System design document for the HueStew project

Version: 1.0
Date: 2016-05-28

Author: Adam Andreasson, Daniel Illipe, Patrik Olson, Marcus Randevik

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The application is to be designed in a way to make it possible to easily add new types of lamps to use. The design must also be testable i.e it should be possible to isolate parts of the program (modules, classes) for test. Regarding the usability of the application, see RAD.

1.2 Definitions, acronyms and abbreviations

- **GUI,** graphical user interface.
- **Java**, platform independent programming language.
- **JRE**, the Java Runtime Environment. Additional software needed o run an Java application.
- **Light state,** digital information regarding the state of a lamp, includes color and brightness.
- **Key frame,** determines when in time a new state shall be sent to a lamp, includes a lightstate.
- **Light track,** a set of keyframes ordered by their timestamp.
- **Show,** a collection of one or more light tracks as well as an audio file (music).
- **Sequence**, a shorter list of key frames that can be placed on a light track.
- **Drum**, has a sequence, a light track and a key on the keyboard. Allows the user to input a sequence of key frames with a single tap.

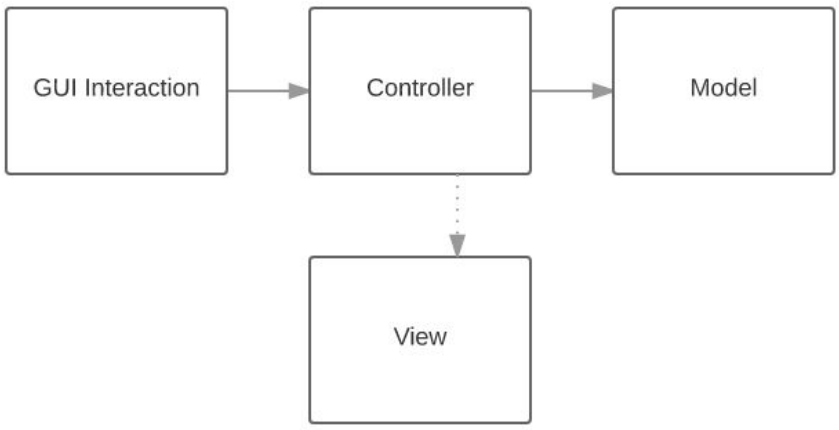
2 System design

2.1 Overview

The application will use the standard MVC pattern. Additional services such as file input/output and a plugin service for the different kind of light sources is also included but placed outside of the MVC structure in order to support extensibility.

2.1.1 Event paths

User input will be handled by standard Java event handling mechanisms (JavaFx). The call chain consists of the following steps: GUI → controller responsible for that part of the GUI -> model class. Depending on the type of user interaction, this might also result in a refresh of the view.



2.1.2 Available lamps

Since HueStew does not require the user to have physical lamps connected to the application at all times, currently available lamps are stored in the LightBank class. This includes physical lights, as well as virtual lights used in the virtual room. Upon startup as well as when the user commands the application to, the light bank will update its list of lamps.

2.1.3 Light state

Since there is no industry standard as for how to send information regarding which color and brightness a lamp should shine, light state stores the color as a custom color and the brightness as a value between 0-255. This enables us to only store information in one way and let plugins translate it as necessary when issuing a command to a physical light.

2.1.4 Show

The core of HueStew is the creation of shows. These hold all the data necessary to create a light show. A show consists of an audio track as well as one or several light tracks which in term contain zero or more key frames. When playing a show, these light tracks will be linked to specific lamps which will shine in a specific color according to the current timestamp.

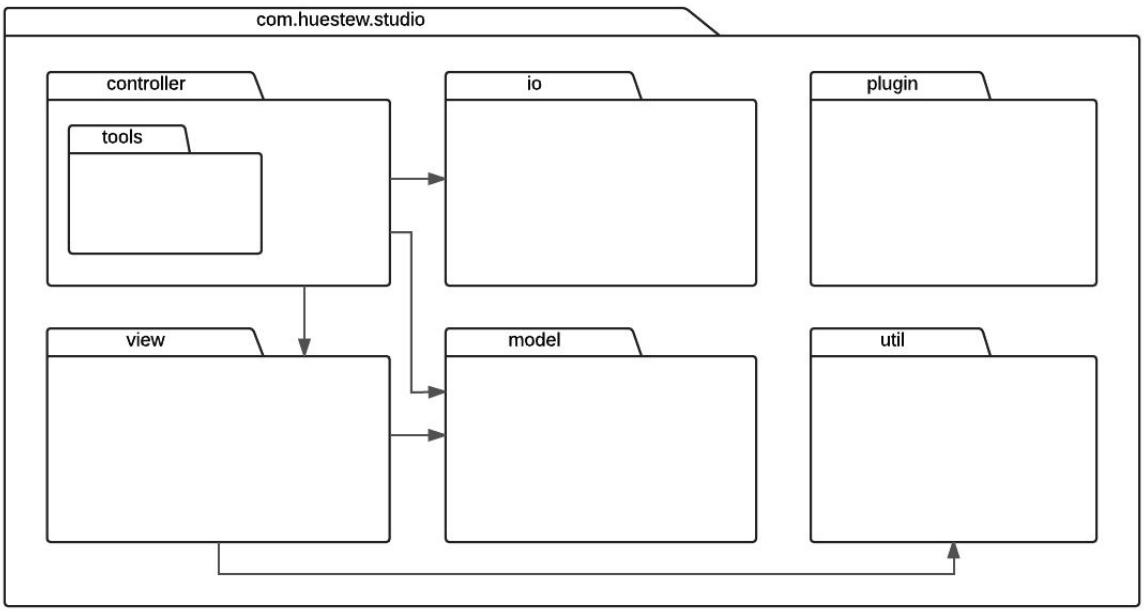
2.1.5 Lookups

As of now, references to the model are explicitly passed on to both controllers and the view. This should, in later development, be reworked to simplify loading/saving shows and handling undo/redo. That is, optimally the current Show would only be stored in one object and could easily be replaced. This would require heavy refactoring of the application, and was not done due to time constraints.

2.2 Software decomposition

2.2.1 General

The application consists of the following top level packages (arrows are for dependencies).



- controller, the top level package for controllers.
    - tools, a package containing tools that interact with the show.
- io, a service package for handling input and output.
- plugin, a package for handling plugins.
- view, the top level package for the views.
- Model, the top level package for the model.
- util, general utilities, possibly reusable by several classes.

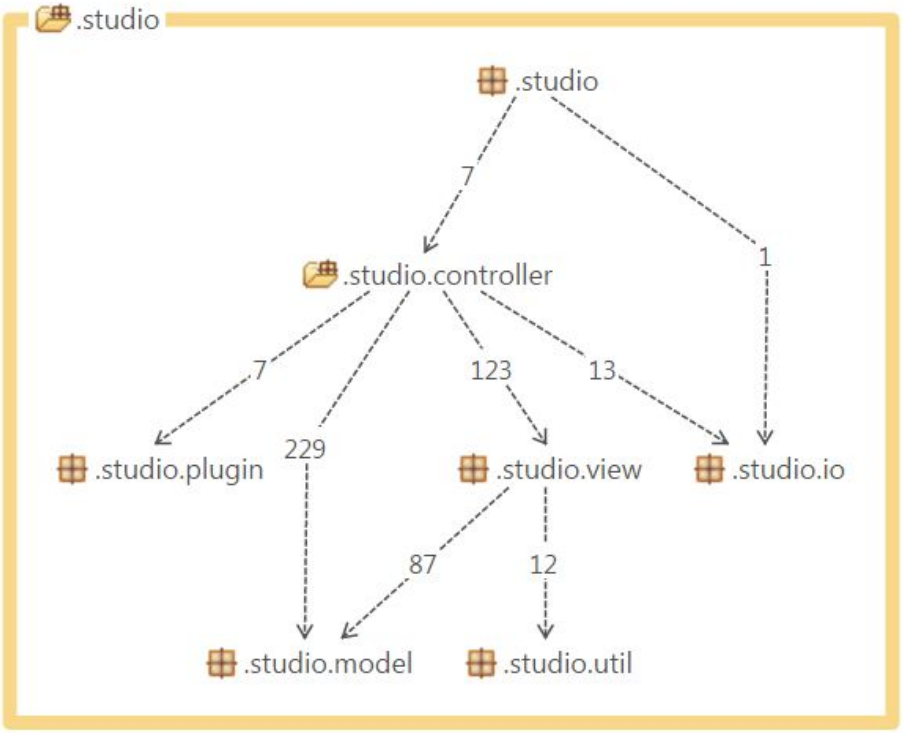
2.2.2 Decomposition into subsystems

The ShowController accesses a Util method to convert mp3 files to wav. This is done through the external Java library Xuggler.

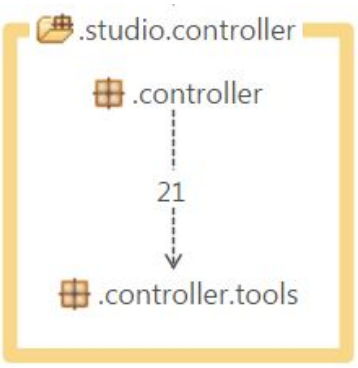
2.2.3 Layering

The application follows the general layer structure of the MVC pattern. The controller is the top layer, which controls both the view and the model. There are additional layers, that can be seen in Structure Analysis diagrams below.

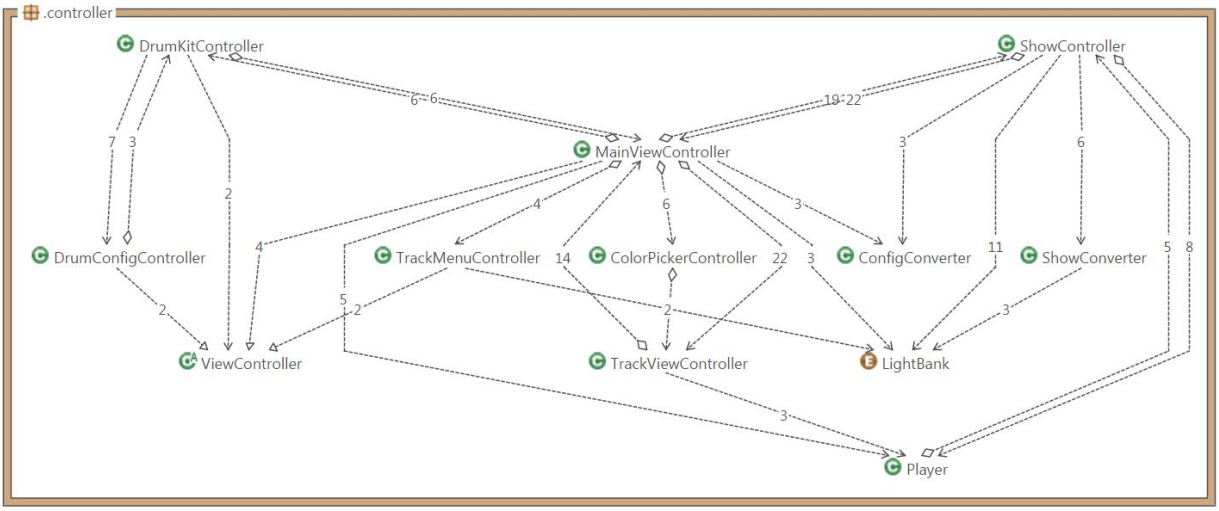
2.2.4 Dependency analysis



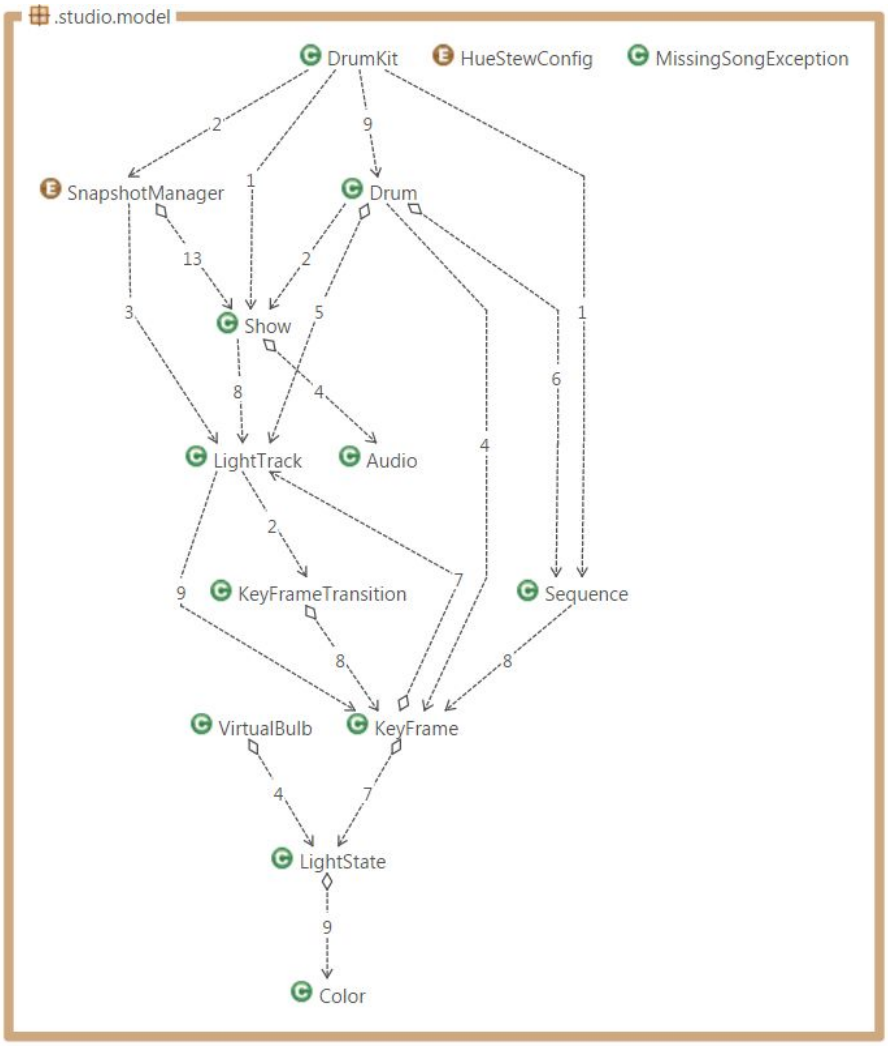
STAN overview of all packages in the program.



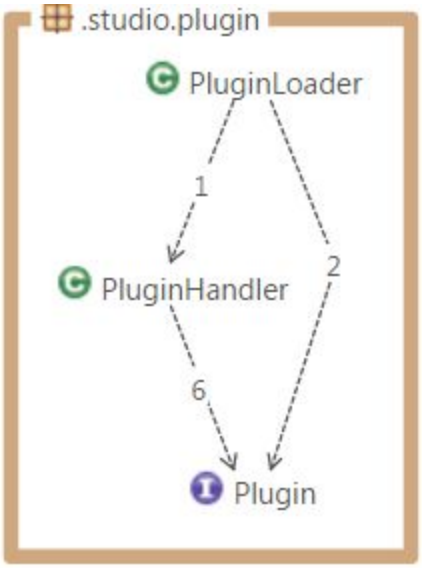
STAN view of controller package.



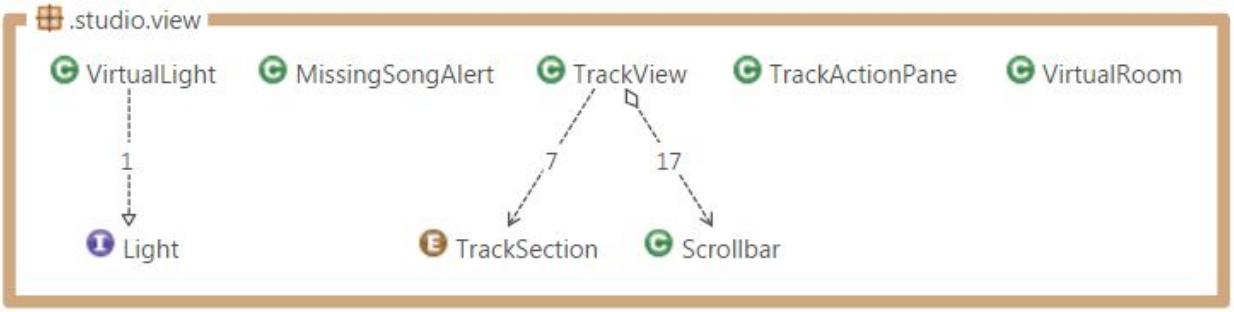
Internal structure of controller package.



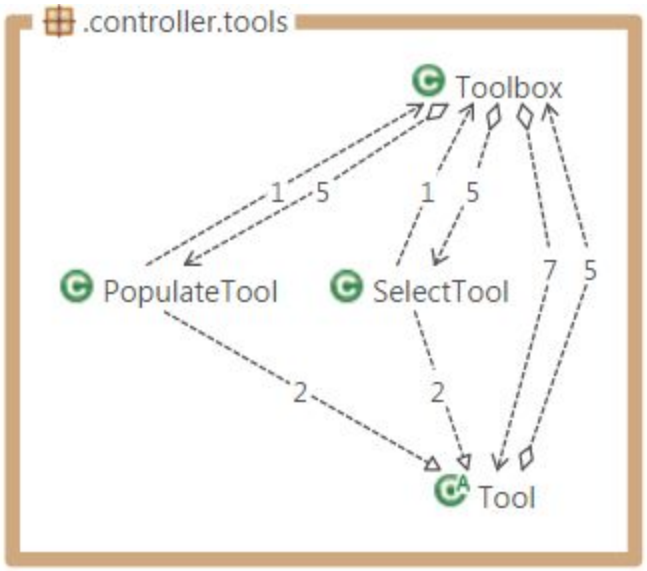
Internal structure of model package.



Internal structure of plugin package.



Internal structure of view package.



Internal structure of tool package.

2.3 Concurrency issues

In addition to the main application thread, there are separate threads that cause the player to tick and the track view to redraw itself. The thread that controls the track view accesses an atomic boolean, which is guaranteed to be thread safe. Any other method calls triggered by these threads are pushed as Runnables to the main application thread. As a result, there are no concurrency issues in the application.

2.4 Persistent data management

Persistent data is stored in flat files, formatted as either properties- or JSON-files. Most files are stored in the user's home directory. A properties file is created that remembers opened files, volume, window size. This file is updated when the program closes. Shows are stored in JSON files, either as an autosave file in the user's home directory, or as specific shows in user specified locations.

Preferred save location is not hard coded, but rather saved in the autosave properties file. This means the user could change their preferred save directory in future versions, should their home directory have limited space.

2.5 Access control and security

NA

2.6 Boundary conditions

NA. Application launched and exited as normal desktop application.

3 References

NA