Design Document

420 UML Builder

Bit_Champions

Development Team

Kyle Hopkins

Jared Mcandrews

Jesse Platts

Vincent Smith

Vincent Viggiano

Overview

1. Introduction

Users require a tool with which to create and edit UML class diagrams. Our editor will provide an environment to satisfy these needs.

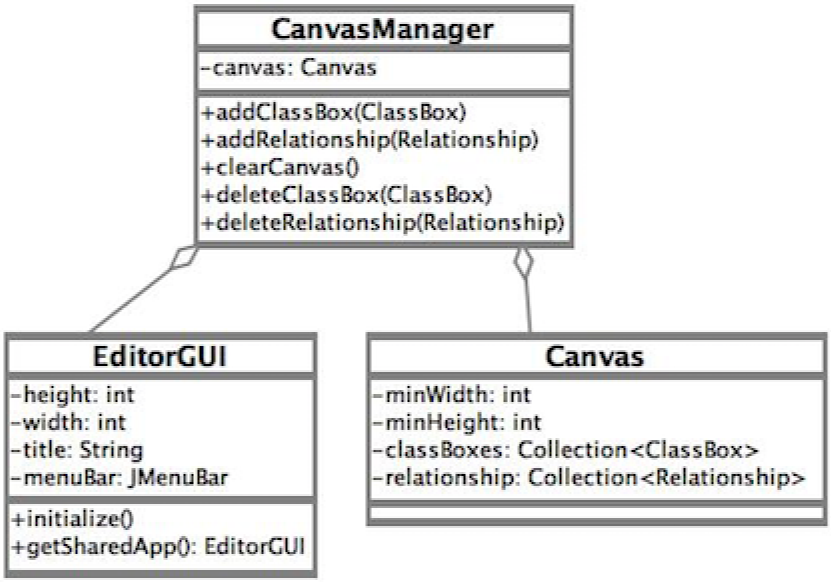
1.1 Purpose

The purpose of this document is to describe the UML diagram in greater detail.

2. Design Overview

2.1 Introduction

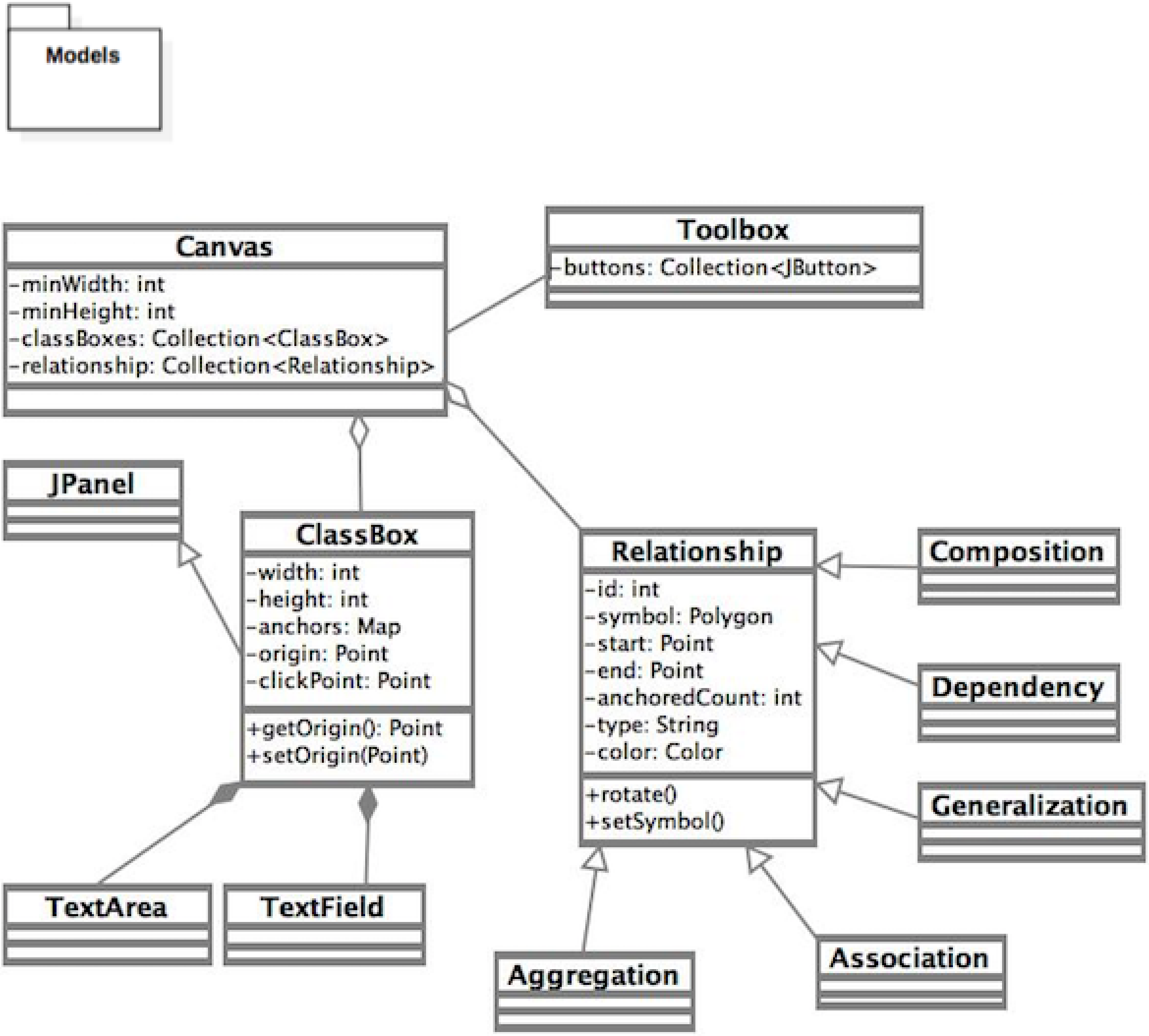
At a high level, this app will be following an MVC architecture.


CanvasManager
-canvas: Canvas
+addClassBox(ClassBox)
+addRelationship(Relationship)
+clearCanvas()
+deleteClassBox(ClassBox)
+deleteRelationship(Relationship)
EditorGUI
-height: int
-width: int
-title: String
-menuBar: JMenuBar
+initialize()
+getSharedApp(): EditorGUI
Canvas
-minWidth: int
-minHeight: int
-classBoxes: Collection<ClassBox>
-relationship: Collection<Relationship>

Our main view will be the EditorGUI, and our CanvasManager will be the leading controller. We will be maintaining a significant portion of our apps state in the Canvas model.

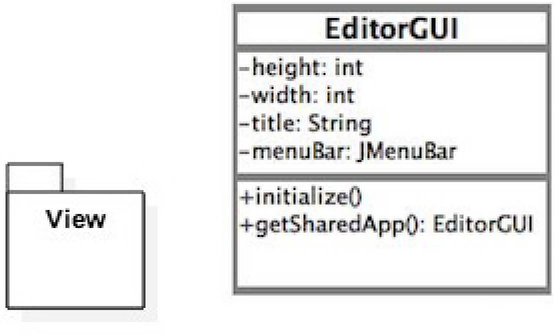
2.2 System Architecture Diagram

2.2.1 Model UML



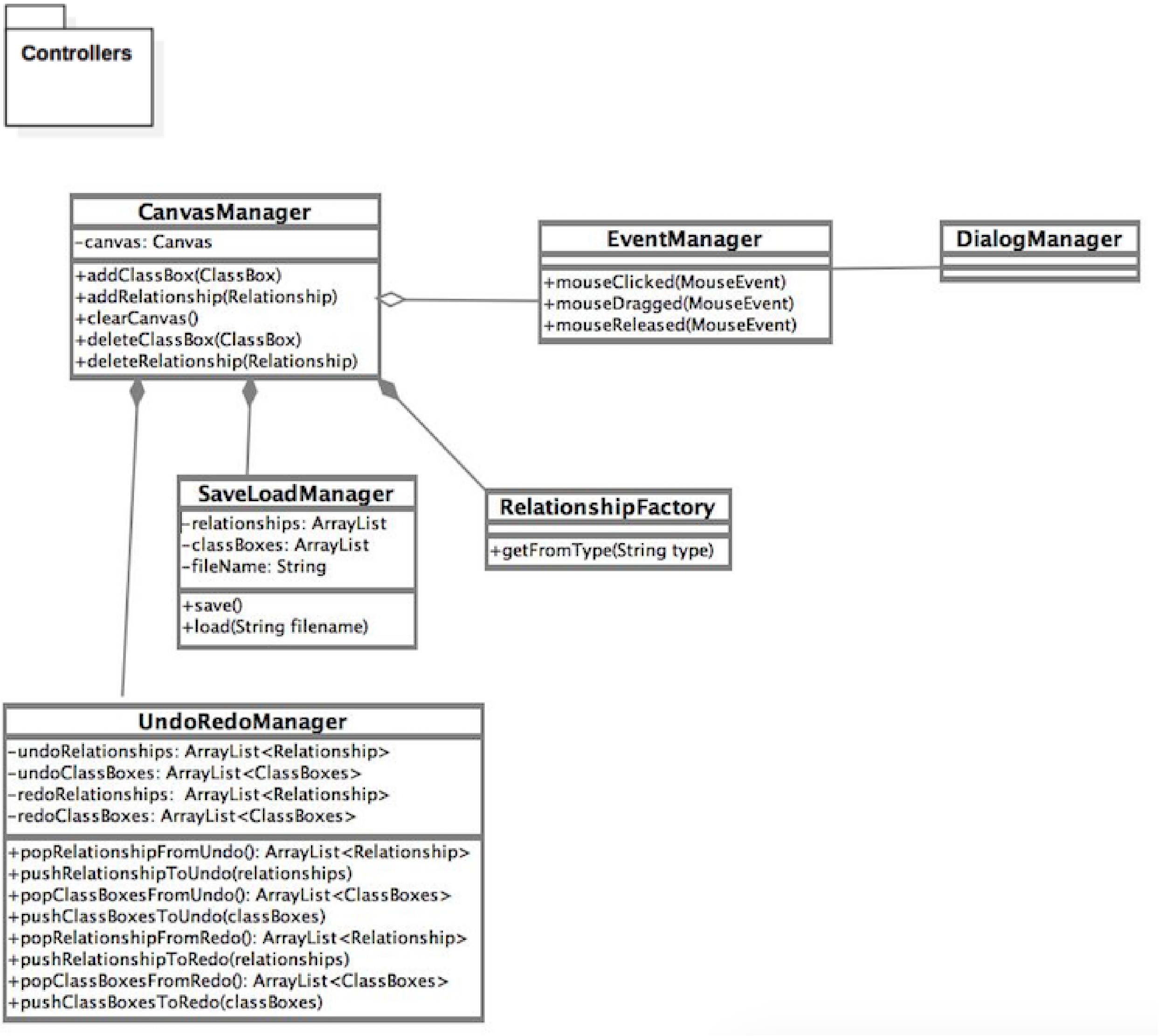
We will be modeling a number of classes for this Editor. We will have a canvas, on which class boxes can be placed. There will be a toolbox to contain buttons for handling events. Class boxes will have 2 text areas (for attributes and methods), and an additional text field for the name of the class. Various relationship types will be derived from a super “Relationship” class.

2.2.2 View UML



Our EditorGUI class will represent the Main class of the program. EditorGUI is a singleton class that will be maintained for the entire runtime.

2.2.3 Controller UML


Controllers
CanvasManager
-canvas: Canvas
+addClassBox(ClassBox)
+addRelationship(Relationship)
+clearCanvas()
+deleteClassBox(ClassBox)
+deleteRelationship(Relationship)
EventManager
+mouseClicked(MouseEvent)
+mouseDragged(MouseEvent)
+mouseReleased(MouseEvent)
DialogManager
SaveLoadManager
-relationships: ArrayList
-classBoxes: ArrayList
-fileName: String
+save()
+load(String filename)
RelationshipFactory
+getFromType(String type)
UndoRedoManager
-undoRelationships: ArrayList<Relationship>
-undoClassBoxes: ArrayList<ClassBoxes>
-redoRelationships: ArrayList<Relationship>
-redoClassBoxes: ArrayList<ClassBoxes>
+popRelationshipFromUndo(): ArrayList<Relationship>
+pushRelationshipToUndo(relationships)
+popClassBoxesFromUndo(): ArrayList<ClassBoxes>
+pushClassBoxesToUndo(classBoxes)
+popRelationshipFromRedo(): ArrayList<Relationship>
+pushRelationshipToRedo(relationships)
+popClassBoxesFromRedo(): ArrayList<ClassBoxes>
+pushClassBoxesToRedo(classBoxes)

The majority of interactions between the user and our system will be funnelled through CanvasManager. CanvasManager will own an UndoRedoManager for undoing or redoing user actions, a SaveLoadManager for saving and loading .sav files, and a RelationshipFactory for adding relationships to the canvas. CanvasManager will also have an EventManager for monitoring user interaction. EventManager will have a loose association with DialogManager for displaying information to the user.

2.3 System Interface

2.3.1 User Interface

The user interface will consist of a menu bar, a toolbox, and a canvas to work on. Users will use the menu bar to save and open .sav files, save .jpg files, and close the application. Users will use the tool box to add and delete elements to and from the canvas, clear the canvas, and undo or redo actions. The canvas will allow users to drag elements across it’s surface. Users will also be able to place different relationships onto the canvas which can in turn be moved, rotated, and anchored to class boxes.