Authoring Application

Design Document

Group 2

EECS 2311

Final Submission

August 21st 2018

1. Table Of Contents

2. Class Design

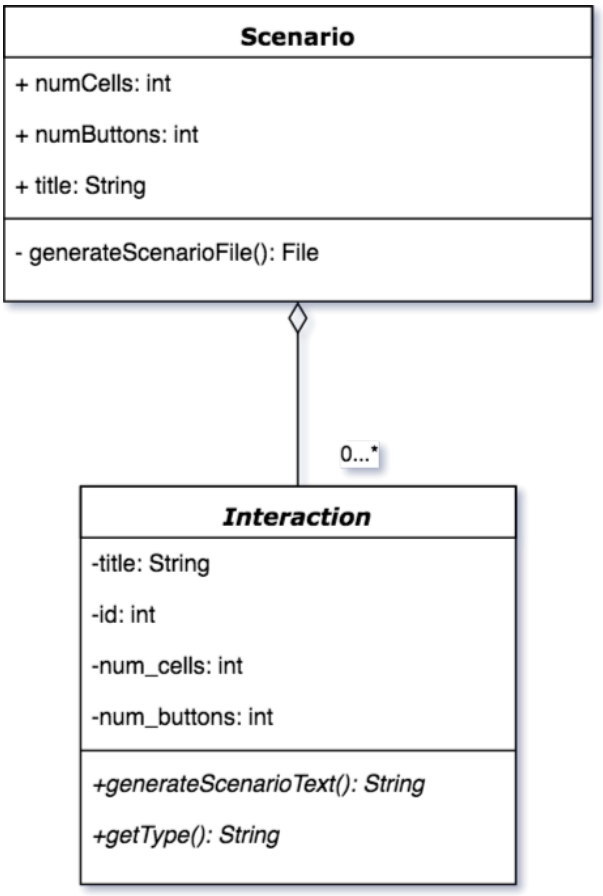

Figure 1. Class Diagram demonstrating relationship between scenario and interaction

1.1 Classes

The primary classes of our application are the Scenario class and the Interaction class.

The scenario class has one or more Interaction objects in a list to keep track of each of the interactions that the user has added to a given scenario, and the application enables the user to access the different properties of the interactions through the interface. This class also holds scenario-wide properties such as number of buttons, number of braille cells, and the title of the scenario. By storing all of the interactions and information about the scenario in this class, we are able to create the scenario file from this object in a very scalable manner.

The interaction class is an abstract class from which each of the interactions (read, voice, display braille, etc.) will implement. Each interaction must hold different data, but when compiling the scenario text file, they must all be able to create the necessary string to insert. Each interaction has a corresponding view to be displayed in the main application window when the user selects it from the list.

1.2 Methods

The methods that create the scenario file from the Scenario class, as well as the Scenario constructor that takes a scenario file for input are the main workhorses of the scenario portion of the model.

The interaction classes are all required to be able to create their own translated scenario text through the class. Since each Interaction will be different, the parent Interaction class has been made abstract so that each type of interaction can implement their own method.

The generateScenarioFile() method from the Scenario class creates the actual simulator-compatible scenario file. This file is created by utilizing the generateScenarioText() method that is implemented by the Interaction classes.

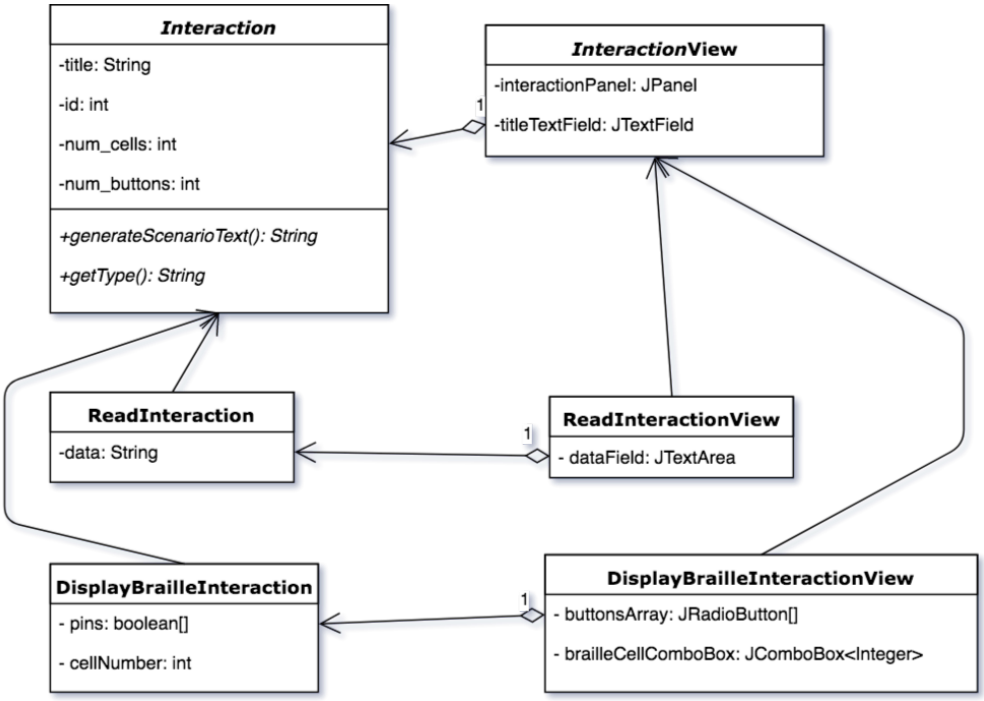

Figure 2 - Inheritance structure for the read and display braille interaction classes

3. Runtime

There are a number of important objects that are created at runtime.

Object	**Function**
StartupPane	UI object which shows the interface that the user is first shown. Consists of the primary functions (New, Open, Exit) to allow the user to quickly begin working on their scenario. Also contains the menu bar which is persistent through all interfaces of the application.
MainViewController	Controller which responds to UI events triggered by actions of the user. Responsible for instantiating and passing control to the Editor classes for creating and opening scenarios. Holds functionality of saving / running scenario files (through the menu bar).
MainViewModel	Model to contain simple information about the state of the application so the UI can display the correct configuration of menu items (if scenario is open, application title changes, etc).
Scenario	The main model which encapsulates the list of interactions and is responsible for constructing scenario files and deconstructing scenario files into their respective

	objects with the correct configurations. Interaction list uses a custom written ListModel to efficiently add, remove, and reorder list items.
EditorPane	The UI object which replaces the StartupPane UI when the user is in editing mode. It has the relevant controls for creating, removing and reordering interactions as well as additional save and run functionalities.
EditorController	Control passes over to the EditorController from the MainViewController once the user is in editing mode (menu items still trigger MainViewController functions however). The EditorController responds to UI events on the EditorPane and communicates with the Scenario model to correctly display and update both the UI and model.
InteractionView	Embedded within the EditorController is a pane for configuring each interaction item. Each interaction object has a different view which has the appropriate controls for configuring it (ie. text fields for read, audio recording / playback controls for voice, etc).
InteractionModel	Each interaction also has a different model which stores the configuration options for them. They also have their own implementations of the generateScenarioText() functions to correctly output the textual representations of each interaction with their corresponding configs.

The sequence diagram below depicts their interaction with each other.

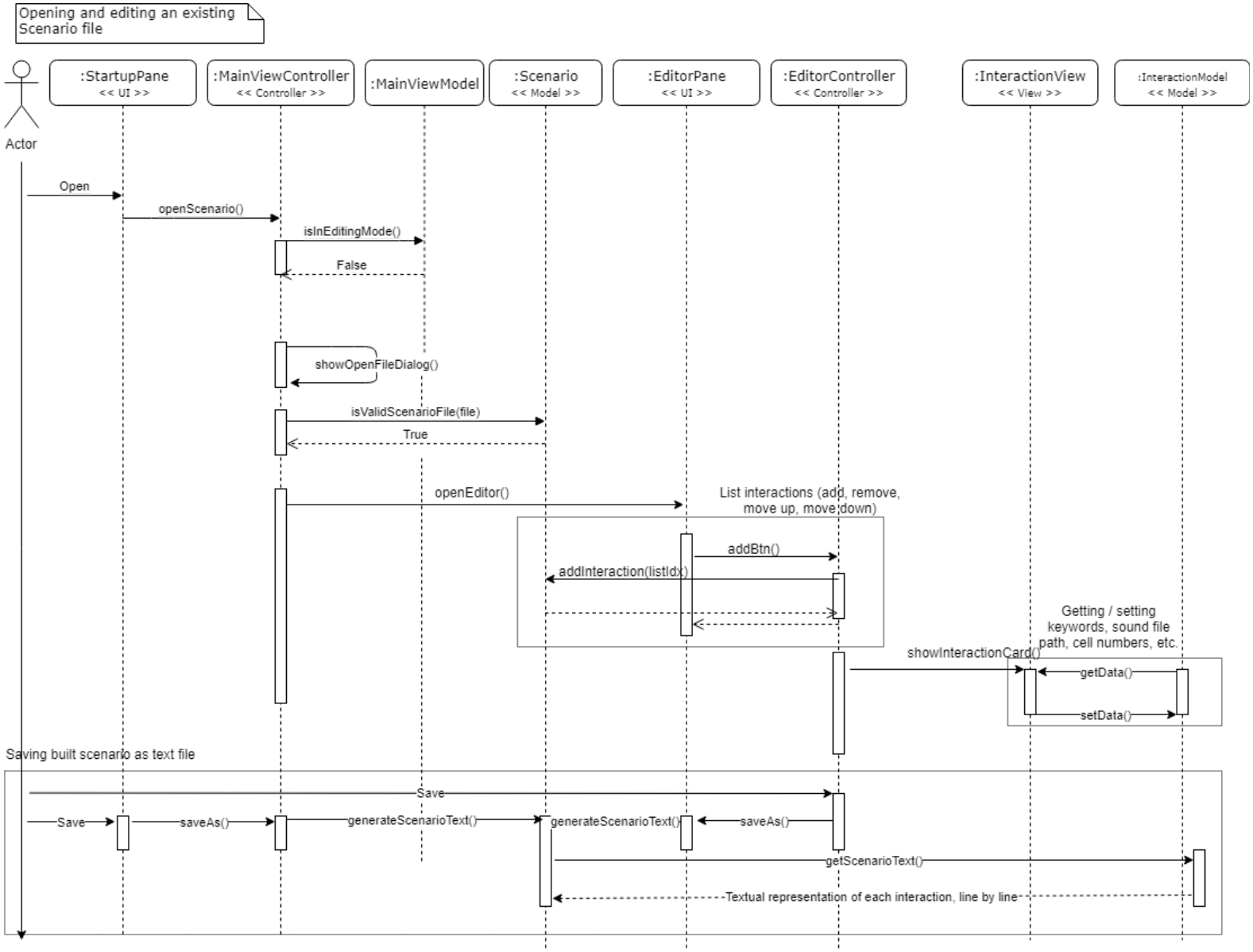

Figure 3. Sequence diagram depicting the interaction of objects during runtime