Boolean Circuits
SDD

Table of contents

1. Introduction

1.1 Design goals

The application should be testable. The model part, i.e the data-part, containing circuit and components should be highly independent with regards to all other components. In that way the user will be able to switch the controller and view and utilize any graphical interface, or none at all. The design should allow future extension such as implementation of not yet implemented use cases.

1.2 Definitions, acronyms and abbreviations

The following expressions and abbreviations will be used in this report:

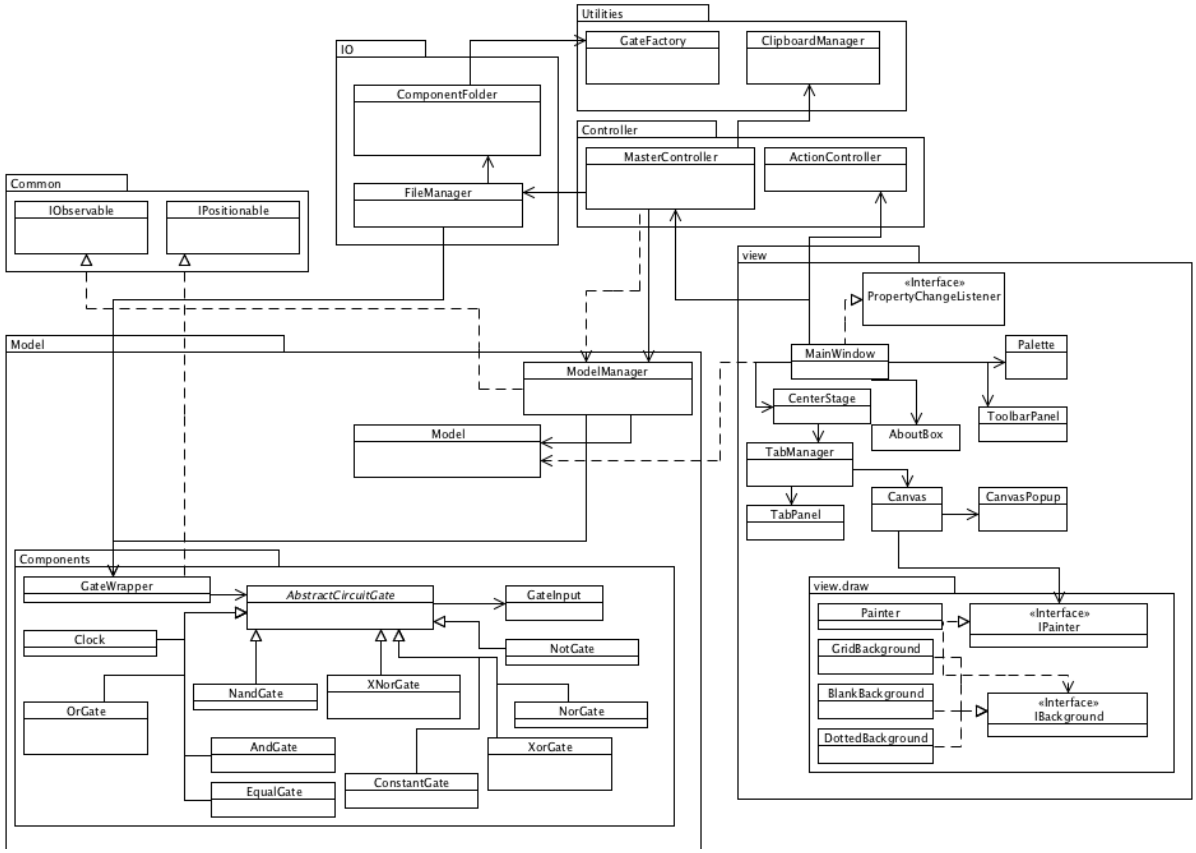
- *Logic gates* will be referred to as *gates*.
- *Workspace* is defined as the area in which the user places and connects components.
- *GUI* - Graphical user interface
- *JRE* - Java Run time Environment. An additional software needed to run a Java application.
- *ALU* - Arithmetic Logical Unit
- *MVC* - an architectural structuring of the code. Dividing the code in to three main parts, model, view and controller.

1.3 References

Logic gate: http://en.wikipedia.org/wiki/Logic_gate
ALU: http://sv.wikipedia.org/wiki/ALU

2 Proposed system architecture

In this section we propose a high level architecture.



2.1 Overview

The application uses an MVC-model where the model is divided into a Model class, representing a circuit, and a Model Manager, which delegates work to the currently active Model. The controller is divided into an ActionController class, providing actions for view components and connecting them to MasterController, a class bridging the view and the model. Our model will be fat and tested with STAN.

2.1.1 GUI

Our GUI is designed to be as easy to use as possible while retaining a sense of seriousness. It consists of the main window, a palette containing different components, a toolbar, a menu and the center stage where circuits are edited and represented. We have decided to keep a class for each different component, thus:

MainWindow - Application window
Palette - Component chooser with components represented in a JTree.
CenterStage - Area in the middle, managing all circuits/workspaces with a tab-structure.
Canvas - Placed on each tab, this is were you interact and see the components.

Whenever an action is to be done, the view calls on respective methods in the master controller. The exception is the menu and toolbar where actions are first sent to the ActionController which in turn calls on MasterController.

2.1.2 User-interaction

Creating new circuits, loading etc. are done via the menu or toolbar. However most of the interaction are done in the workspace where you build circuits. Interactions are mostly done with the mouse via left and right mouseclicks done in Canvas.

2.1.3 Circuits

Circuits basically consists of collections of gates. A circuit's main purpose is to manage those gates and update those, whenever prompted, in the right order to ensure correct output values.

2.1.4 Components

Components are the meat of circuits and contains output values, booleans, and inputs. Each input is a class containing a reference to the connected gate and which output port of the gate it is connected to. When prompted to update the component applies boolean logic to its input values and sets its outputs accordingly.

More advanced features of the components is that they can detect which connected components that in turn, somewhere, is connected back to themselves.

All gates extend the AbstractCircuitGate class and implements template methods for the different characteristics of logic gates.

2.1.5 Draw handling

Whenever a model has changed, i.e. component has been added, component has been moved, a new model is created etc, an event is fired from ModelManager. The view, which listens to ModelManager, picks up this event, organizes tabs based on the models existing and draws the currently active workspace to the tab's Canvas. Drawing of components and backgrounds are

delegated to a dedicated class.

2.1.6 Save format

To save circuits to files we use our own script-like standard which is easy to use, not only in our own implementation, but also by other software. You can even edit and create circuits manually since they are just a list of commands. One command for adding a component and one for connecting said components.

2.2 Software decomposition

2.2.1 General

- Controller - Controller package contains a MasterController and an ActionController. These are controllers to bridge view and model.
- Model - Our model and the modelmanager handling multiple models.
- Model.components - Contains logic gates and special components such as clock generator and constant values.
- Utilities - General utilities.
- View - Our basic swing using classes.
- View.draw - Draw logic for drawing logic gates and background for workspaces.
- Main - The application's entry class.
- Common – Abstract interfaces which are not necessarily specified for this application.
- IO - IO class for saving and loading circuits.

2.2.2 Tiers

There are no tiers for now.

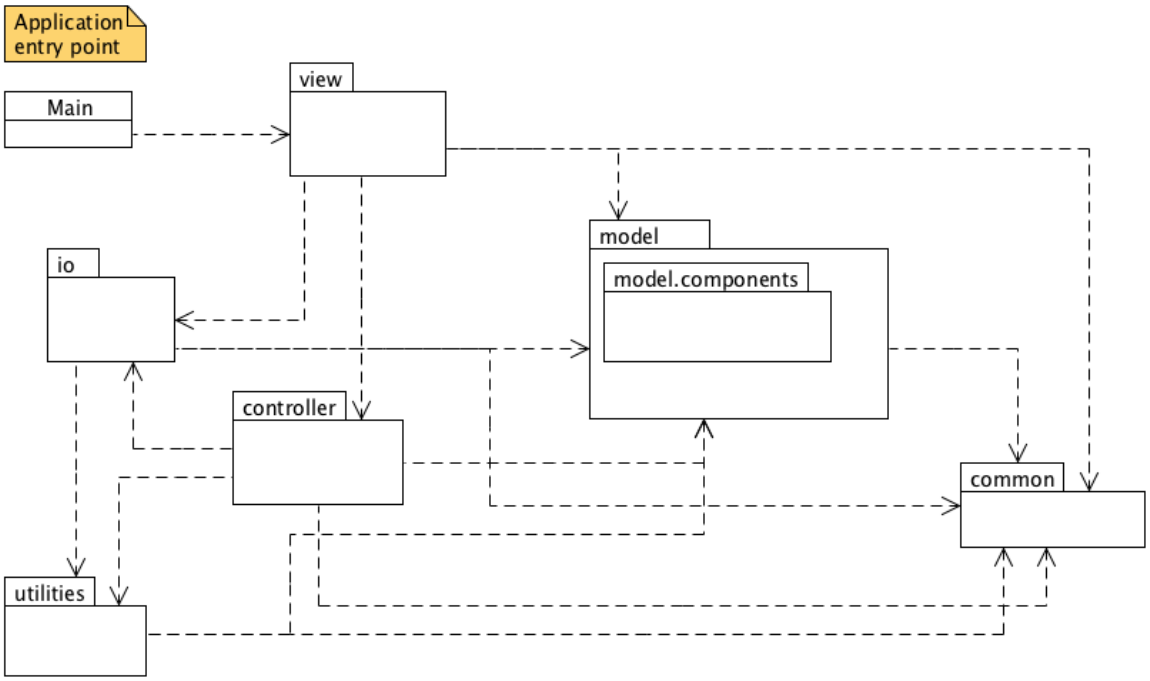
2.2.3 Communication

N/A

2.2.4 Decomposition into subsystems

Our subsystems are the file handling in io and the clipboard management.

2.2.5 Layering



2.2.6 Dependency analysis

Dependencies are shown in figure above. STAN eclipse plugin has been used to analyse structure. There are no circular dependencies between packages or internal circular dependencies inside packages.

2.3 Concurrency issues

N/A. This is a mainly a singlethreaded application.

2.4 Persistent data management

All persistent data will be stored in files without a filename extension. The files that will exist is a save file for circuits. This is stored with our own light script-based language and can easily be read by any software.

2.5 Access control and security

N/A

2.6 Boundary conditions

N/A. Application is launched as a normal desktop application.

2.7 Unimplemented

Some functionality is still unimplemented such as undo/redo functionality. This is hopefully going to be implemented in the future with the help of the command pattern. But for now we will have to do without.

2.8 References

Boolean circuit: http://en.wikipedia.org/wiki/Boolean_circuit
The command pattern: http://en.wikipedia.org/wiki/Command_pattern