The *Regio Vinco* ™ The Map Maker Software Design Description

Author: Richard McKenna
Daye Eun
Debugging Enterprises™
May, 2018
Version 1.0

Abstract: This document describes the software design for The Regio Vinco The Map Maker, a map editor tool in development as part of CSE219 project.

Based on IEEE Std 830™-1998 (R2009) document format

1 Introduction

This is the Software Design Description (SDD) for the ***Regio Vinco*** Map Maker application. Casual games should be easy to learn but hard to master. Serious (i.e. learning) games should impart a deeper understanding of real world information through gameplay. Both game styles should entertain. In this project, we aim to develop a tool to help with the construction of a serious, casual game titled Regio Vinco.

The mechanics of gameplay for casual games should simple, only requiring the player to perform simple actions like mouse clicks, letting the user easily pick up the game and put it down. This is particularly true of casual games played on mobile platforms. Regio Vinco will be a desktop game that would also be appropriate to port to a mobile or Web platform. It will be serious, it will be casual, and it will be fun.

The Regio Vinco game is a game for learning about the world and its geography. Through playing the game, the player can practice their knowledge of continent, nation, state, and province borders, as well as capitals, flags, and leaders. The game will let the user select the map of their choice to play and will keep track of player accomplishments for all provided maps. The construction of maps, therefore, is an important component of developing this game application.

1.1 **Purpose**

The purpose of this document is to specify how our ***Regio Vinco*** Map Maker should look and operate. The intended audience for this document is all the members of the development team, from the game designers to the artists, to the software engineers and level designers. This document serves as an agreement among all parties and a reference for how the map creation tool should ultimately be constructed. Upon completing the reading of this document, one should clearly visualize how the game application will look and operate as well as understand the game mechanics and rules. In addition, one should understand how the map editor will look and operate and how it will benefit the development process such that we may create consistent high-quality maps.

1.2 Scope

For this project the goal is to let a map designer easily make maps that look precisely the way they want. This means making maps that will work properly in the game, will be tied to game data, and will have appropriate use of colors, borders, and images as the game requires.

1.3 Definitions, acronyms, and abbreviations

Casual Game – A game that can be picked up and played quickly, where 15 minutes of entertainment may suffice, but typically where richer experiences are also possible. Easy to pick up, easy to put down.

Class Diagram – A UML document format that describes classes graphically. Specifically, it describes their instance variables, method headers, and relationships to other classes.

IEEE – Institute of Electrical and Electronics Engineers, the "world's largest professional association for the advancement of technology".

Framework – In an object-oriented language, a collection of classes and interfaces that collectively provide a service for building applications or additional frameworks all with a common need.

GUI – Graphical User Interface, visual controls like buttons inside a window in a software application that collectively allow the user to operate the program.

Serious Game – A game that proves an experience for the player through which one learns out of game knowledge of some value.

UML – Unified Modeling Language, a standard set of document formats for designing software graphically.

Use Case Diagram – A UML document format that specifies how a user will interact with a system. Note that these diagrams do not include technical details. Instead, they are fed as input into the design stage (stage after this one) where the appropriate software designs are constructed based in part on the Use Cases specified in the SRS.

1.4 References

IEEE Std 830TM-1998 (R2009) – IEEE Recommended Practice for Software Requirements Specification

The Regio VincoTM The Map Maker SRS– Debugging Enterprises' Software Requirements Specification for the ***Regio Vinco*** Map Maker application.

1.5 Overview

This Software Design Description document provides a working design for the ***Regio Vinco*** Map Maker software application as described in the ***Regio Vinco*** Map Maker Software Requirements Specification. Note that all parties in the implementation stage must agree upon all connections between components before proceeding with the implementation stage. Section 2 of this document will provide the Package-Level Viewpoint, specifying the packages and frameworks to be designed. Section 3 will provide the Class-Level Viewpoint, using UML Class Diagrams to specify how the classes should be constructed. Section 4 will provide the Method-Level System Viewpoint, describing how methods will interact with one another. Section 5 provides deployment information like file structures and formats to use. Section 6 provides supporting information, such as a Table of Contents, an Index, and References. Note that all UML Diagrams in this document were created using the VioletUML editor.

2 Package-Level Design Viewpoint

2.1 Application overview

The ***Regio Vinco*** Map Maker will be designed and developed in tandem. Figure 2.1 specifies all the components to be developed and places all classes in home packages.

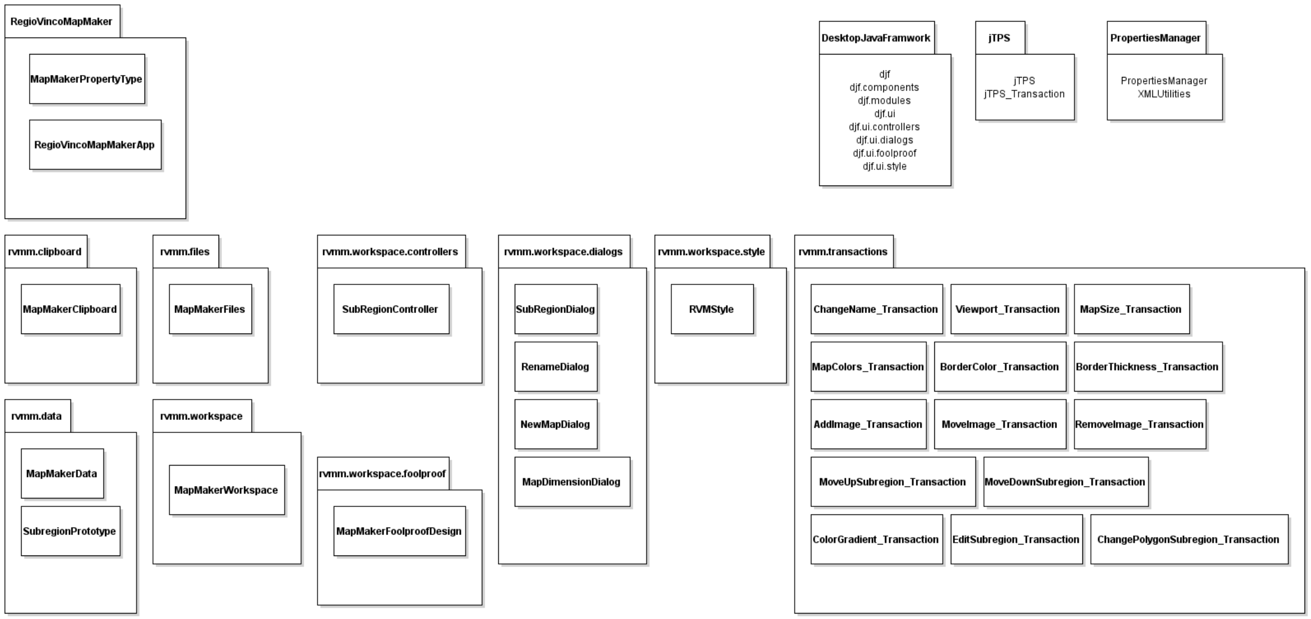

Figure 2.1: Design Packages Overview

2.2 Java API Usage

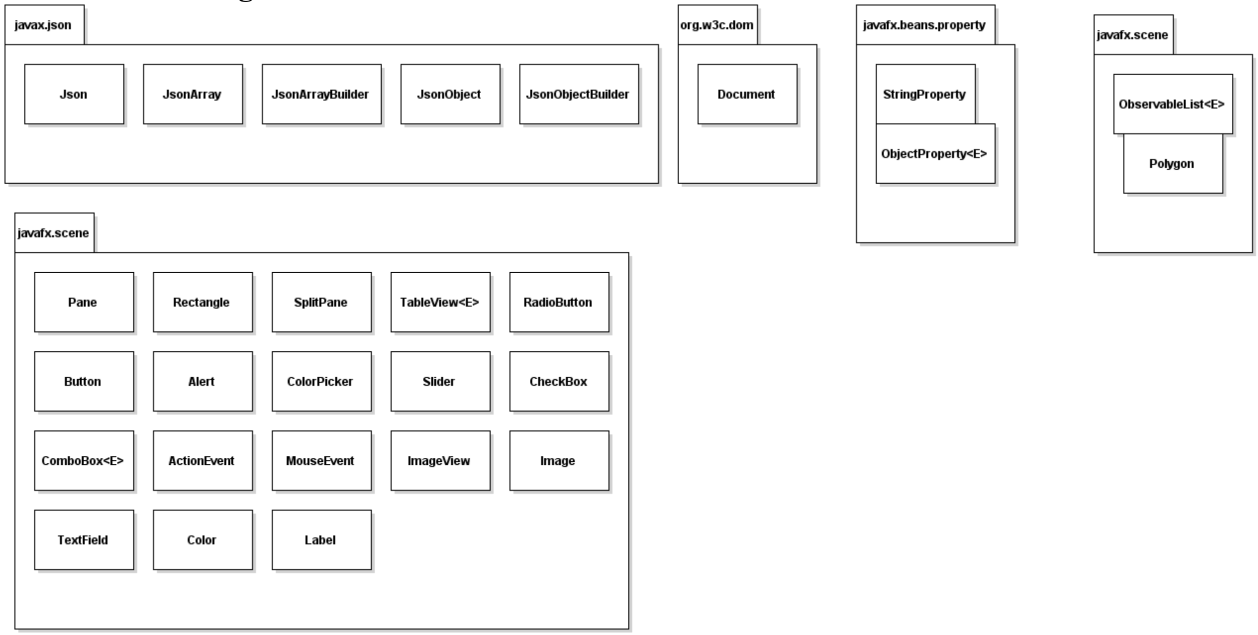

Figure 2.2: Java API Classes and Packages To Be Used

3 Class-Level Design Viewpoint

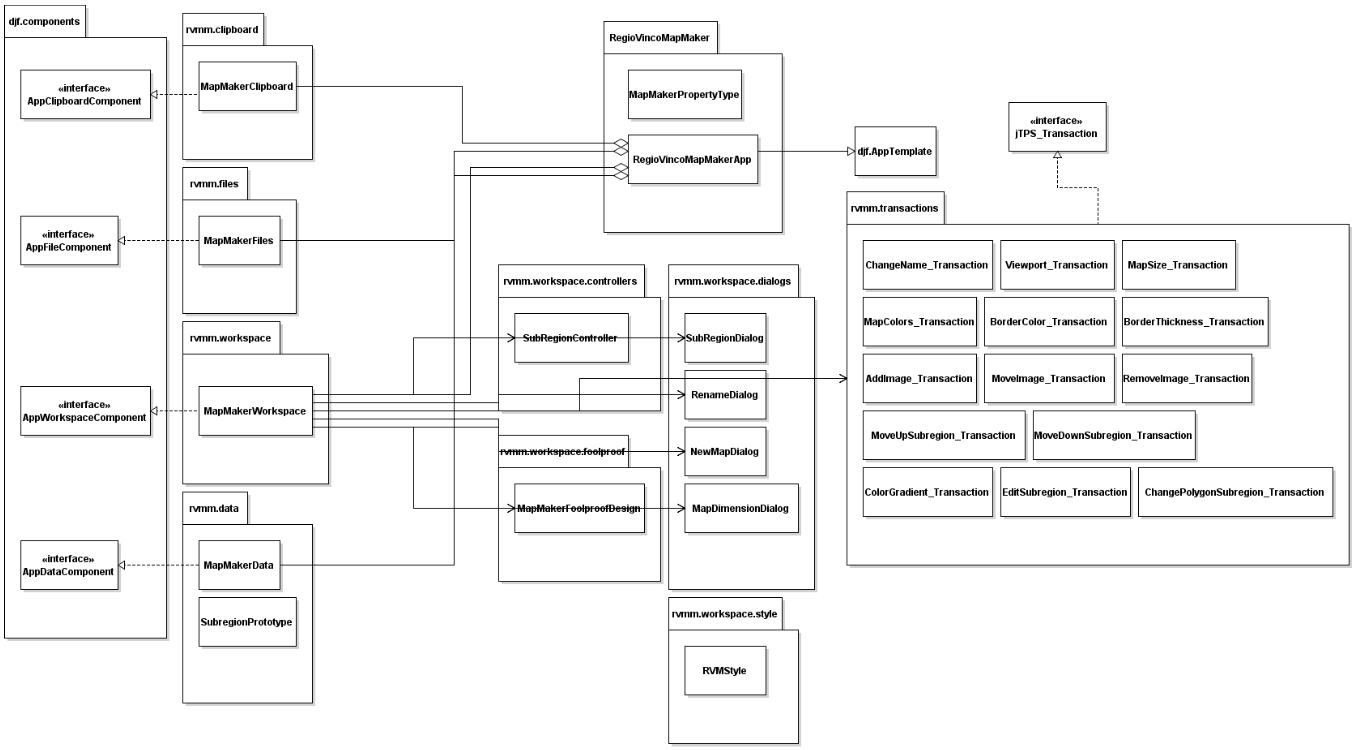

Figure 3.1: ***Regio Vinco*** **Map Maker Overview UML Class Diagram**

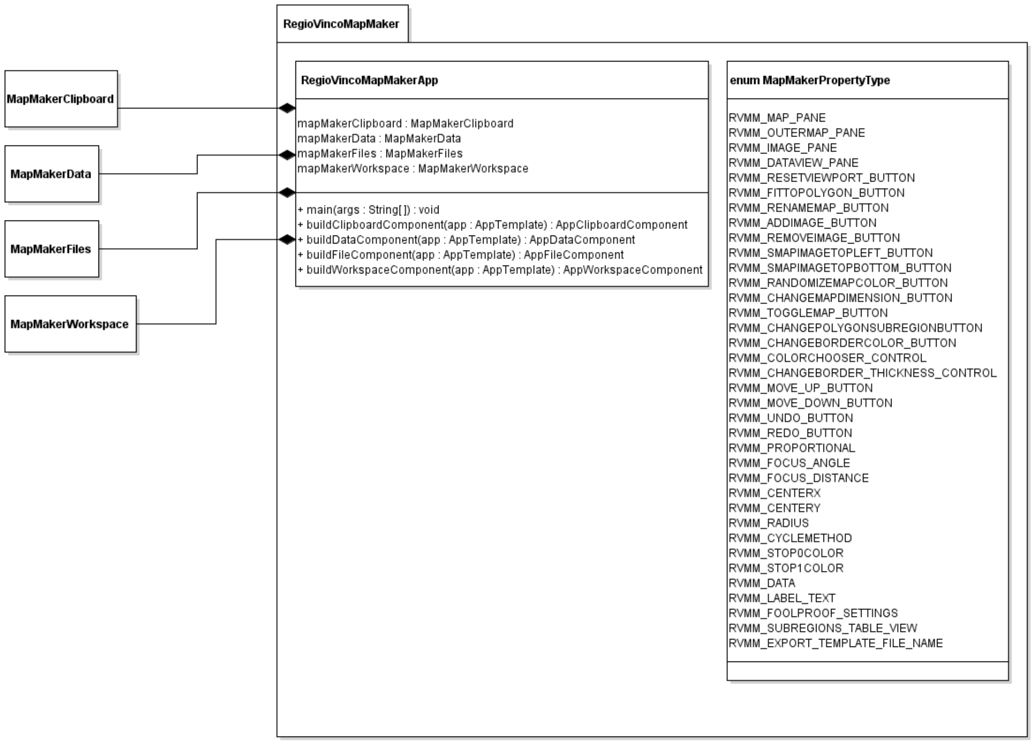
RegioVincoMapMaker
MapMakerClipboard
MapMakerData
MapMakerFiles
MapMakerWorkspace
RegioVincoMapMakerApp
mapMakerClipboard : MapMakerClipboard
mapMakerData : MapMakerData
mapMakerFiles : MapMakerFiles
mapMakerWorkspace : MapMakerWorkspace
+ main(args : String[ ]) : void
+ buildClipboardComponent(app : AppTemplate) : AppClipboardComponent
+ buildDataComponent(app : AppTemplate) : AppDataComponent
+ buildFileComponent(app : AppTemplate) : AppFileComponent
+ buildWorkspaceComponent(app : AppTemplate) : AppWorkspaceComponent
enum MapMakerPropertyType
RVMM_MAP_PANE
RVMM_OUTERMAP_PANE
RVMM_IMAGE_PANE
RVMM_DATAVIEW_PANE
RVMM_RESETVIEWPORT_BUTTON
RVMM_FITTOPOLYGON_BUTTON
RVMM_RENAMEMAP_BUTTON
RVMM_ADDIMAGE_BUTTON
RVMM_REMOVEIMAGE_BUTTON
RVMM_SMAPIMAGETOPLEFT_BUTTON
RVMM_SMAPIMAGETOPBOTTOM_BUTTON
RVMM_RANDOMIZEMAPCOLOR_BUTTON
RVMM_CHANGEMAPDIMENSION_BUTTON
RVMM_TOGGLEMAP_BUTTON
RVMM_CHANGEPOLYGONSUBREGIONBUTTON
RVMM_CHANGEBORDERCOLOR_BUTTON
RVMM_COLORCHOOSER_CONTROL
RVMM_CHANGEBORDER_THICKNESS_CONTROL
RVMM_MOVE_UP_BUTTON
RVMM_MOVE_DOWN_BUTTON
RVMM_UNDO_BUTTON
RVMM_REDO_BUTTON
RVMM_PROPORTIONAL
RVMM_FOCUS_ANGLE
RVMM_FOCUS_DISTANCE
RVMM_CENTERX
RVMM_CENTERY
RVMM_RADIUS
RVMM_CYCLEMETHOD
RVMM_STOP0COLOR
RVMM_STOP1COLOR
RVMM_DATA
RVMM_LABEL_TEXT
RVMM_FOOLPROOF_SETTINGS
RVMM_SUBREGIONS_TABLE_VIEW
RVMM_EXPORT_TEMPLATE_FILE_NAME

Figure 3.2: Detailed ***RegioVinco*****MapMaker UML Class Diagram**

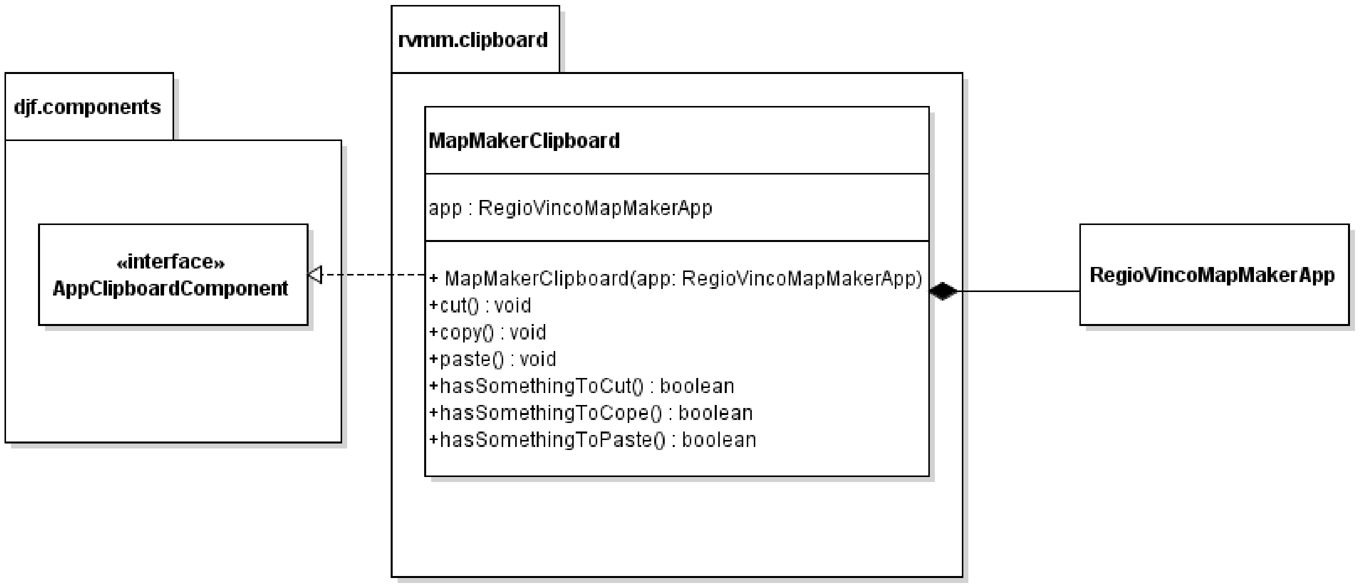
rvmm.clipboard
djf.components
«interface»
AppClipboardComponent
MapMakerClipboard
app : RegioVincoMapMakerApp
+ MapMakerClipboard(app: RegioVincoMapMakerApp)
+cut() : void
+copy() : void
+paste() : void
+hasSomethingToCut() : boolean
+hasSomethingToCope() : boolean
+hasSomethingToPaste() : boolean
RegioVincoMapMakerApp

Figure 3.3: Detailed MapMakerClipboard UML Class Diagram

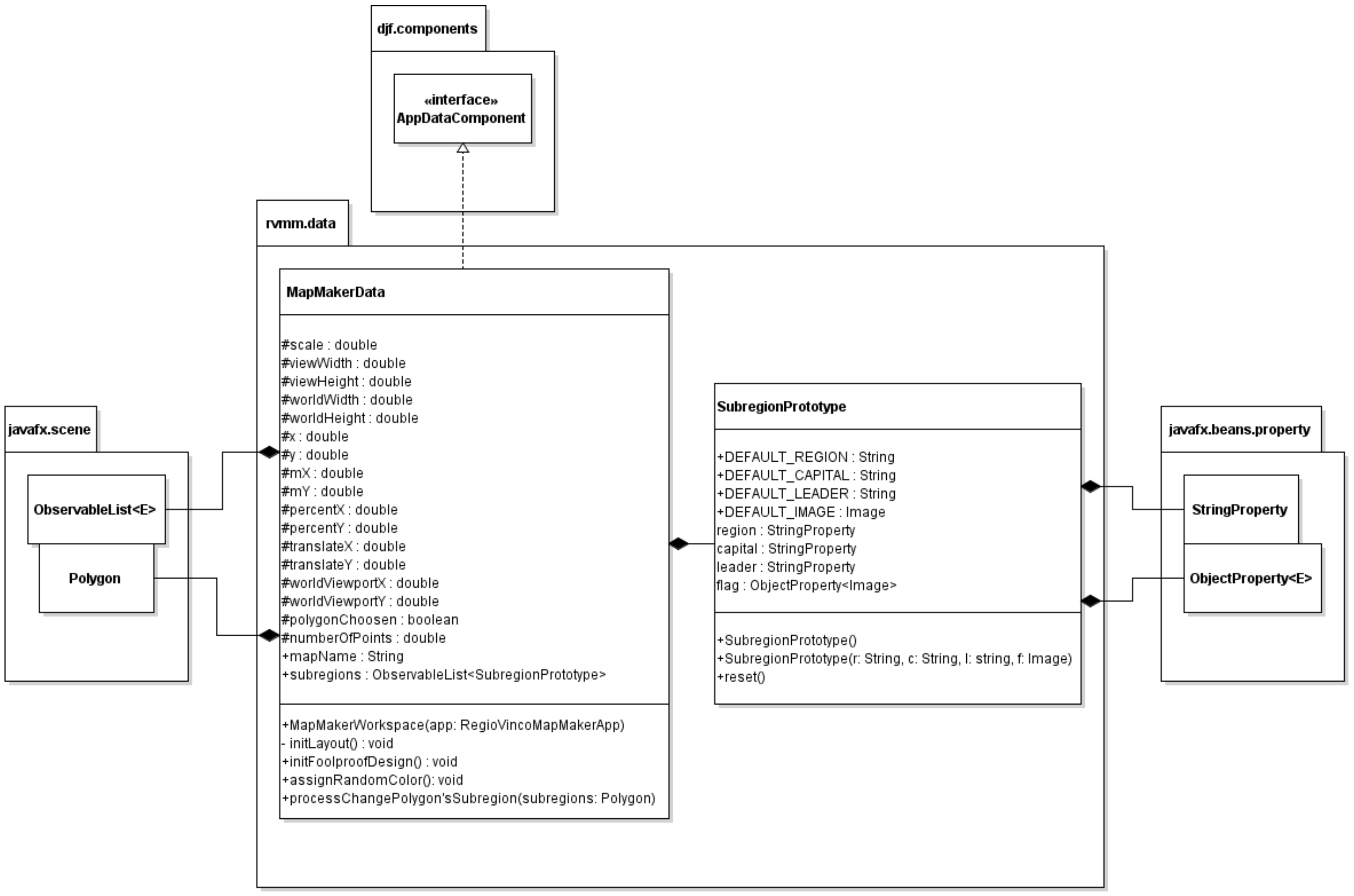

Figure 3.4: Detailed MapMakerData UML Class Diagram

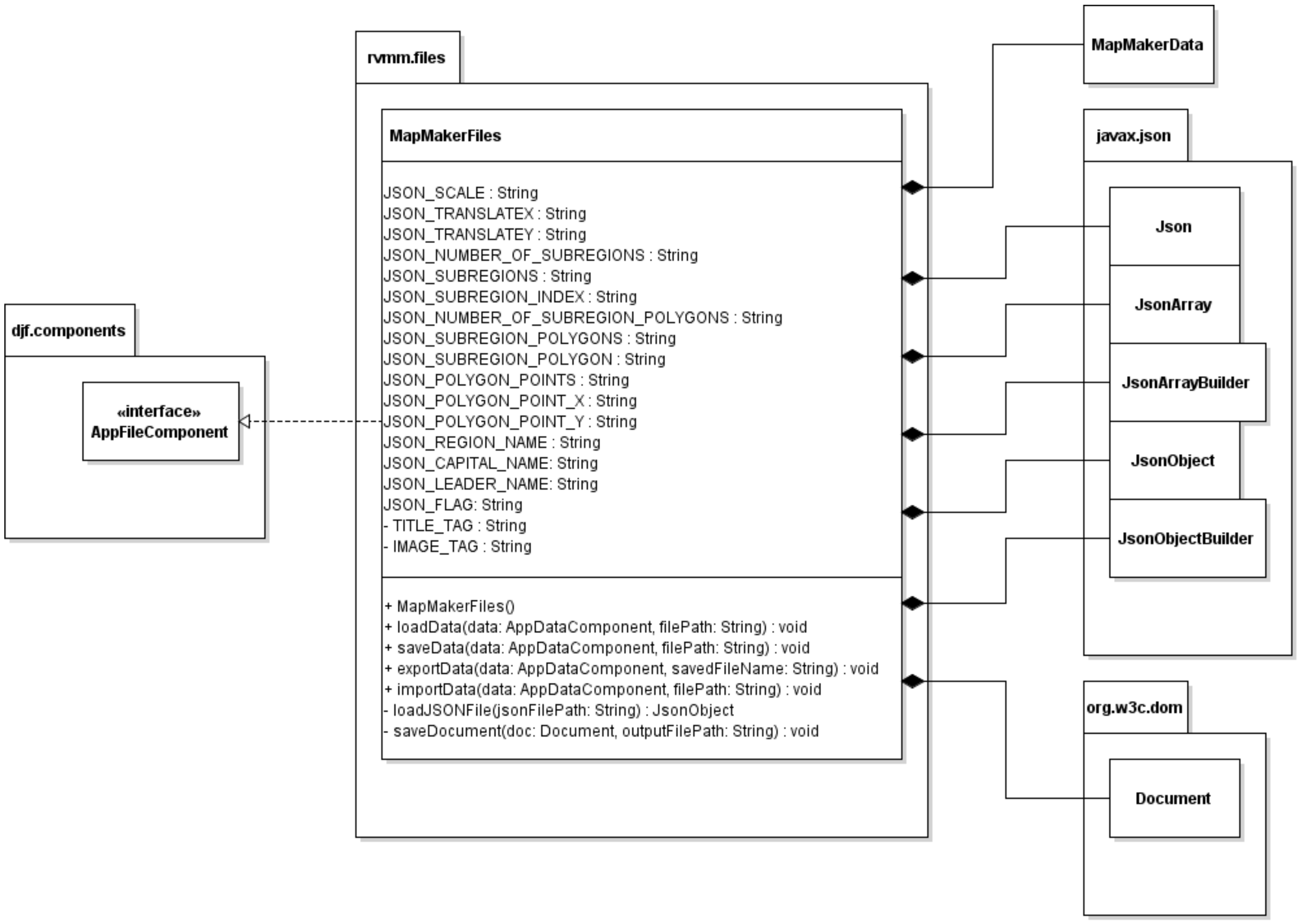
rvmm.files
MapMakerFiles
JSON_SCALE : String
JSON_TRANSLATEX : String
JSON_TRANSLATEY : String
JSON_NUMBER_OF_SUBREGIONS : String
JSON_SUBREGIONS : String
JSON_SUBREGION_INDEX : String
JSON_NUMBER_OF_SUBREGION_POLYGONS : String
JSON_SUBREGION_POLYGONS : String
JSON_SUBREGION_POLYGON : String
JSON_POLYGON_POINTS : String
JSON_POLYGON_POINT_X : String
JSON_POLYGON_POINT_Y : String
JSON_REGION_NAME : String
JSON_CAPITAL_NAME: String
JSON_LEADER_NAME: String
JSON_FLAG: String
- TITLE_TAG : String
- IMAGE_TAG : String
+ MapMakerFiles()
+ loadData(data: AppDataComponent, filePath: String) : void
+ saveData(data: AppDataComponent, filePath: String) : void
+ exportData(data: AppDataComponent, savedFileName: String) : void
+ importData(data: AppDataComponent, filePath: String) : void
- loadJSONFile(jsonFilePath: String) : JsonObject
- saveDocument(doc: Document, outputFilePath: String) : void
djf.components
«interface»
AppFileComponent
MapMakerData
javax.json
Json
JsonArray
JsonArrayBuilder
JsonObject
JsonObjectBuilder
org.w3c.dom
Document

Figure 3.4: Detailed MapMakerFiles UML Class Diagram

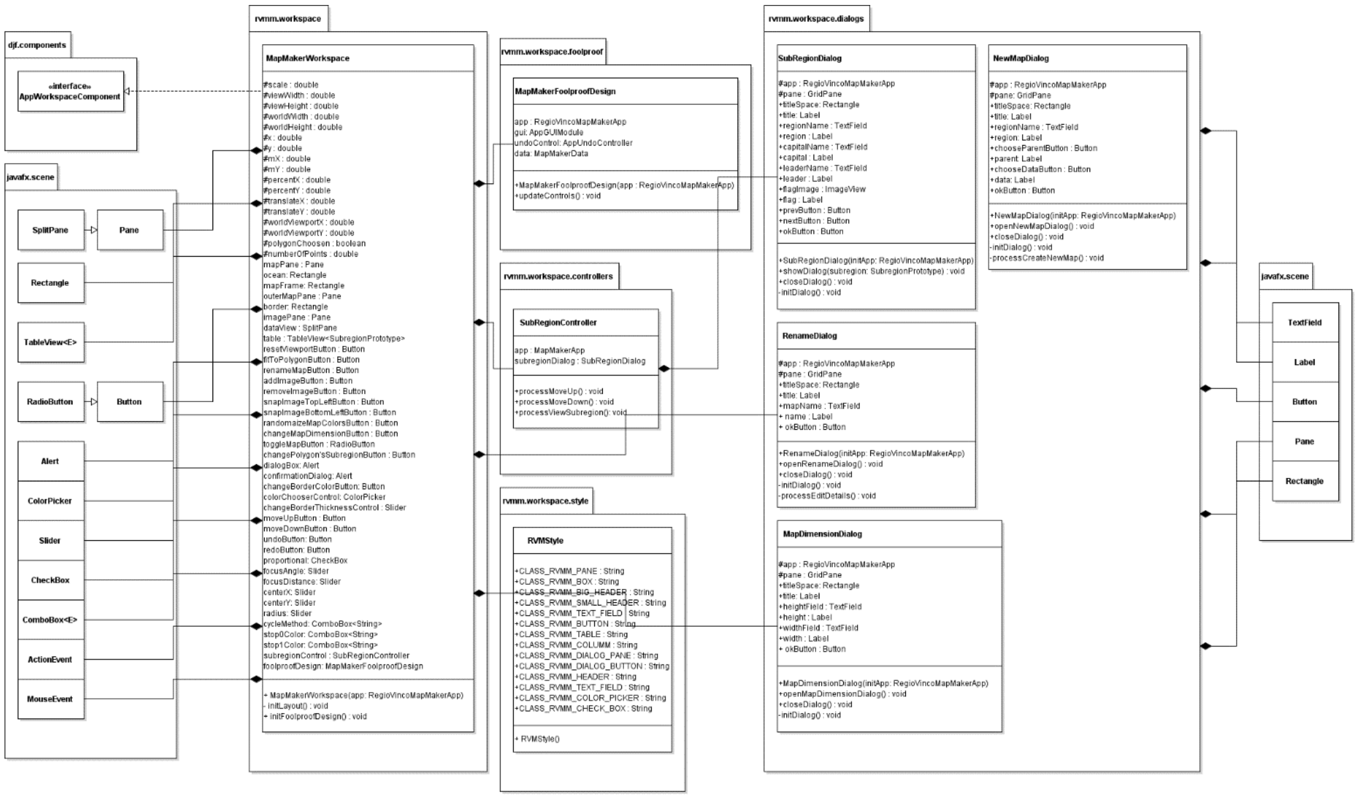

Figure 3.5: Detailed MapMakerWorkspace UML Class Diagram

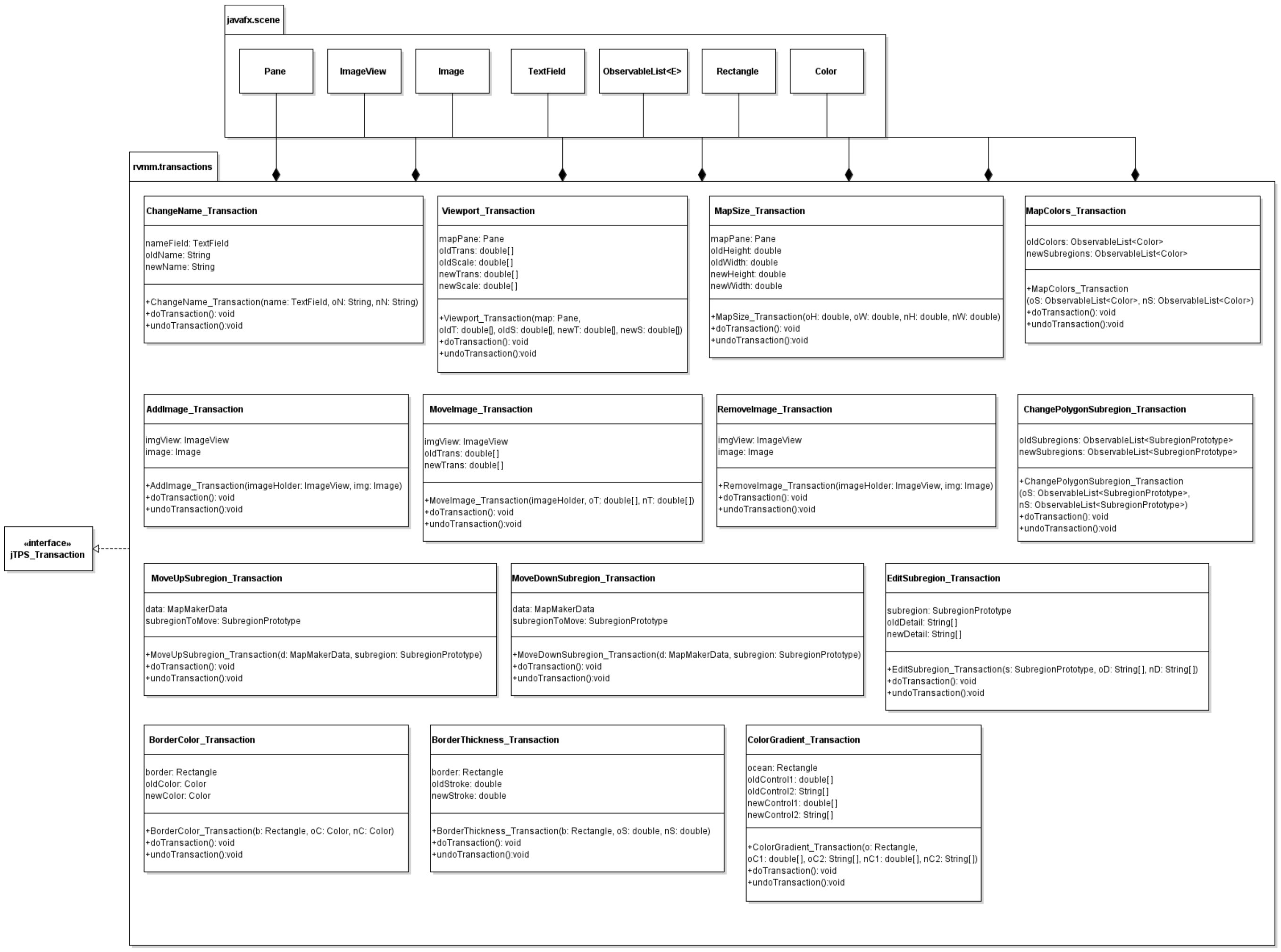

Figure 3.6: Detailed Transactions UML Class Diagram

4 Method-Level Design Viewpoint

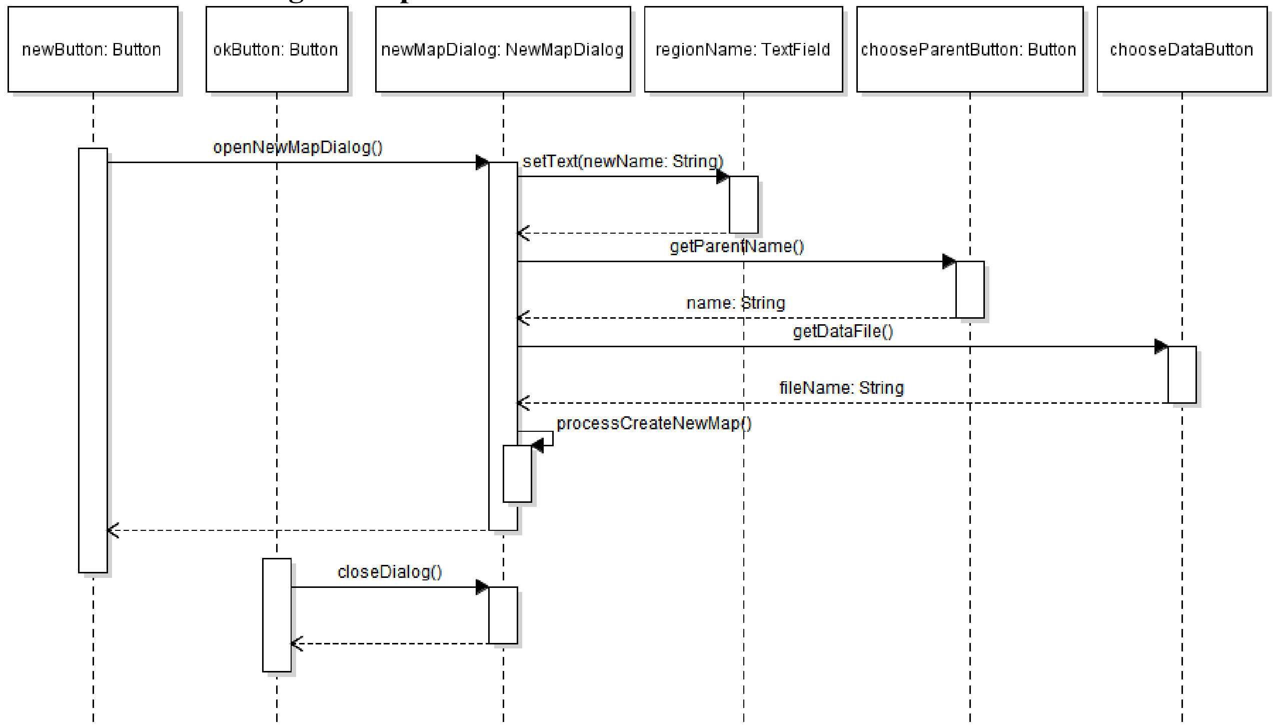

Figure 4.1: Create New Map UML Sequence Diagrams

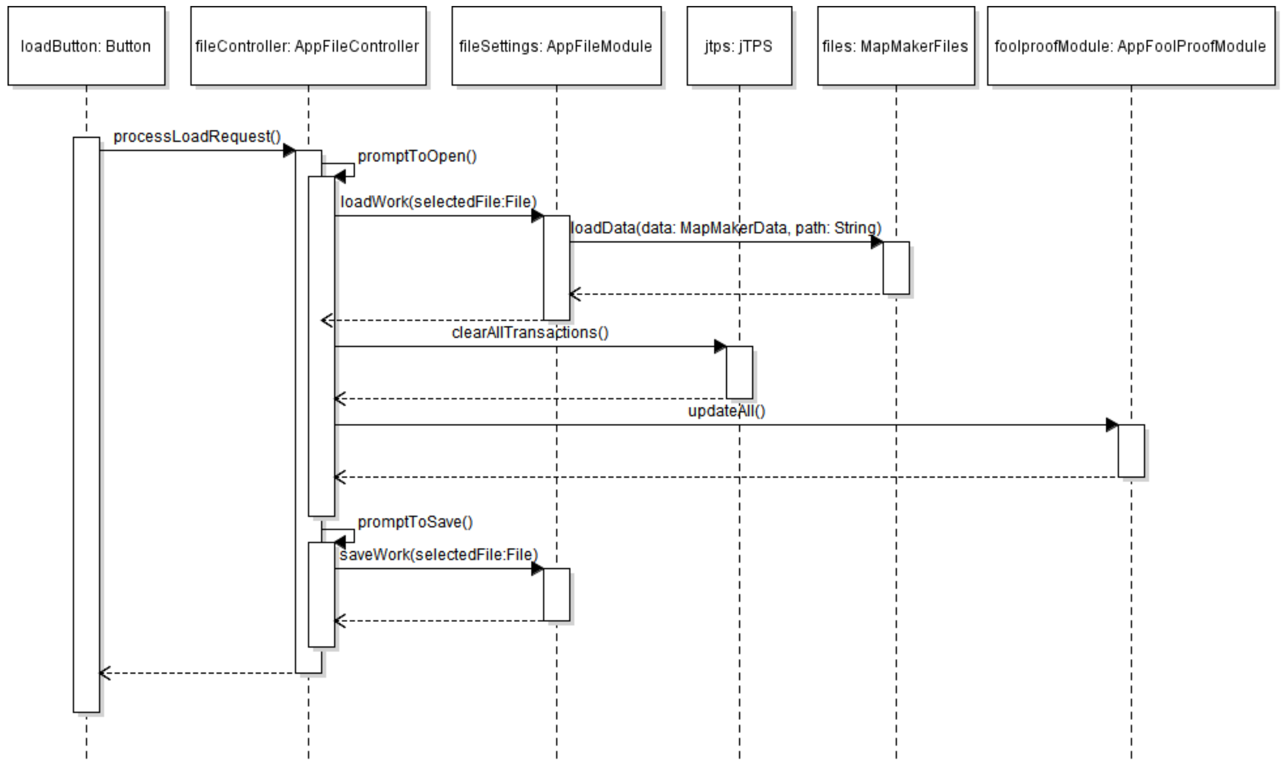
loadButton: Button
fileController: AppFileController
fileSettings: AppFileModule
jtps: jTPS
files: MapMakerFiles
foolproofModule: AppFoolProofModule
processLoadRequest()
promptToOpen()
loadWork(selectedFile:File)
loadData(data: MapMakerData, path: String)
clearAllTransactions()
updateAll()
promptToSave()
saveWork(selectedFile:File)

Figure 4.2: Load Map UML Sequence Diagrams

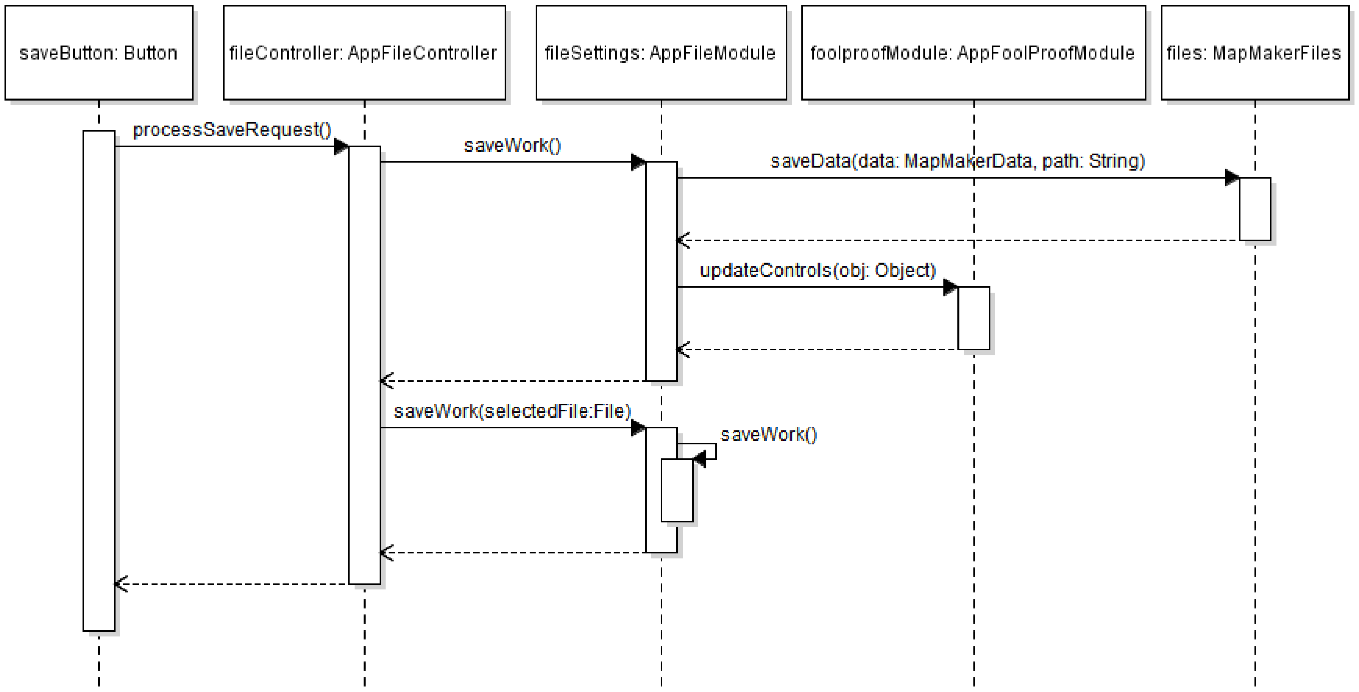
saveButton: Button
fileController: AppFileController
fileSettings: AppFileModule
foolproofModule: AppFoolProofModule
files: MapMakerFiles
processSaveRequest()
saveWork()
saveData(data: MapMakerData, path: String)
updateControls(obj: Object)
saveWork(selectedFile:File)
saveWork()

Figure 4.3: Save Map UML Sequence Diagrams

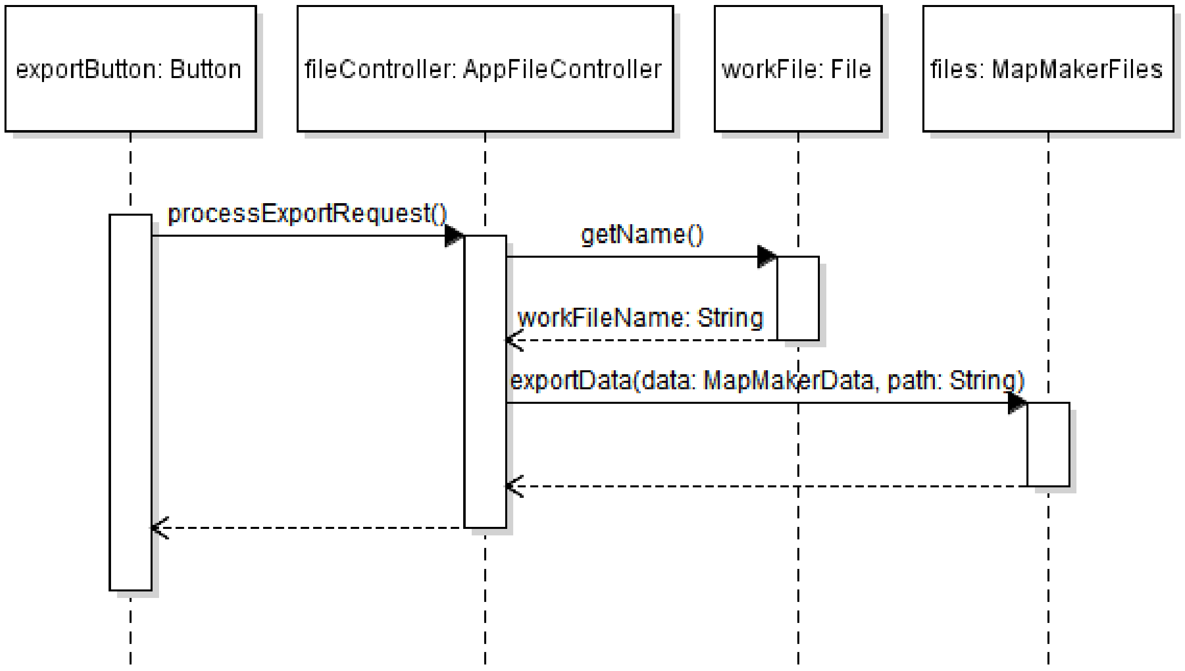

Figure 4.4: Export Map UML Sequence Diagrams

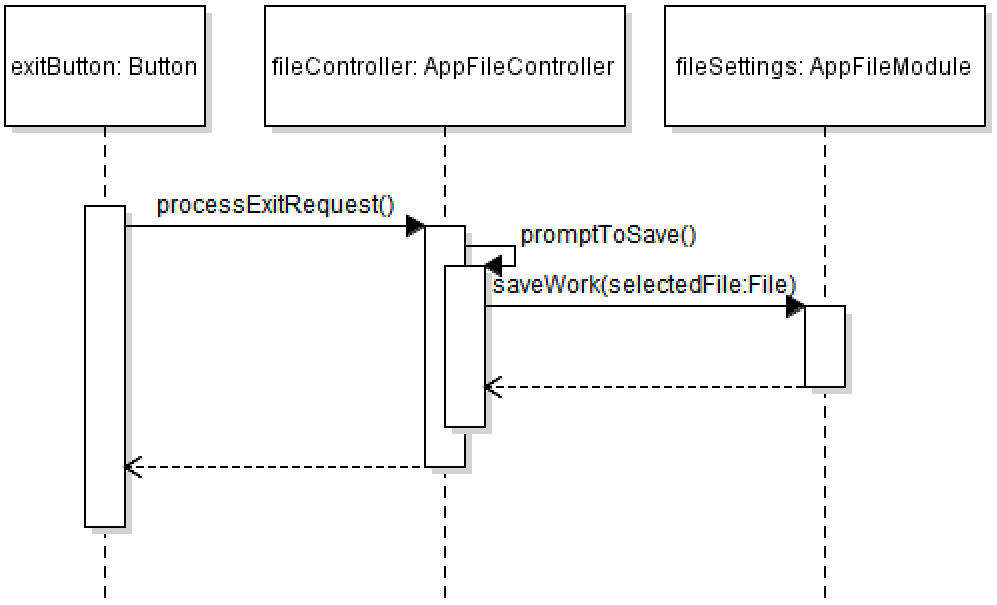
exitButton: Button
fileController: AppFileController
fileSettings: AppFileModule
processExitRequest()
promptToSave()
saveWork(selectedFile:File)

Figure 4.5: Exit Application UML Sequence Diagrams

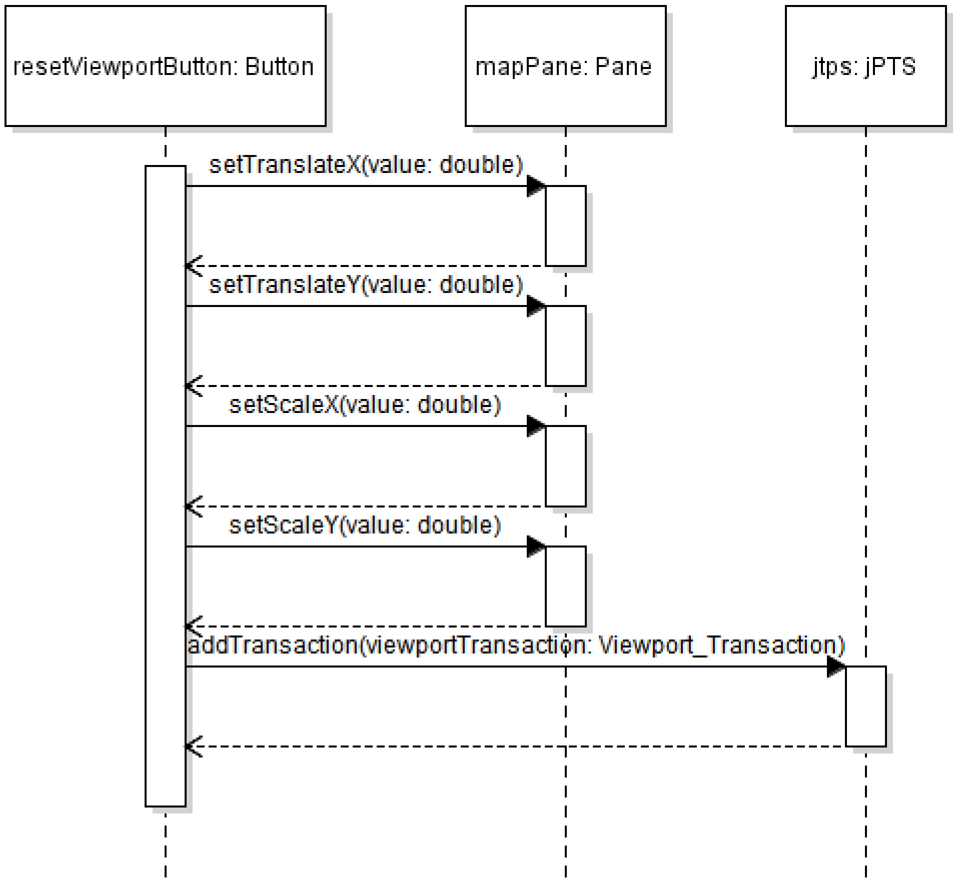
resetViewportButton: Button
mapPane: Pane
jtps: jPTS
setTranslateX(value: double)
setTranslateY(value: double)
setScaleX(value: double)
setScaleY(value: double)
addTransaction(viewportTransaction: Viewport_Transaction)

Figure 4.6: Reset Viewport UML Sequence Diagrams

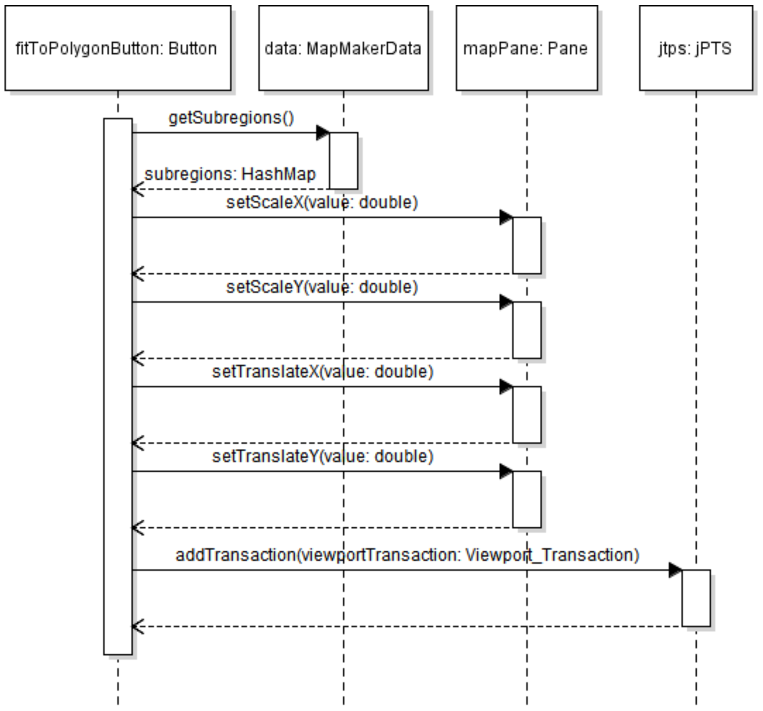
fitToPolygonButton: Button
data: MapMakerData
mapPane: Pane
jtps: jPTS
getSubregions()
subregions: HashMap
setScaleX(value: double)
setScaleY(value: double)
setTranslateX(value: double)
setTranslateY(value: double)
addTransaction(viewportTransaction: Viewport_Transaction)

Figure 4.7: Fit Viewport to Polys UML Sequence Diagrams

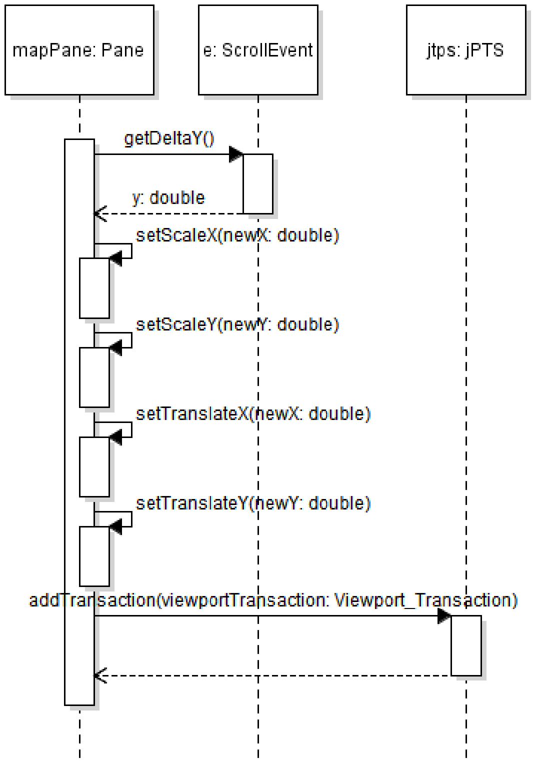
mapPane: Pane
e: ScrollEvent
jtps: jPTS
getDeltaY()
y: double
setScaleX(newX: double)
setScaleY(newY: double)
setTranslateX(newX: double)
setTranslateY(newY: double)
addTransaction(viewportTransaction: Viewport_Transaction)

Figure 4.8: Zoom In/Out UML Sequence Diagrams

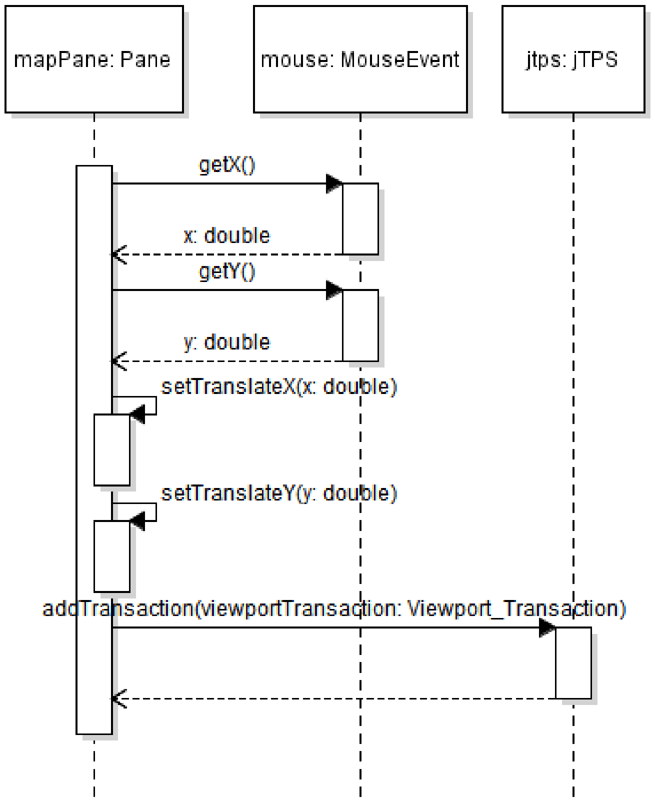

Figure 4.9: Grab and Move Map UML Sequence Diagrams

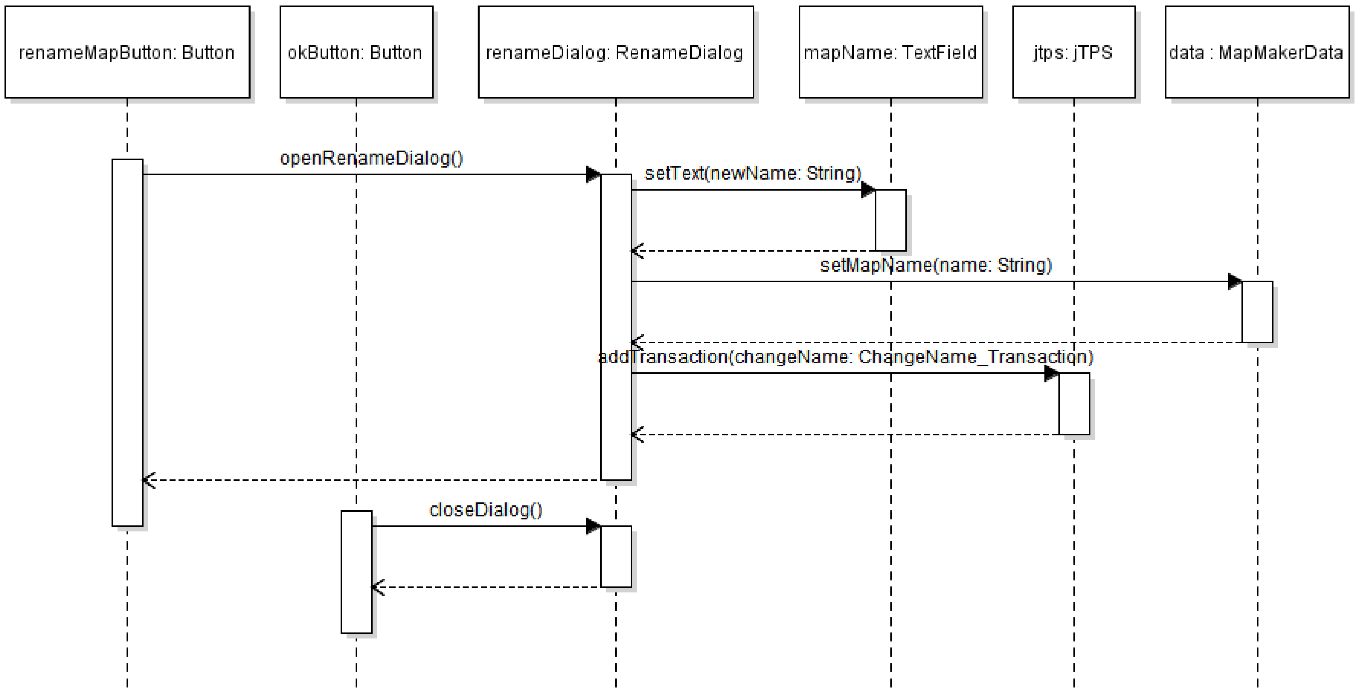

Figure 4.10: Change Map Name UML Sequence Diagrams

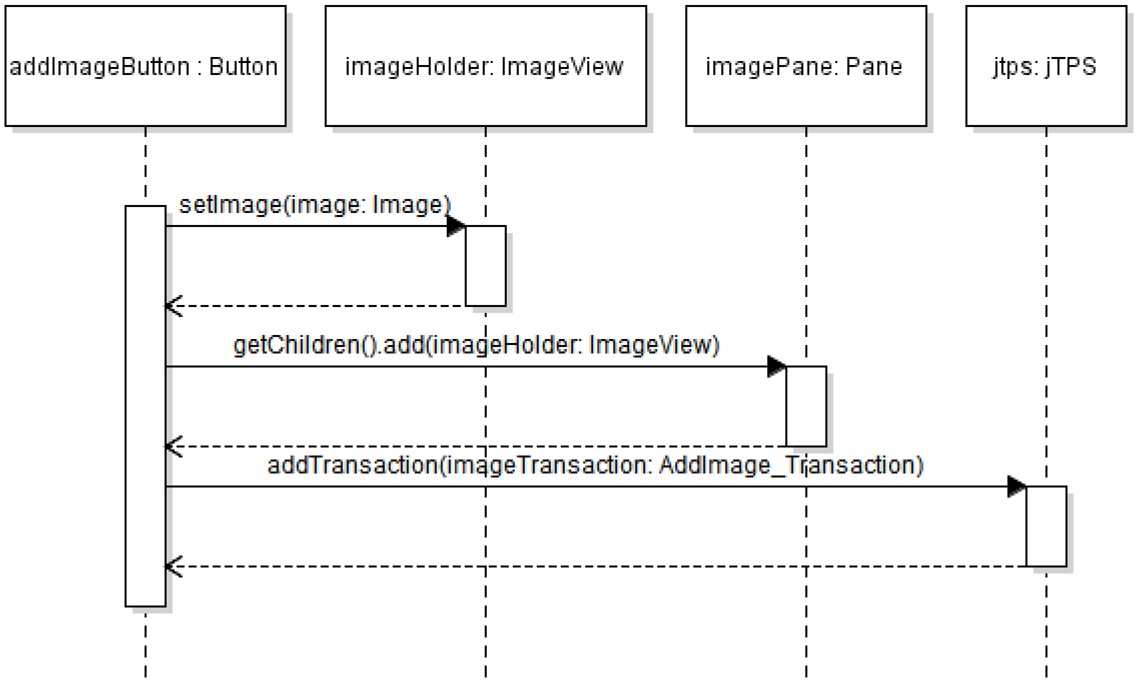
addImageButton : Button
imageHolder: ImageView
imagePane: Pane
jtps: jTPS
setImage(image: Image)
getChildren().add(imageHolder: ImageView)
addTransaction(imageTransaction: AddImage_Transaction)

Figure 4.11: Add Image To Map UML Sequence Diagrams

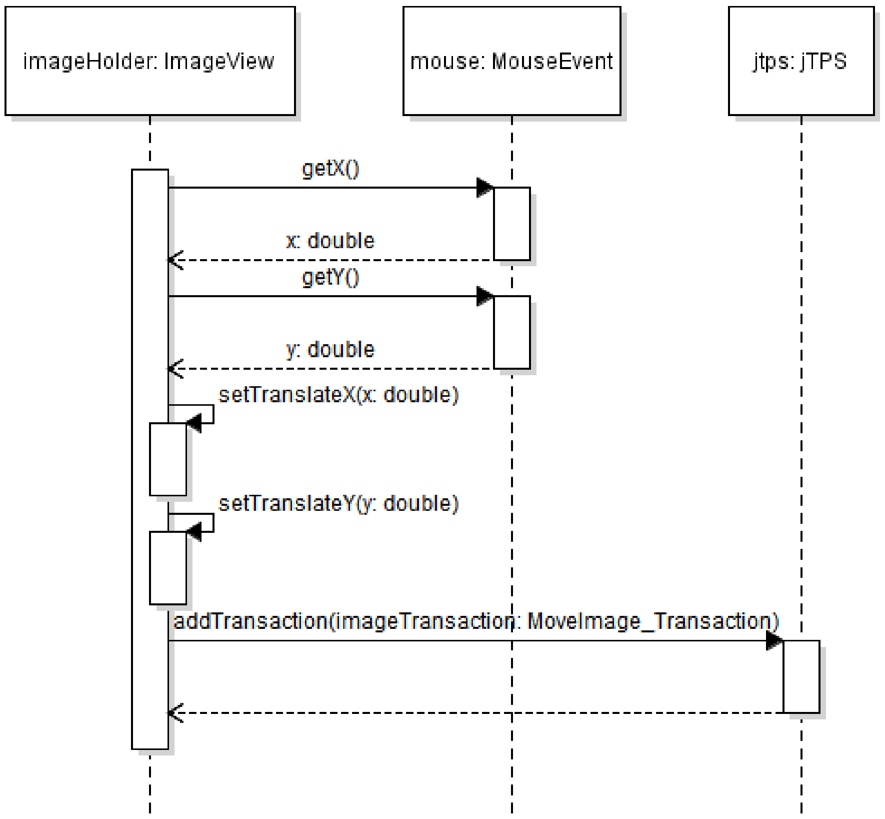

Figure 4.12: Move Image On Map UML Sequence Diagrams

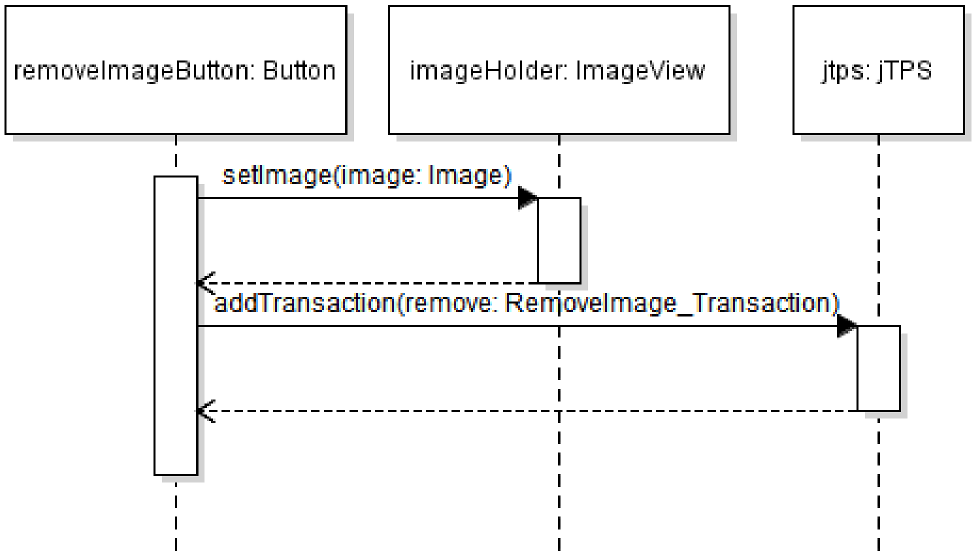
removeImageButton: Button
imageHolder: ImageView
jtps: jTPS
setImage(image: Image)
addTransaction(remove: RemoveImage_Transaction)

Figure 4.13: Remove Image From Map UML Sequence Diagrams

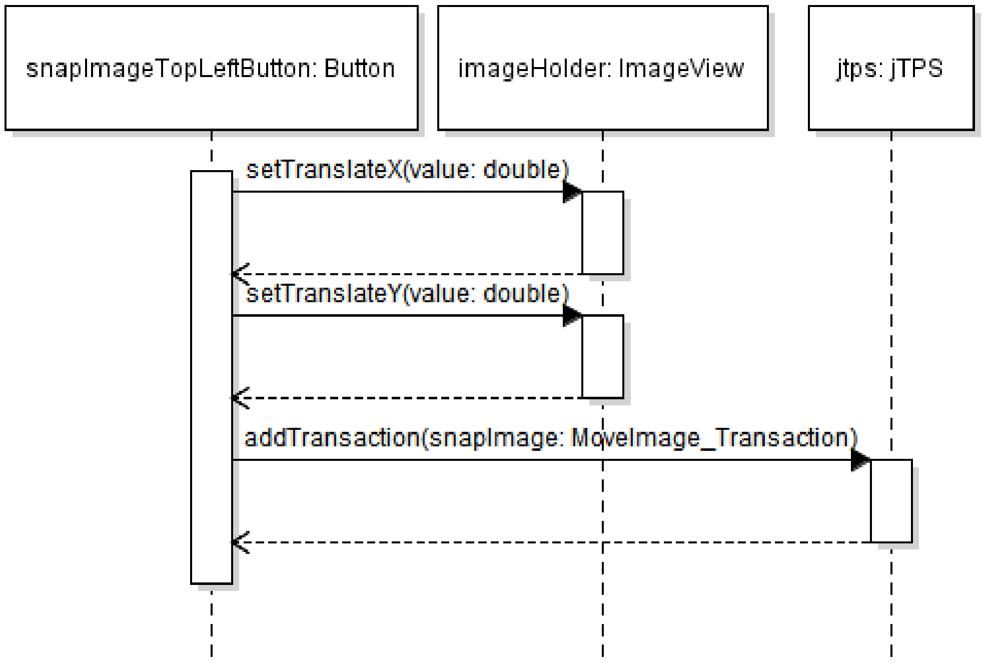
snapImageTopLeftButton: Button
imageHolder: ImageView
jtps: jTPS
setTranslateX(value: double)
setTranslateY(value: double)
addTransaction(snapImage: MoveImage_Transaction)

Figure 4.14: Snap Image To Top Left Corner UML Sequence Diagrams

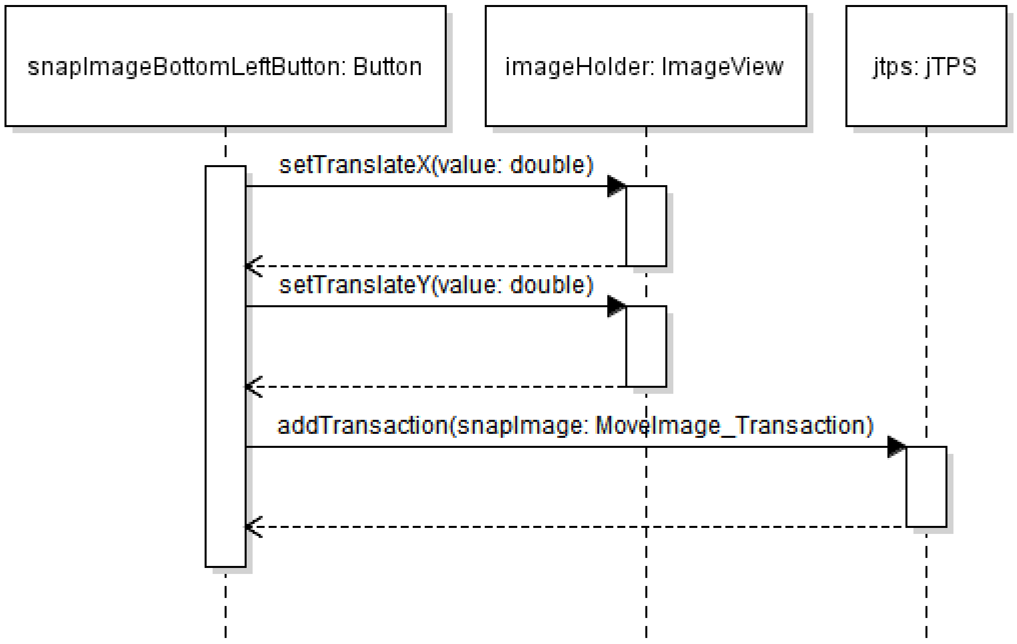
snapImageBottomLeftButton: Button
imageHolder: ImageView
jtps: jTPS
setTranslateX(value: double)
setTranslateY(value: double)
addTransaction(snapImage: MoveImage_Transaction)

Figure 4.15: Snap Image To Top Bottom Corner UML Sequence Diagrams

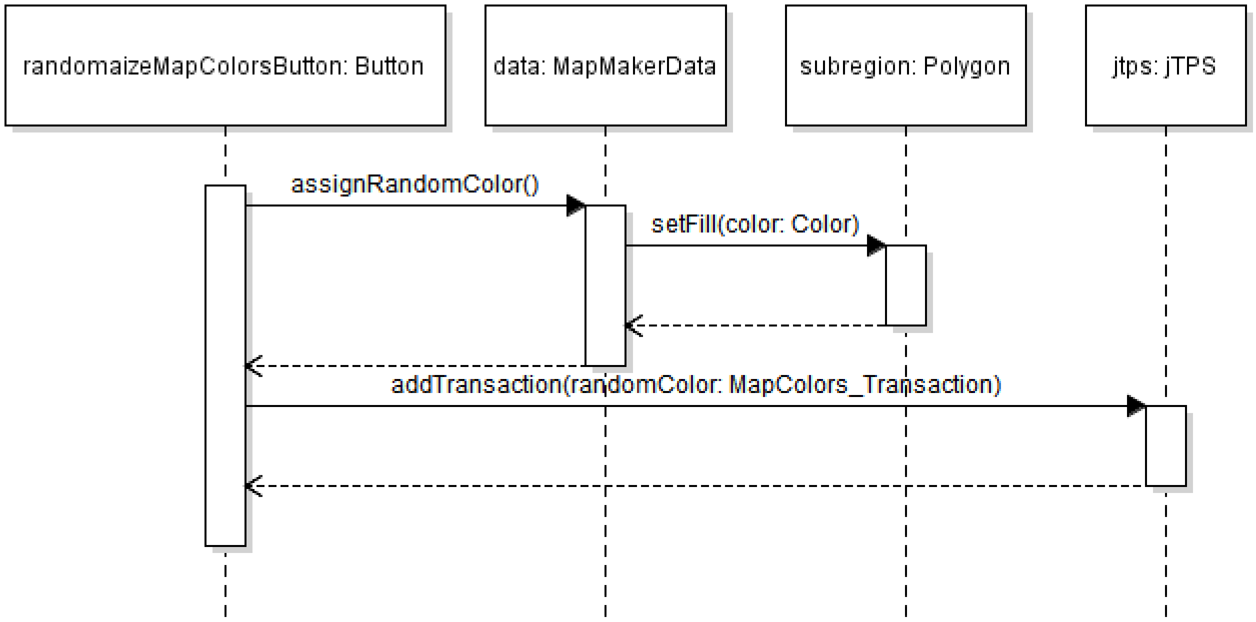

Figure 4.16: Reassign Map Colors UML Sequence Diagrams

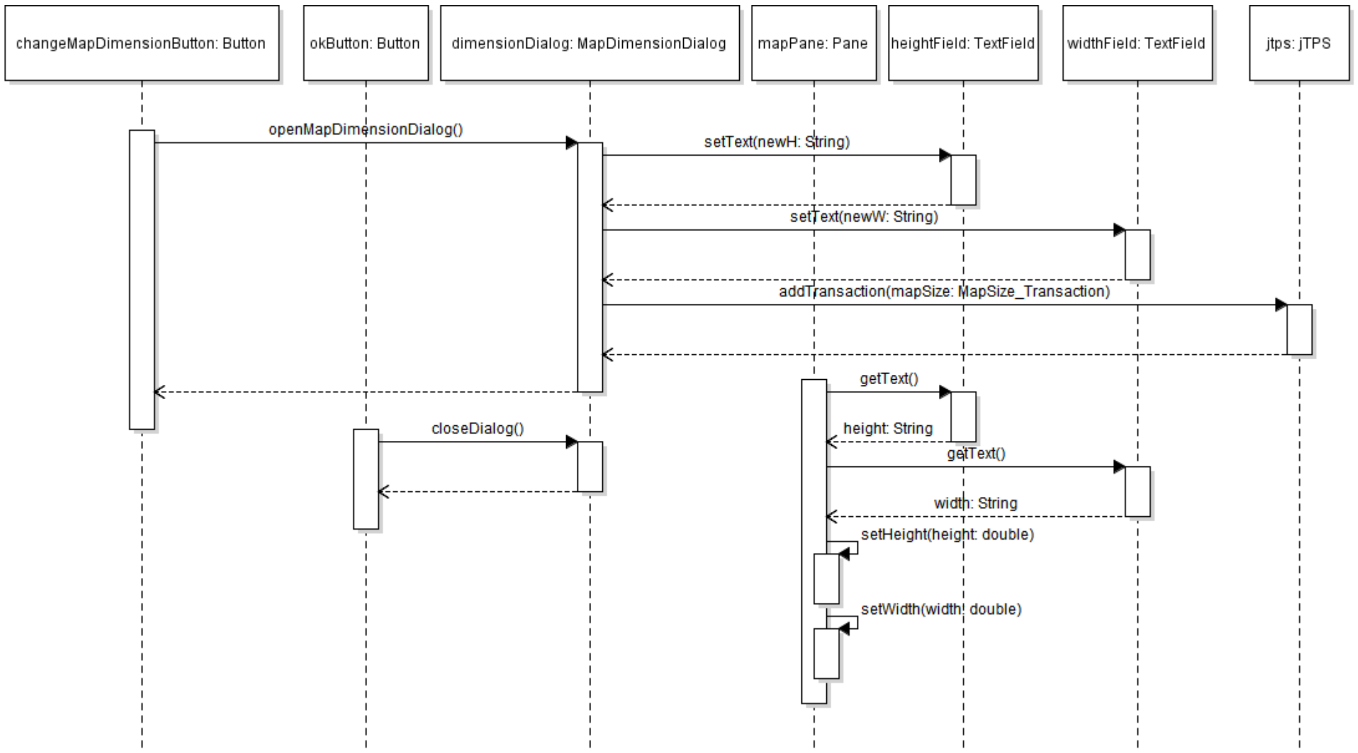

Figure 4.17: Change Map Size UML Sequence Diagrams

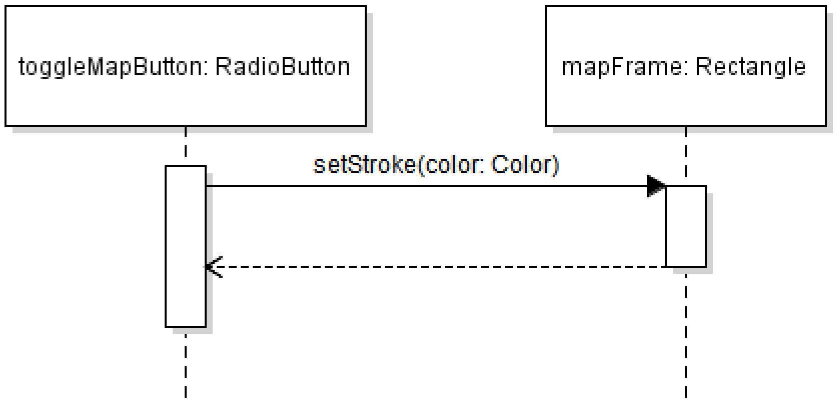

Figure 4.18: Toggle Map Frame UML Sequence Diagrams

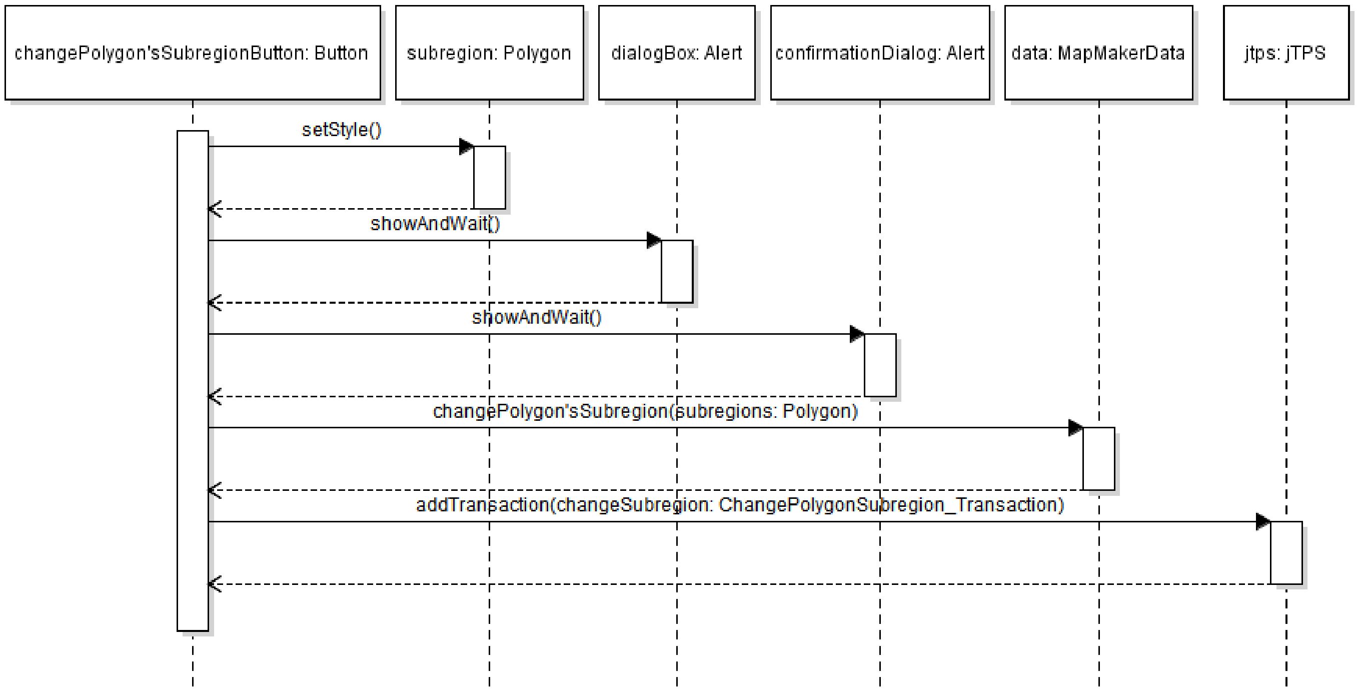
changePolygon'sSubregionButton: Button
subregion: Polygon
dialogBox: Alert
confirmationDialog: Alert
data: MapMakerData
jtps: jTPS
setStyle()
showAndWait()
showAndWait()
changePolygon'sSubregion(subregions: Polygon)
addTransaction(changeSubregion: ChangePolygonSubregion_Transaction)

Figure 4.19: Change Polygon's Subregion UML Sequence Diagrams

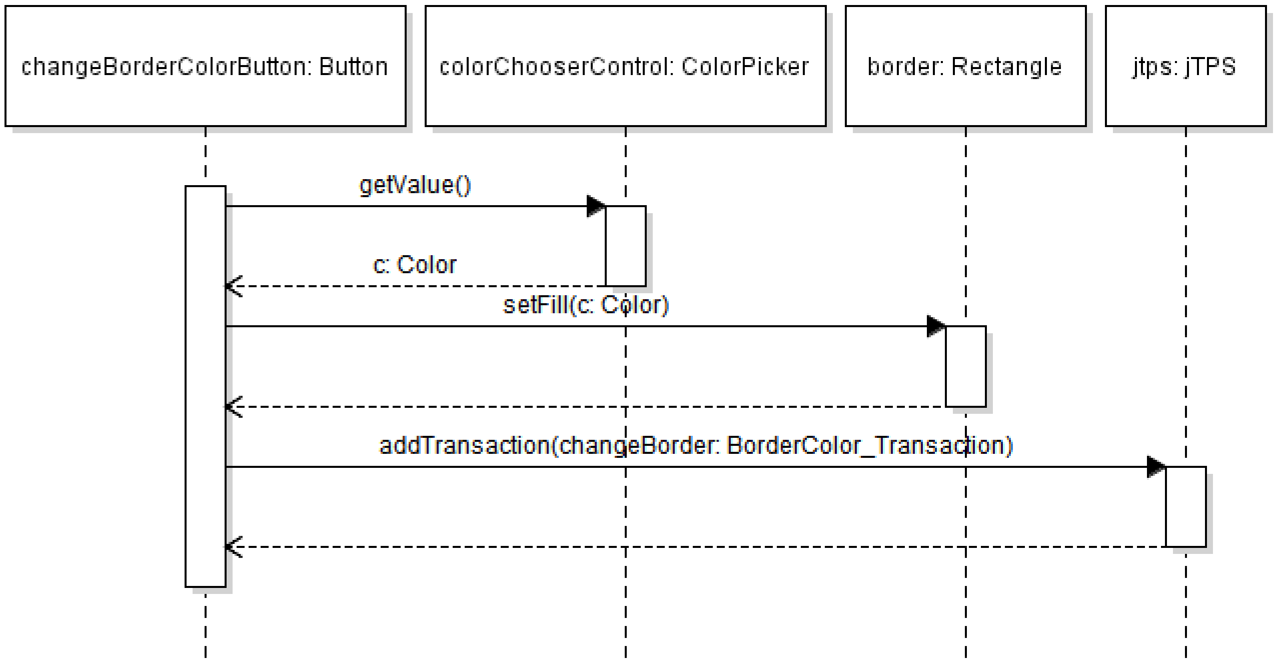
changeBorderColorButton: Button
colorChooserControl: ColorPicker
border: Rectangle
jtps: jTPS
getValue()
c: Color
setFill(c: Color)
addTransaction(changeBorder: BorderColor_Transaction)

Figure 4.20: Change Border Color UML Sequence Diagrams

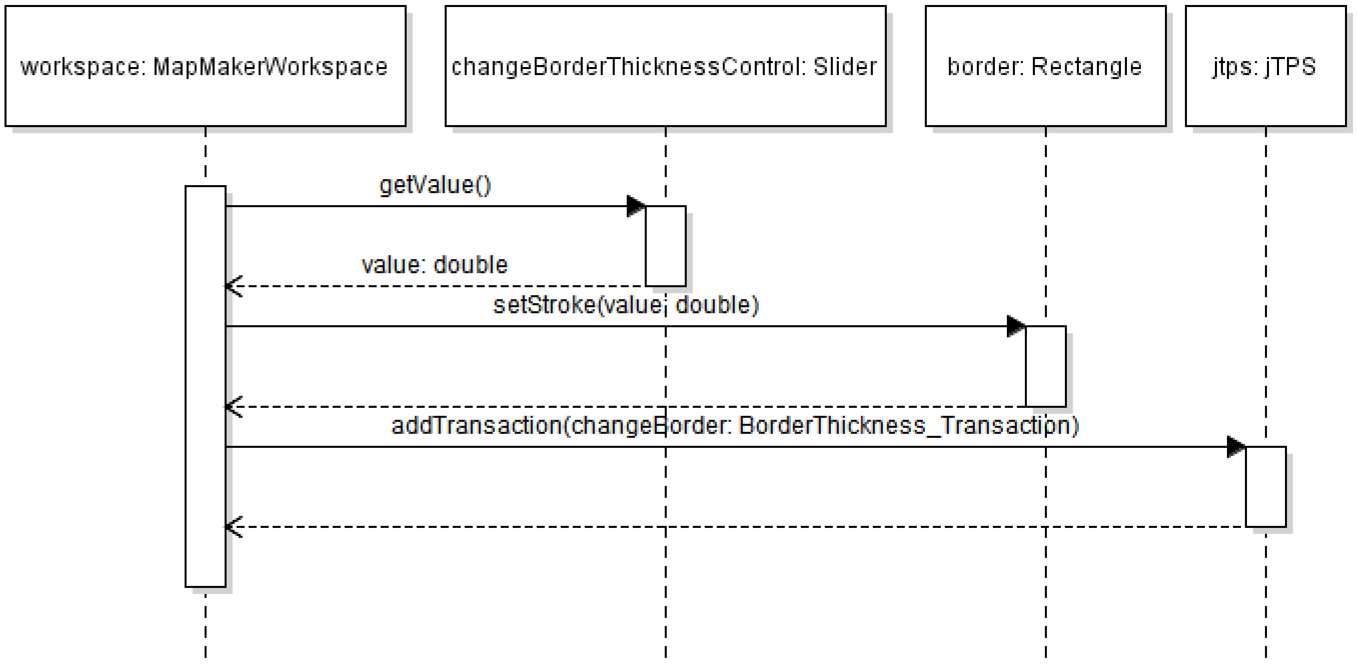
workspace: MapMakerWorkspace
changeBorderThicknessControl: Slider
border: Rectangle
jtps: jTPS
getValue()
value: double
setStroke(value: double)
addTransaction(changeBorder: BorderThickness_Transaction)

Figure 4.21: Change Border Thickness UML Sequence Diagrams

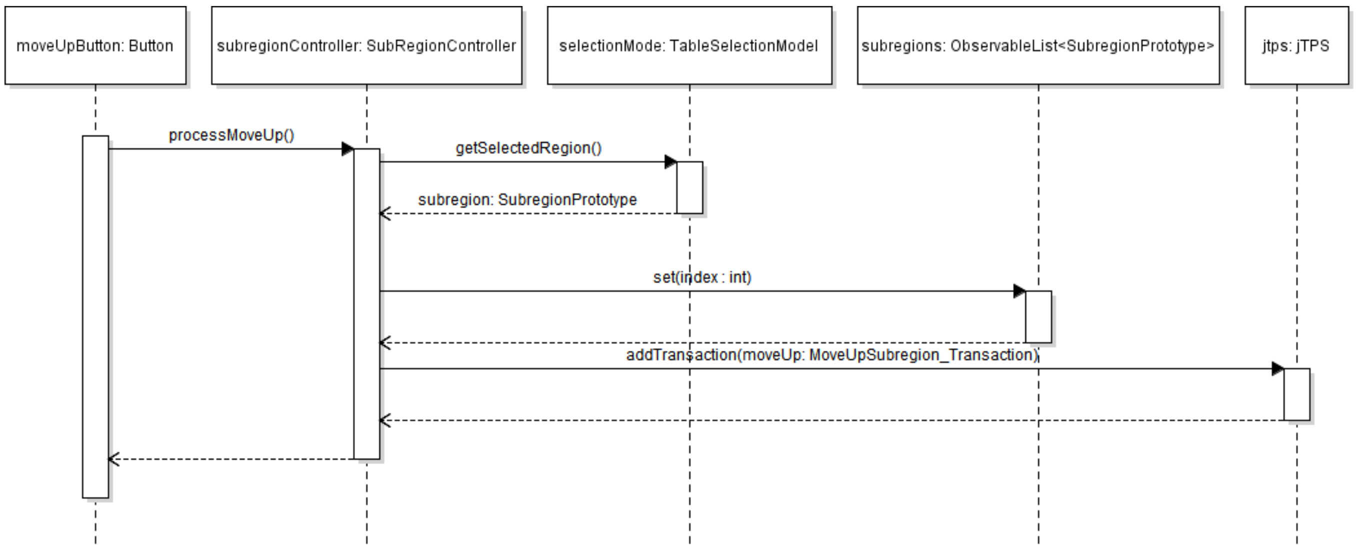
moveUpButton: Button
subregionController: SubRegionController
selectionMode: TableSelectionModel
subregions: ObservableList<SubregionPrototype>
jtps: jTPS
processMoveUp()
getSelectedRegion()
subregion: SubregionPrototype
set(index : int)
addTransaction(moveUp: MoveUpSubregion_Transaction)

Figure 4.22: Move Subregion Up UML Sequence Diagrams

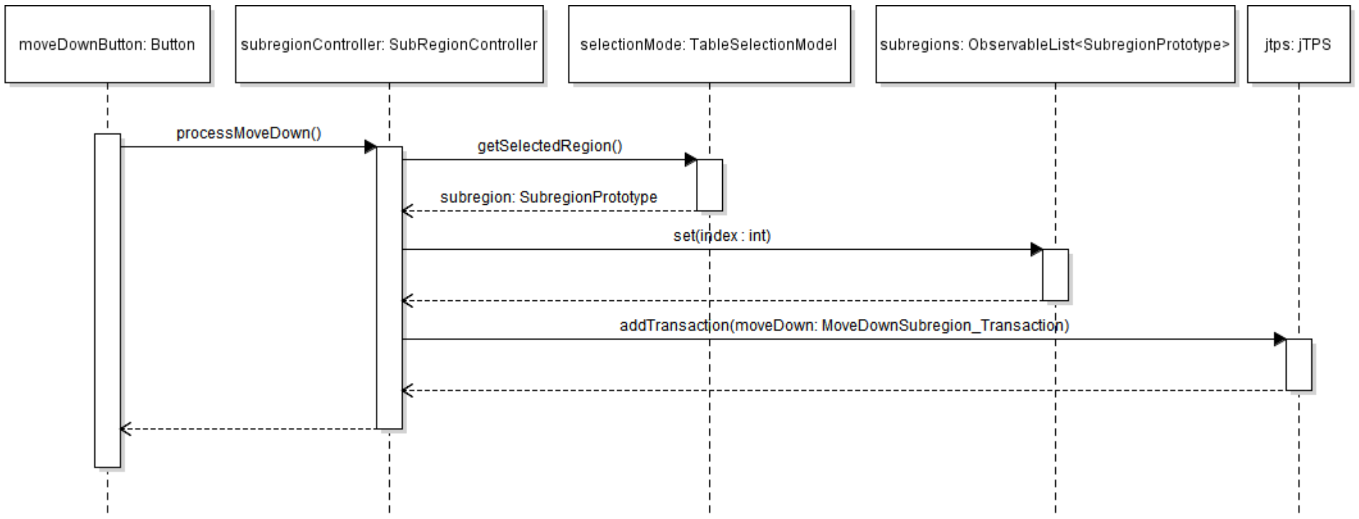
moveDownButton: Button
subregionController: SubRegionController
selectionMode: TableSelectionModel
subregions: ObservableList<SubregionPrototype>
jtps: jTPS
processMoveDown()
getSelectedRegion()
subregion: SubregionPrototype
set(index : int)
addTransaction(moveDown: MoveDownSubregion_Transaction)

Figure 4.23: Move Subregion Down UML Sequence Diagrams

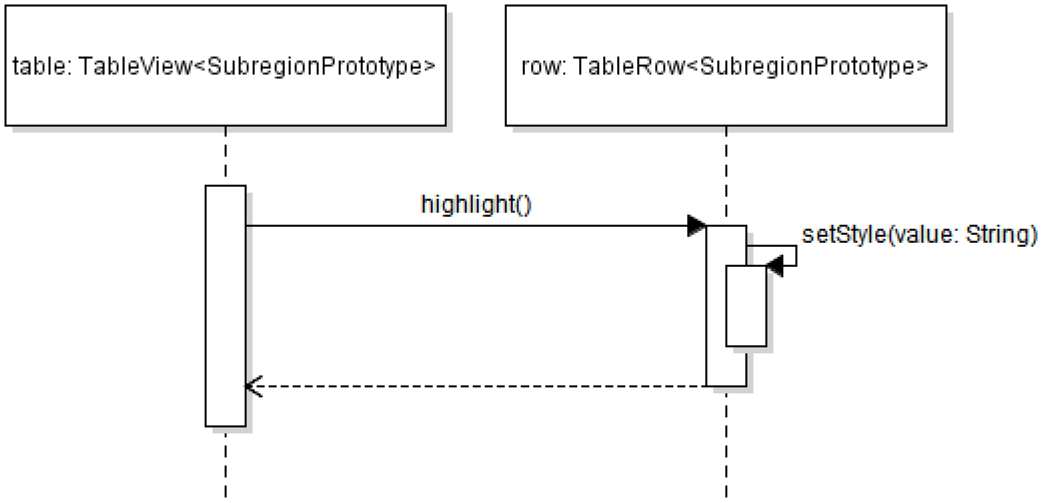
table: TableView<SubregionPrototype>
row: TableRow<SubregionPrototype>
highlight()
setStyle(value: String)

Figure 4.24: Select Map Subregion UML Sequence Diagrams

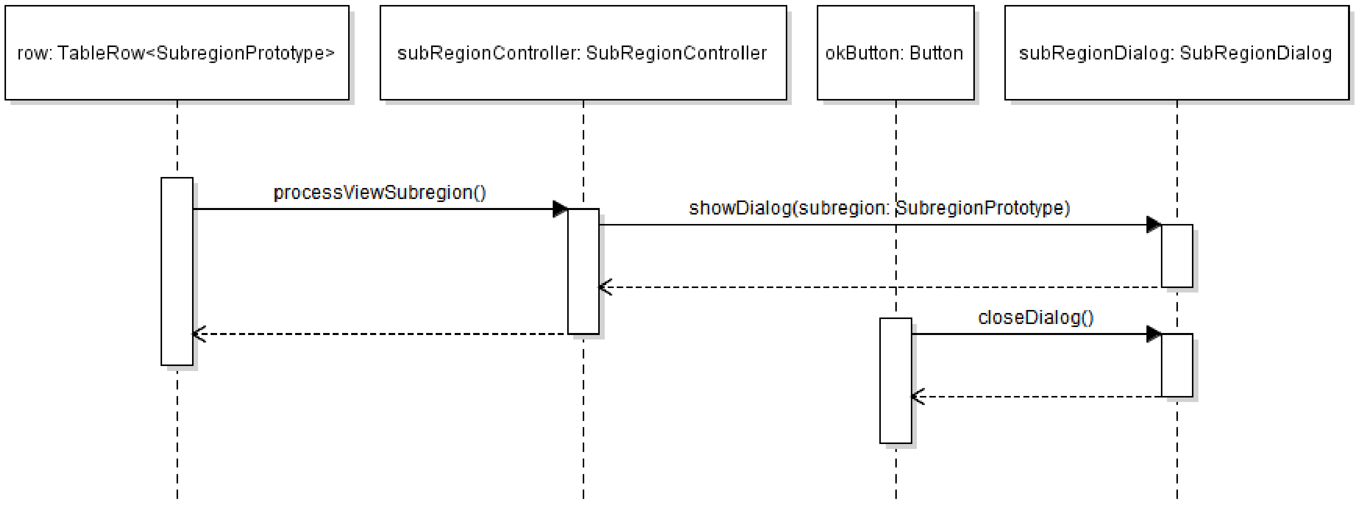
row: TableRow<SubregionPrototype>
subRegionController: SubRegionController
okButton: Button
subRegionDialog: SubRegionDialog
processViewSubregion()
showDialog(subregion: SubregionPrototype)
closeDialog()

Figure 4.25: View and Close Subregion Details UML Sequence Diagrams

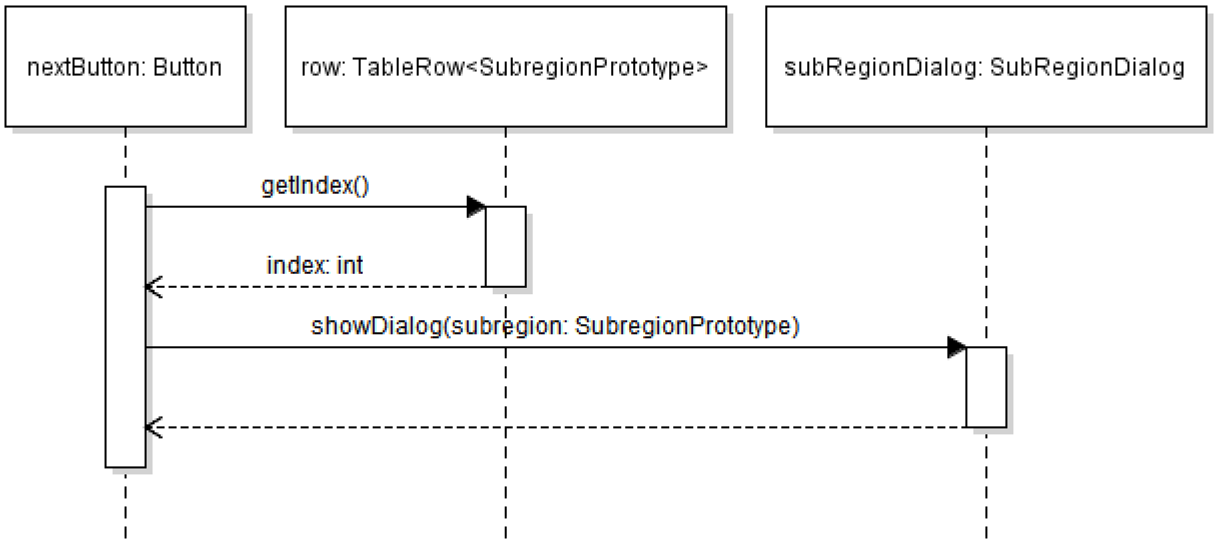
nextButton: Button
row: TableRow<SubregionPrototype>
subRegionDialog: SubRegionDialog
getIndex()
index: int
showDialog(subregion: SubregionPrototype)

Figure 4.26: Go to Next Sub Region UML Sequence Diagrams

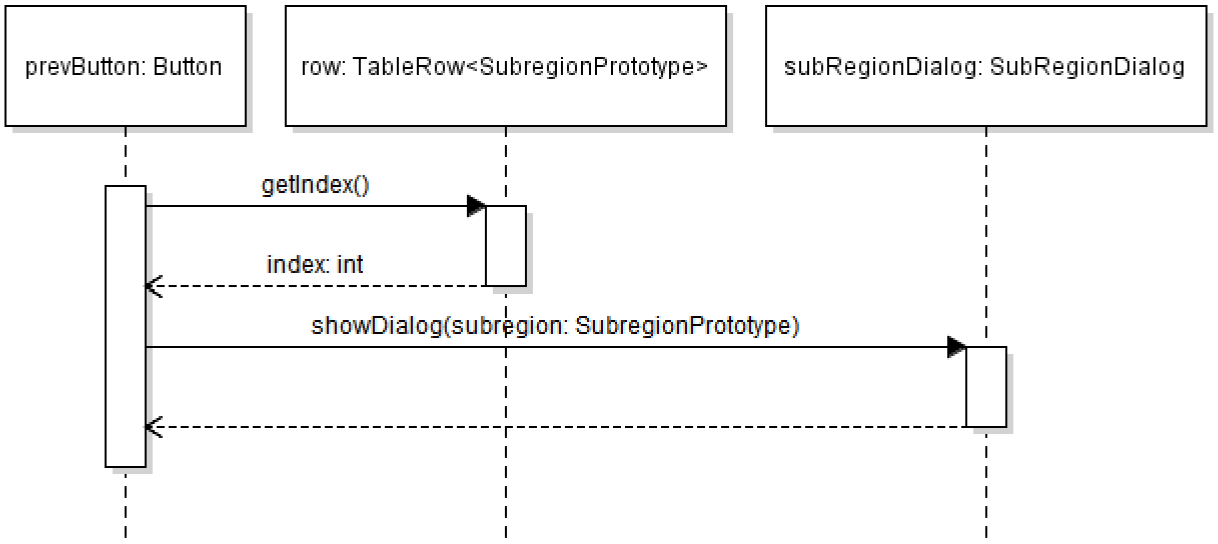
prevButton: Button
row: TableRow<SubregionPrototype>
subRegionDialog: SubRegionDialog
getIndex()
index: int
showDialog(subregion: SubregionPrototype)

Figure 4.27: Go to Previous Sub Region UML Sequence Diagrams

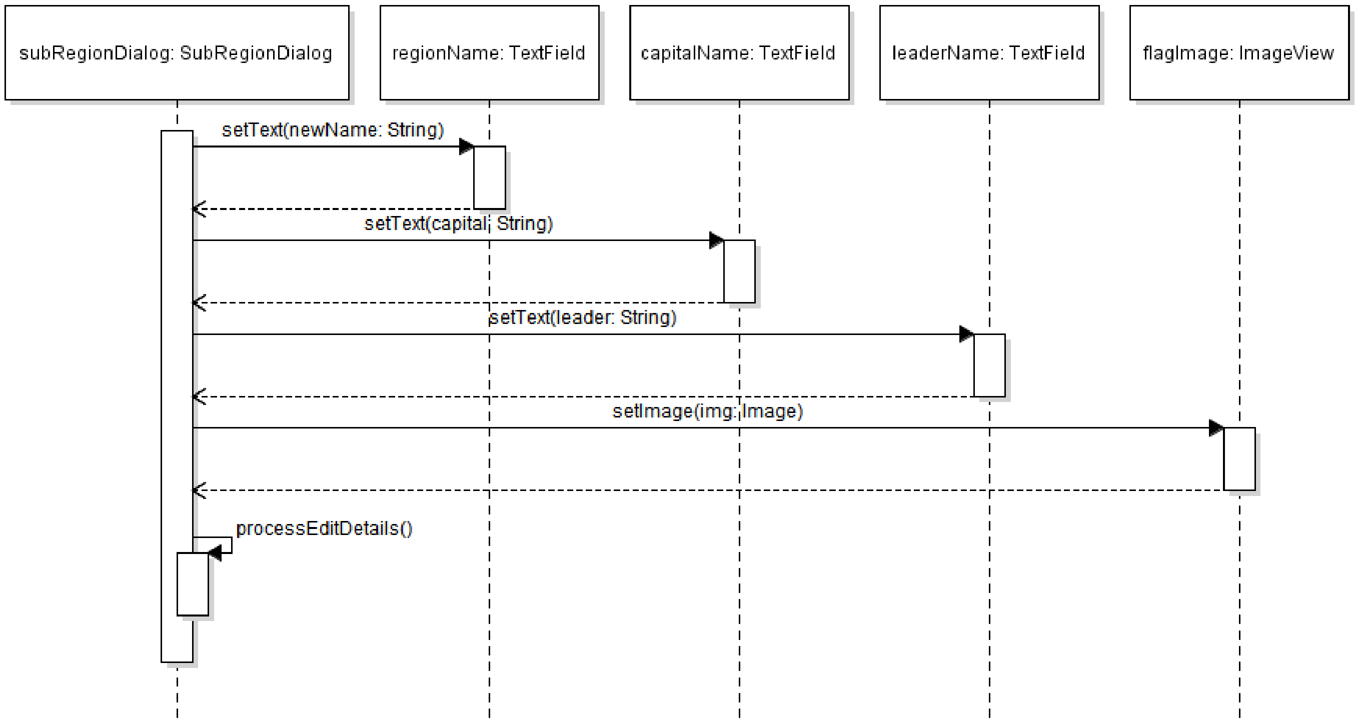

Figure 4.28: Edit Sub Region Details UML Sequence Diagrams

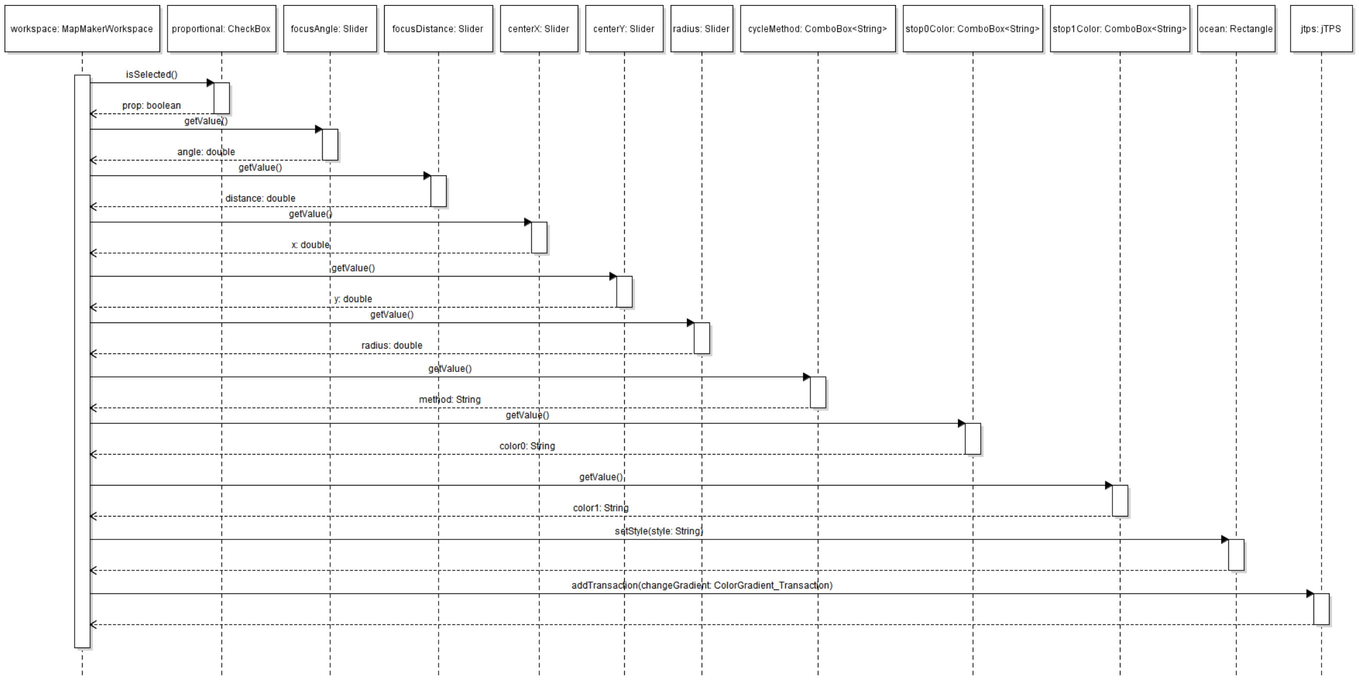

Figure 4.29: Change Background Color Gradient UML Sequence Diagrams

5 File Structure and Formats

Note that the Java Desktop Framework will be provided inside JavaDeskopFramework.jar, a Java ARchive file that will encapsulate the entire framework. This should be imported into the necessary project for the ***Regio Vinco*** Map Maker application and will be included in the deployment of a single, executable JAR file titled ***RegioVinco***MapMaker.jar. Note that all necessary data and art files must accompany this program. Figure 5.1 specifies the necessary file structure the launched application should use. Note that all necessary images should of course go in the image directory.

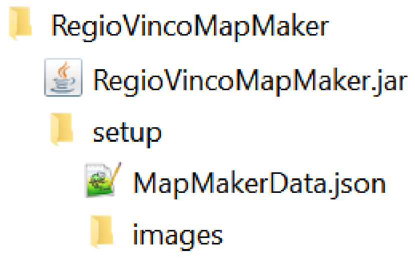

Figure 5.1: *RegioVinco*MapMaker File Structure

The MapMakerData.json provides the file and state names for all states. The file is a Json file that can be used to describe to MapMakerData as follows:

SCALE | TRANSLATE_X | TRANSLATE_Y | NUMBER_OF_SUBREGIONS |

SUBREGIONID | SUBREGIONS |

COLOR | GRADIENT | BORDERCOLOR | THICKNESS

We can describe these values as follows:

SCALE – A double that associates with the scale of the map.

TRANSLATE_X – A double that associates with the X coordinate of the map.

TRANSLATE_Y – A double that associates with the Y coordinate of the map.

NUMBER_OF_SUBREGIONS – An int that associates with the number of subregions in the map.

SUBREGIONID – An int that associates with the id of the sub region in the map.

SUBREGIONS – A hashmap of data that have coordinates that represents subregions.

COLOR – A double that associates with the color of the ocean.

GRADIENT – A double that associates with the gradient of the ocean.

BORDERCOLOR – A double that associates with the color of the border.

THICKNESS – A double that associates with the thickness of the border.

6 Supporting Information

Note that this document should serve as a reference for those implementing the code, so we'll provide a table of contents to help quickly find important sections.

6.1 Table of contents

6.2 **Appendixes**

N/A

Debugging Enterprises