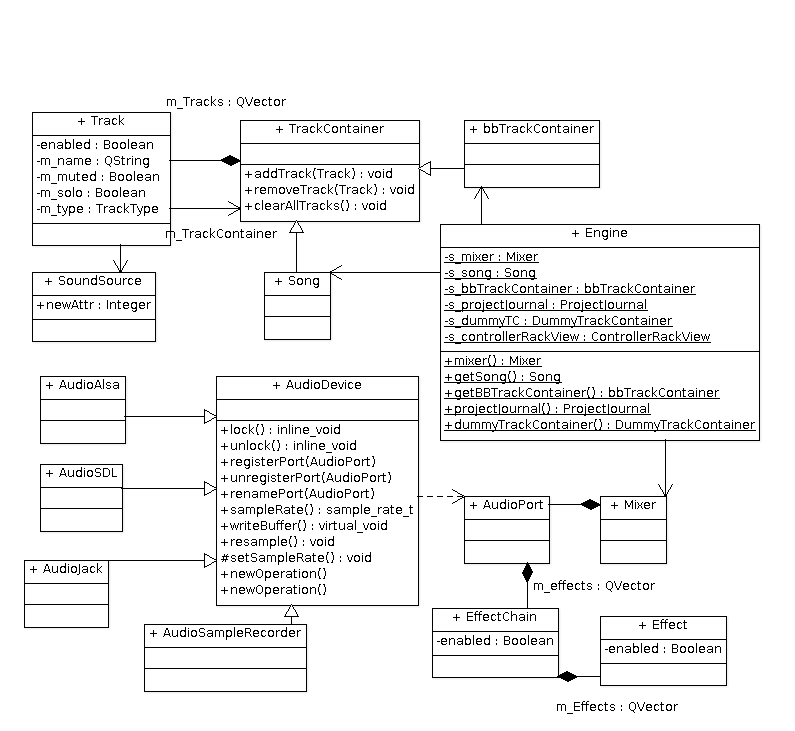
LMMS (Linux Multimedia Studio) is a DAW (Digital Audio Workstation), very similar to commercial ones, such as FL Studio. It provides facilities for you to engineer synthesizers in order to create instruments, and also piano rolls to create possible melodies. There exists a beat-bassline interface as well, to assign repetitive patterns quickly.

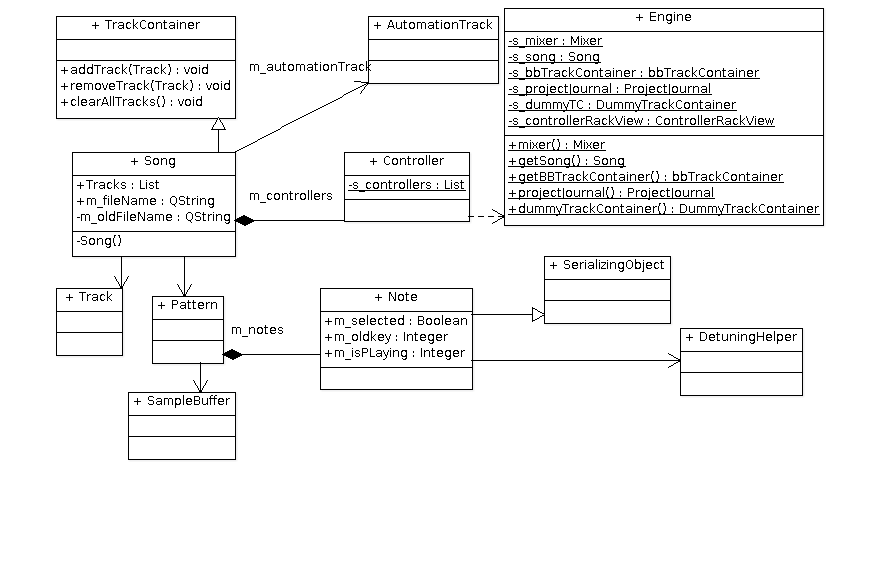
# Class Diagram of Actual System

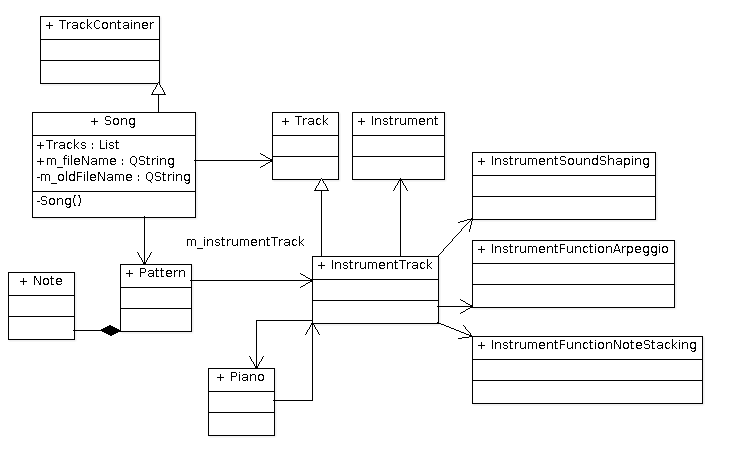
Below is the class diagram with the corrections from the feedback of the previous milestone.



## Class Diagram

Figure 1: General Overview





In figure 1 we see the overal organisation of the software.

## Conceptual Classes to Actual Classes

asd

* + Is there a discrepancy between the concepts and the actual classes?

## Discrepancy Between the Concepts and Classes

* + What does it mean to the architecture of the system?

## Reverse Engineering Tool (Doxygen)

We used Doxygen as a reverse engineering tool. If a system using Doxygen has the extra package 'graphviz', then apart from extracting the API of the application, we are able to extract many other diagrams of interest, namely class diagrams, and collaboration diagrams.

However in order to present the work, ArgoUML was used in order to make diagrams that are more readable. As previously stated, some classes in those diagrams were left blank or partly filled because in this project, long classes are quite common. This already raises an obvious need for refactoring.

* In maximum of one page, for two classes and the relationship between them:

For the two classes requested, we chose *TrackContainer*, and *Track*. We show how the relation exists, in this case the aggregation. We also show the similar methods and attributes that are demonstrated in the class diagram by selecting to paste *specific* code. The definition, and implementation of these classes is quite long, so most code is omitted.

|  |
| --- |
| Track Definition (header) [/include/track.h] |
|  |
| Track Implementation (cpp) [/src/core/track.h] |
|  |

|  |
| --- |
| TrackContainer (header) [/include/TrackContainer.h] |
|  |
| TrackContainer Implementation (cpp) [/src/core/track.h] |
|  |

# Code Smells and System Level Refactorings

Be idealistic! Restructure the system to fix some of the code smells you identified above.

* Describe the code smell(s) and how you will combine together a series of refactorings to fix the smells.
* Describe how the refactorings are interrelated and how they correct the problems you identified in the above. For example, “First, I moved the methods X and fields Y and Z, to increase cohesion and to reduce feature envy. Second I renamed the class to reflect its new limited responsibilities. Third I removed the coupling to …”
* 1 to 2 pages of text + 1 to 2 UML diagrams (diagrams should be at most 2 additional pages).

# Specific Refactorings that you will implement in M4

Be realistic, you have to implement at least two of these! Suggest two to four refactorings that you will implement in Milestone 4. In maximum of one page, for one of the refactorings you suggest: Copy-and-paste the class, method, and attributes **declarations** (and anything else that is necessary) directly from the source code. Do not include code that is unnecessary (be selective, you will lose marks for large dumps of source code.)

Notes:

1. Algorithmic changes are not acceptable refactorings. You must deal with the logic and relationships between the classes -- ideally real world or domain entities.
2. Stay away from the GUI classes as they often contain a lot of autogenerated non-domain code.
3. Moving a code around without modifying it is unacceptable.
4. Isolated changes that do not affect other parts of the system is unacceptable.
5. Example of possible refactorings: Refactoring 1, fix some complex if statements. Refactoring 2, introduce a strategy pattern and use delegation to keep the system running. Gradually remove the delegation on some of the cases.