The *Metro Map Maker*^{TM} Software Design Description

Author: Myungsuk Moon and Richard McKenna
Debugging Enterprises^{TM}
October 2017

Version 1.0

Abstract: This document describes the software design for the Metro Map Maker, a drawing tool for graphical representations of city subway systems.

Based on IEEE Std 1016^{TM}-2009 document format

1 Introduction

This is the Software Design Description (SDD) for the Metro Map Maker^{TM} *(i.e. M^3)* application. Note that this document format is based on the IEEE Standard 1016^{TM}-2009 recommendation for software design.

1.1 Purpose

This document is to serve as the blueprint for the construction of the Metro Map Maker application. This design will use UML class diagrams to provide complete detail regarding all packages, classes, instance variables, class variables, and method signatures needed to build the application. In addition, UML Sequence diagrams will be used to specify object interactions post-initialization of the application, meaning in response to user interactions or timed events.

1.2 Scope

The Metro Map Maker will be the application that lets the user draw and edit the subway map, and it should be easy to use for everyone. It will be built based on the framework called DesktopJavaFramework. So, this design contains design descriptions for the development of both the framework and the map maker. Note that Java is the target language for this software design.

1.3 Definitions, acronyms, and abbreviations

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.
Java – A high-level programming language that uses a virtual machine layer between the Java application and the hardware to provide program portability.

JavaScript – the default scripting language of the Web, JavaScript is provided to pages in the form of text files with code that can be loaded and executed when a page loads so as to dynamically generate page content in the DOM.

Stylesheet – a static text file employed by HTML pages that can control the colors, fonts, layout and other style components in a Web page.

Sequence Diagram – A UML document format that specifies how object methods interact with one another.

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

Use Case Descriptions – A formal format for specifying how a user will interact with a system.

1.4 References

IEEE Std 1016^{TM}-2009 – IEEE Standard for Information Technology – Systems Design – Software Design Descriptions

1.5 Overview

This Software Design Description document provides a working design for the Metro Map Maker software application as described in the Metro 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 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

As mentioned, this design will encompass both the Metro Map Maker application and its framework to be used in its construction. In building both we will heavily rely on the Java API to provide services. Following are descriptions of the components to be built, as well as how the Java API will be used to build them.

2.1 MetroMapMaker Overview

Figure 2.1 specifies all the components to be developed and places all classes in home packages.

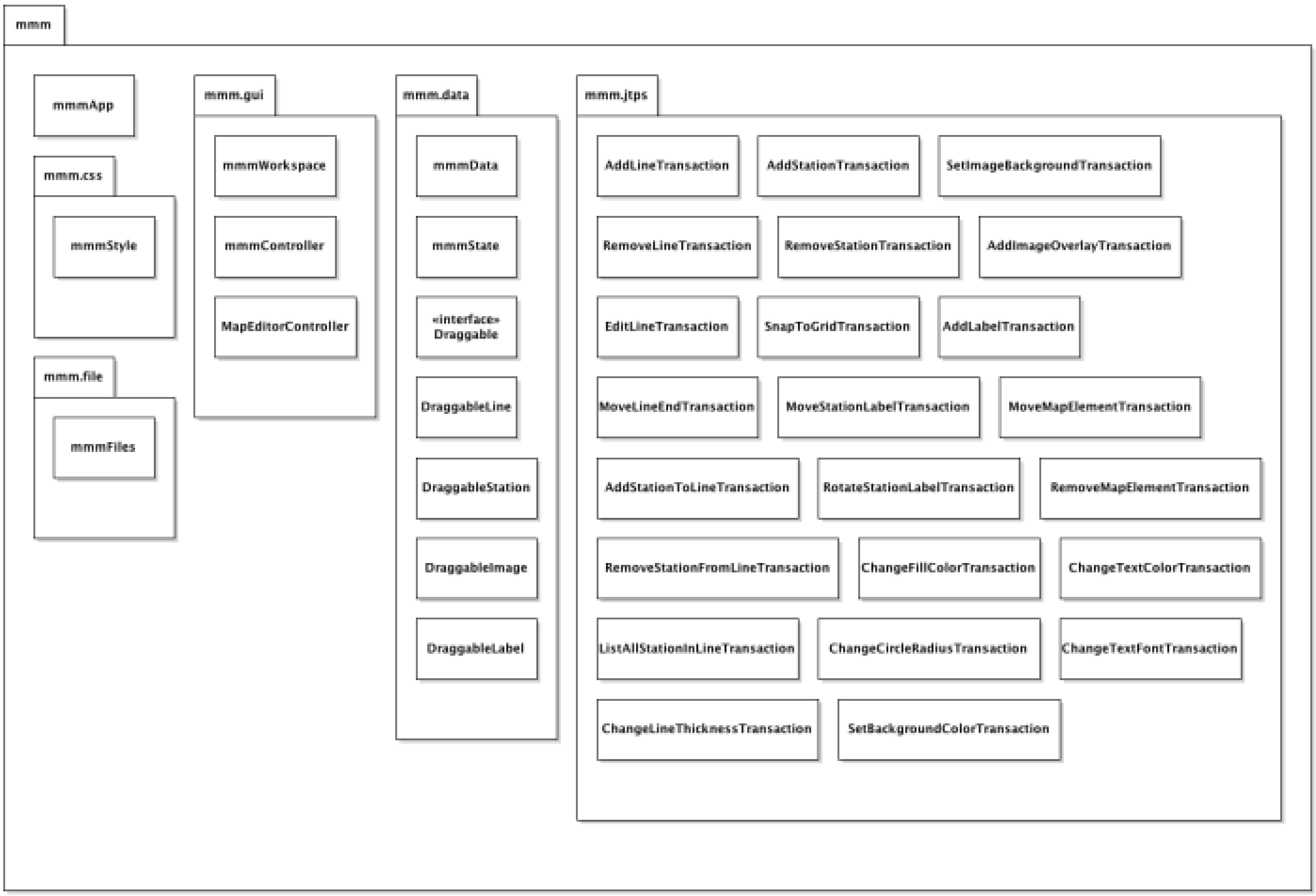

mmm
mmmApp
mmm.css
mmmStyle
mmm.file
mmmFiles
mmm.gui
mmmWorkspace
mmmController
MapEditorController
mmm.data
mmmData
mmmState
«interface» Draggable
DraggableLine
DraggableStation
DraggableImage
DraggableLabel
mmm.jtps
AddLineTransaction
AddStationTransaction
SetImageBackgroundTransaction
RemoveLineTransaction
RemoveStationTransaction
AddImageOverlayTransaction
EditLineTransaction
SnapToGridTransaction
AddLabelTransaction
MoveLineEndTransaction
MoveStationLabelTransaction
MoveMapElementTransaction
AddStationToLineTransaction
RotateStationLabelTransaction
RemoveMapElementTransaction
RemoveStationFromLineTransaction
ChangeFillColorTransaction
ChangeTextColorTransaction
ListAllStationInLineTransaction
ChangeCircleRadiusTransaction
ChangeTextFontTransaction
ChangeLineThicknessTransaction
SetBackgroundColorTransaction

Figure 2.1: Design Packages Overview

2.2 Java API Usage

The MetroMapMaker will be developed using the Java programming languages. As such, this design will make use of the classes specified in Figure 2.2.

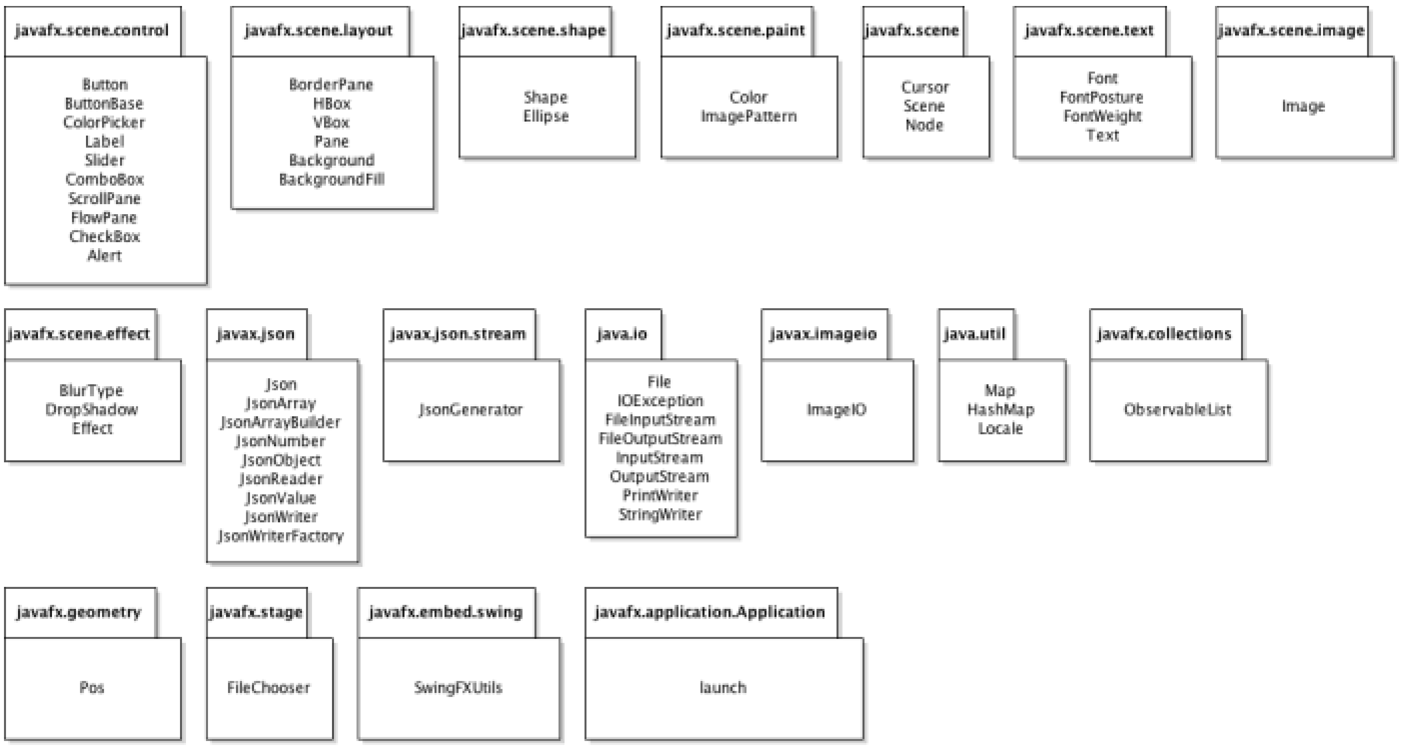


Figure 2.2: Java API Classes and Packages To Be Used

2.3 Java API Usage Descriptions

Tables 2.1-2.18 below summarize how each of these classes will be used.

Class/Interface	Use
Button	For adding buttons in the user interface
ButtonBase	For helping to add buttons and its controls
ColorPicker	For adding a control that lets the user pick a color
Label	For labeling each left control rows in the user interface
Slider	For adding a control that lets the user adjust values such as station radius
ComboBox	For adding a control that lets the user choose through options
ScrollPane	For making certain pane scrollable vertically
FlowPane	For adding buttons or user controls in this pane
CheckBox	For adding box that is checkable to either true or false
Alert	For creating the pop-up dialogue for user interactions

Table 2.1: Uses for classes in the Java API's javafx.scene.control package

Class/Interface	Use
BorderPane	For adding toolbars top, left and center then adding this BorderPane to Pane
HBox	For adding buttons or user controls horizontally
VBox	For adding buttons or user controls vertically
Pane	For creating the workspace that can then filled with toolbars
Background	For setting the background color for the map editor
BackgroundFill	For filling the background that is used for the map editor

Table 2.2: Uses for classes in the Java API's javafx.scene.layout package

Class/Interface	Use
Shape	For adding nodes such as station, line, image or label
Ellipse	For circles that is used for stations

Table 2.3: Uses for classes in the Java API's javafx.scene.Shape package

Class/Interface	Use
Color	For setting the rendering colors for text and the progress bar
ImagePattern	For filling this pattern with images then adding the filled pattern to shape

Table 2.4: Uses for classes in the Java API's java.scene.paint package

Class/Interface	Use
Cursor	For getting the user's horizontal and vertical cursor position
Scene	For creating the workspace then filling this in the stage
Node	For shapes(station, line, image and label) and canvas controls

Table 2.5: Uses for classes in the Java API's java.scene package

Class/Interface	Use
Font	For setting the fonts for rendered text
FontPosture	For setting the fonts italic
FontWeight	For setting the font bold
Text	For text labels in the drawing

Table 2.6: Uses for classes in the Java API's java.scene.text package

Class/Interface	Use
Image	For adding images to the drawing

Table 2.7: Uses for classes in the Java API's java.scene.image package

Class/Interface	Use
BlurType	For setting the type of blur used when highlighting the selected node
DropShadow	For highlighting the selected node
Effect	For storing the highlighted node

Table 2.8: Uses for classes in the Java API's java.scene.effect package

Class/Interface	Use
Json	For using the JSON formatted file
JsonArray	For representing an immutable JSON array value
JsonArrayBuilder	For creating JsonArray models from scratch
JsonNumber	For representing an immutable JSON number value
JsonObject	For representing an immutable JSON object value
JsonReader	For reading JSON object
JsonValue	For representing an immutable JSON value
JsonWriter	For writing a JSON object to an output source
JsonWriterFactory	For creating JsonWriter instances

Table 2.9: Uses for classes in the Java API's javax.json package

Class/Interface	Use
JsonGenerator	For writing JSON data to an output source

Table 2.10: Uses for classes in the Java API's javax.json.stream package

Class/Interface	Use
File	For representing file and directory pathnames
IOException	For letting the user know some type of input or output exception has occured
FileInputStream	For obtaining the input byte from file
FileOutputStream	For writing the data to a File
InputStream	For representing an input stream of bytes
OutputStream	For representing an output stream of bytes
PrintWriter	For printing formatted representations of objects to a text-output stream
StringWriter	For collecting the output in a string buffer

Table 2.11: Uses for classes in the Java API's java.io package

Class/Interface	Use
ImageIO	For locating image related classes and performing encoding and decoding

Table 2.12: Uses for classes in the Java API's java.imageio package

Class/Interface	Use
Map	For mapping keys to value
HashMap	For implementation of map with hash based
Locale	For representing a specific geographical, political, or cultural region

Table 2.13: Uses for classes in the Java API's java.util package

Class/Interface	Use
ObservableList	For representing an observable object

Table 2.14: Uses for classes in the Java API's javafx.collections package

Class/Interface	Use
Pos	For describing vertical and horizontal positioning and alignment

Table 2.15: Uses for classes in the Java API's javafx.geometry package

Class/Interface	Use
FileChooser	For providing support for standard platform file dialogs

Table 2.16: Uses for classes in the Java API's javafx.stage package

Class/Interface	Use
SwingFXUtils	For providing utility methods for converting data types between Swing/AWT and JavaFX formats.

Table 2.17: Uses for classes in the Java API's java.embed.swing package

Class/Interface	Use
launch	For launching the application

Table 2.18: Uses for classes in the Java API's java.application.Application package

3 Class-Level Design Viewpoint

As mentioned, this design will encompass both the Metro Map Maker application its framework. The following UML Class Diagrams reflect this. Note that due to the complexity of the project, we present the class designs using a series of diagrams going from overview diagrams down to detailed ones.

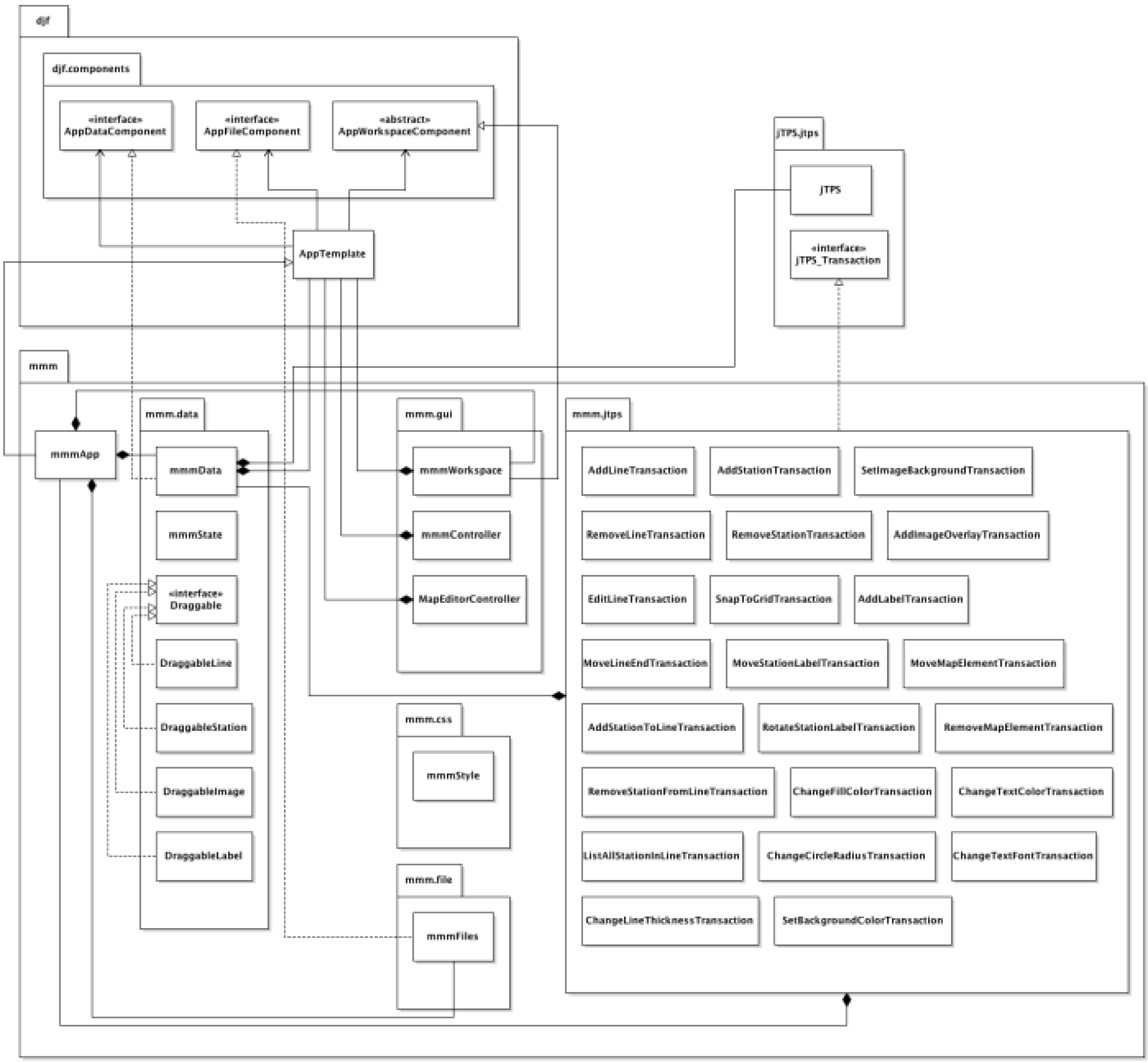
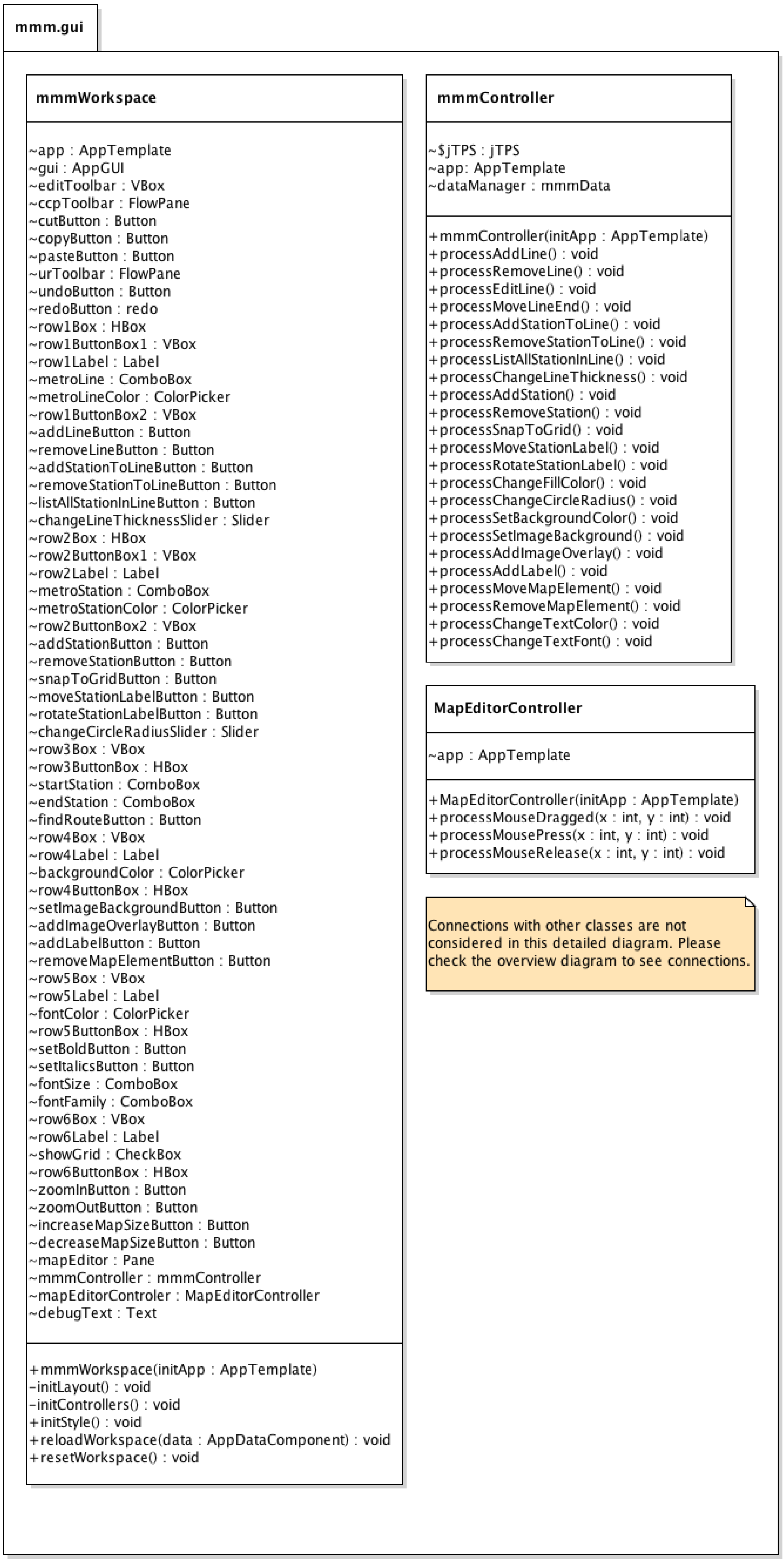

djf
djf.components
«interface» AppDataComponent
«interface» AppFileComponent
«abstract» AppWorkspaceComponent
AppTemplate
jTPS.jtps
jTPS
«interface» jTPS_Transaction
mmm
mmmApp
mmm.data
mmmData
mmmState
«interface» Draggable
DraggableLine
DraggableStation
DraggableImage
DraggableLabel
mmm.gui
mmmWorkspace
mmmController
MapEditorController
mmm.css
mmmStyle
mmm.file
mmmFiles
mmm.jtps
AddLineTransaction
AddStationTransaction
SetImageBackgroundTransaction
RemoveLineTransaction
RemoveStationTransaction
AddImageOverlayTransaction
EditLineTransaction
SnapToGridTransaction
AddLabelTransaction
MoveLineEndTransaction
MoveStationLabelTransaction
MoveMapElementTransaction
AddStationToLineTransaction
RotateStationLabelTransaction
RemoveMapElementTransaction
RemoveStationFromLineTransaction
ChangeFillColorTransaction
ChangeTextColorTransaction
ListAllStationInLineTransaction
ChangeCircleRadiusTransaction
ChangeTextFontTransaction
ChangeLineThicknessTransaction
SetBackgroundColorTransaction

Figure 3.1: MetroMapMaker Overview UML Class Diagram



```
mmm.gui

mmmWorkspace
--------------------------------
~app : AppTemplate
~gui : AppGUI
~editToolbar : VBox
~ccpToolbar : FlowPane
~cutButton : Button
~copyButton : Button
~pasteButton : Button
~urToolbar : FlowPane
~undoButton : Button
~redoButton : redo
~row1Box : HBox
~row1ButtonBox1 : VBox
~row1Label : Label
~metroLine : ComboBox
~metroLineColor : ColorPicker
~row1ButtonBox2 : VBox
~addLineButton : Button
~removeLineButton : Button
~addStationToLineButton : Button
~removeStationToLineButton : Button
~listAllStationInLineButton : Button
~changeLineThicknessSlider : Slider
~row2Box : HBox
~row2ButtonBox1 : VBox
~row2Label : Label
~metroStation : ComboBox
~metroStationColor : ColorPicker
~row2ButtonBox2 : VBox
~addStationButton : Button
~removeStationButton : Button
~snapToGridButton : Button
~moveStationLabelButton : Button
~rotateStationLabelButton : Button
~changeCircleRadiusSlider : Slider
~row3Box : VBox
~row3ButtonBox : HBox
~startStation : ComboBox
~endStation : ComboBox
~findRouteButton : Button
~row4Box : VBox
~row4Label : Label
~backgroundColor : ColorPicker
~row4ButtonBox : HBox
~setImageBackgroundButton : Button
~addImageOverlayButton : Button
~addLabelButton : Button
~removeMapElementButton : Button
~row5Box : VBox
~row5Label : Label
~fontColor : ColorPicker
~row5ButtonBox : HBox
~setBoldButton : Button
~setItalicsButton : Button
~fontSize : ComboBox
~fontFamily : ComboBox
~row6Box : VBox
~row6Label : Label
~showGrid : CheckBox
~row6ButtonBox : HBox
~zoomInButton : Button
~zoomOutButton : Button
~increaseMapSizeButton : Button
~decreaseMapSizeButton : Button
~mapEditor : Pane
~mmmController : mmmController
~mapEditorControler : MapEditorController
~debugText : Text
--------------------------------
+mmmWorkspace(initApp : AppTemplate)
-initLayout() : void
-initControllers() : void
+initStyle() : void
+reloadWorkspace(data : AppDataComponent) : void
+resetWorkspace() : void

mmmController
--------------------------------
~$jTPS : jTPS
~app: AppTemplate
~dataManager : mmmData
--------------------------------
+mmmController(initApp : AppTemplate)
+processAddLine() : void
+processRemoveLine() : void
+processEditLine() : void
+processMoveLineEnd() : void
+processAddStationToLine() : void
+processRemoveStationToLine() : void
+processListAllStationInLine() : void
+processChangeLineThickness() : void
+processAddStation() : void
+processRemoveStation() : void
+processSnapToGrid() : void
+processMoveStationLabel() : void
+processRotateStationLabel() : void
+processChangeFillColor() : void
+processChangeCircleRadius() : void
+processSetBackgroundColor() : void
+processSetImageBackground() : void
+processAddImageOverlay() : void
+processAddLabel() : void
+processMoveMapElement() : void
+processRemoveMapElement() : void
+processChangeTextColor() : void
+processChangeTextFont() : void

MapEditorController
--------------------------------
~app : AppTemplate
--------------------------------
+MapEditorController(initApp : AppTemplate)
+processMouseDragged(x : int, y : int) : void
+processMousePress(x : int, y : int) : void
+processMouseRelease(x : int, y : int) : void
```

Connections with other classes are not considered in this detailed diagram. Please check the overview diagram to see connections.

Figure 3.2: Detailed mmm.gui UML Class Diagram

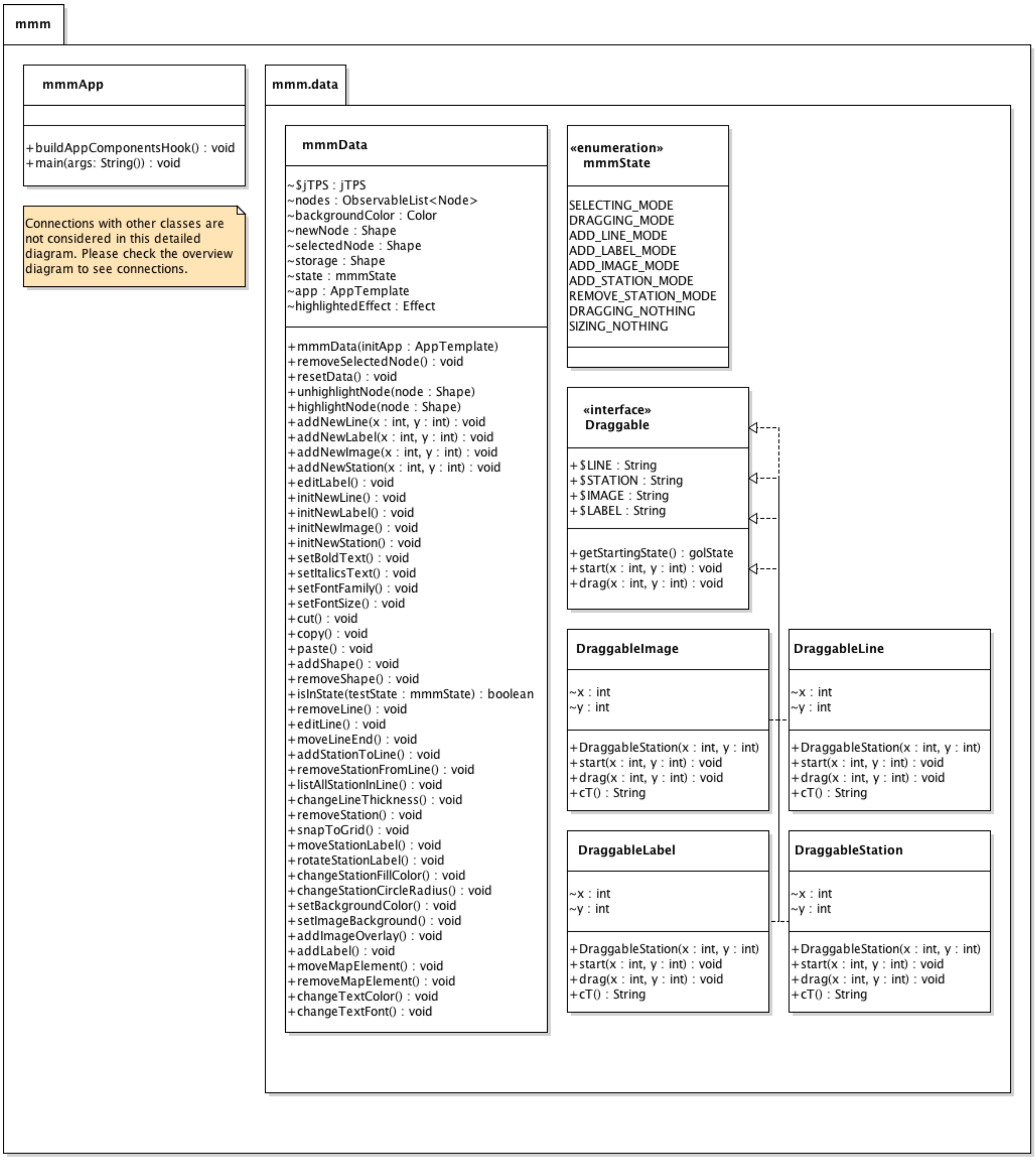

mmm
mmmApp
+buildAppComponentsHook() : void
+main(args: String()) : void
Connections with other classes are not considered in this detailed diagram. Please check the overview diagram to see connections.
mmm.data
mmmData
~$jTPS : jTPS
~nodes : ObservableList<Node>
~backgroundColor : Color
~newNode : Shape
~selectedNode : Shape
~storage : Shape
~state : mmmState
~app : AppTemplate
~highlightedEffect : Effect
+mmmData(initApp : AppTemplate)
+removeSelectedNode() : void
+resetData() : void
+unhighlightNode(node : Shape)
+highlightNode(node : Shape)
+addNewLine(x : int, y : int) : void
+addNewLabel(x : int, y : int) : void
+addNewImage(x : int, y : int) : void
+addNewStation(x : int, y : int) : void
+editLabel() : void
+initNewLine() : void
+initNewLabel() : void
+initNewImage() : void
+initNewStation() : void
+setBoldText() : void
+setItalicsText() : void
+setFontFamily() : void
+setFontSize() : void
+cut() : void
+copy() : void
+paste() : void
+addShape() : void
+removeShape() : void
+isInState(testState : mmmState) : boolean
+removeLine() : void
+editLine() : void
+moveLineEnd() : void
+addStationToLine() : void
+removeStationFromLine() : void
+listAllStationInLine() : void
+changeLineThickness() : void
+removeStation() : void
+snapToGrid() : void
+moveStationLabel() : void
+rotateStationLabel() : void
+changeStationFillColor() : void
+changeStationCircleRadius() : void
+setBackgroundColor() : void
+setImageBackground() : void
+addImageOverlay() : void
+addLabel() : void
+moveMapElement() : void
+removeMapElement() : void
+changeTextColor() : void
+changeTextFont() : void
«enumeration»
mmmState
SELECTING_MODE
DRAGGING_MODE
ADD_LINE_MODE
ADD_LABEL_MODE
ADD_IMAGE_MODE
ADD_STATION_MODE
REMOVE_STATION_MODE
DRAGGING_NOTHING
SIZING_NOTHING
«interface»
Draggable
+$LINE : String
+$STATION : String
+$IMAGE : String
+$LABEL : String
+getStartingState() : golState
+start(x : int, y : int) : void
+drag(x : int, y : int) : void
DraggableImage
~x : int
~y : int
+DraggableStation(x : int, y : int)
+start(x : int, y : int) : void
+drag(x : int, y : int) : void
+cT() : String
DraggableLine
~x : int
~y : int
+DraggableStation(x : int, y : int)
+start(x : int, y : int) : void
+drag(x : int, y : int) : void
+cT() : String
DraggableLabel
~x : int
~y : int
+DraggableStation(x : int, y : int)
+start(x : int, y : int) : void
+drag(x : int, y : int) : void
+cT() : String
DraggableStation
~x : int
~y : int
+DraggableStation(x : int, y : int)
+start(x : int, y : int) : void
+drag(x : int, y : int) : void
+cT() : String

Figure 3.3: Detailed mmmApp and mmm.data UML Class Diagram

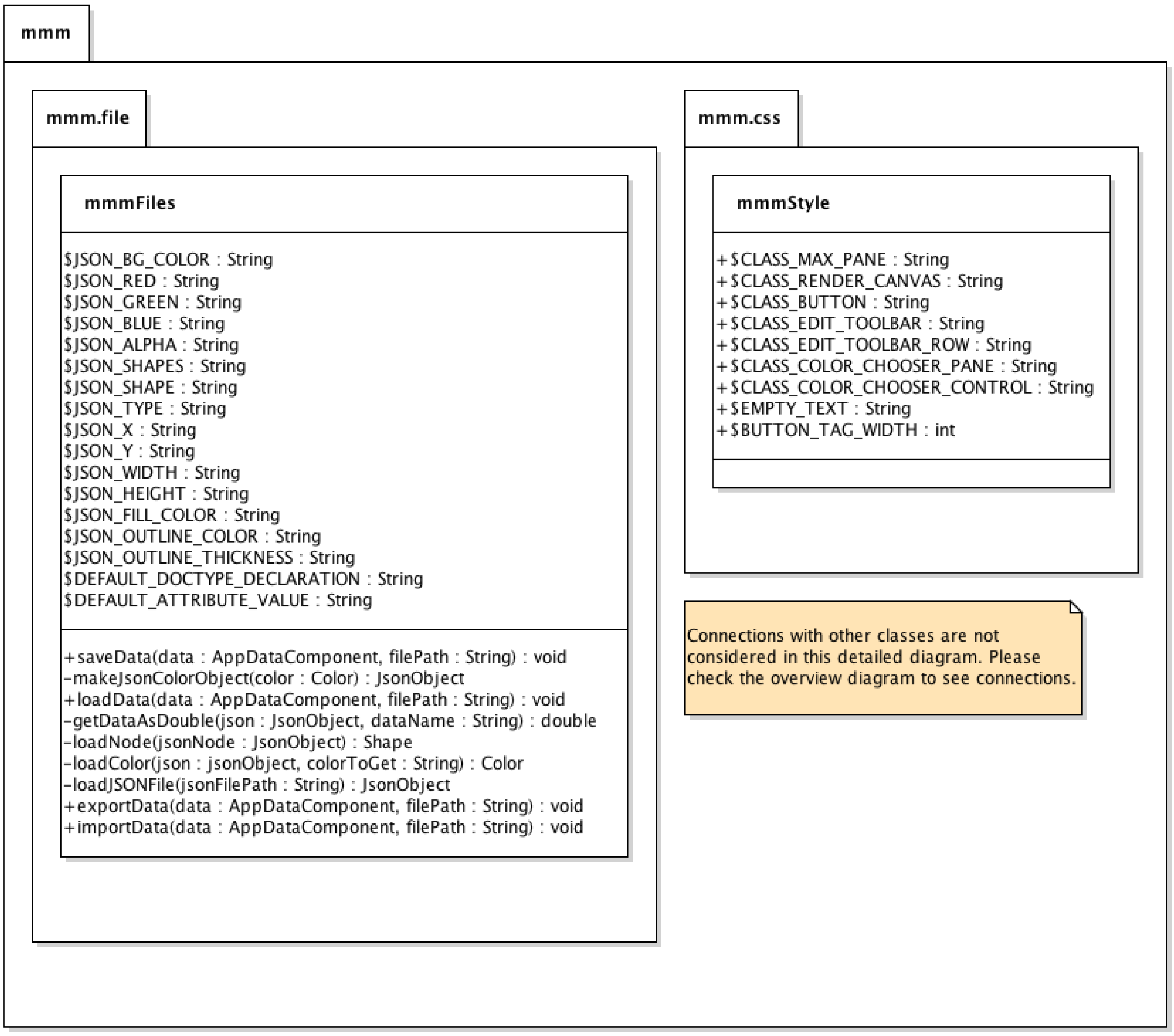

mmm
mmm.file
mmmFiles
$JSON_BG_COLOR : String
$JSON_RED : String
$JSON_GREEN : String
$JSON_BLUE : String
$JSON_ALPHA : String
$JSON_SHAPES : String
$JSON_SHAPE : String
$JSON_TYPE : String
$JSON_X : String
$JSON_Y : String
$JSON_WIDTH : String
$JSON_HEIGHT : String
$JSON_FILL_COLOR : String
$JSON_OUTLINE_COLOR : String
$JSON_OUTLINE_THICKNESS : String
$DEFAULT_DOCTYPE_DECLARATION : String
$DEFAULT_ATTRIBUTE_VALUE : String
+saveData(data : AppDataComponent, filePath : String) : void
-makeJsonColorObject(color : Color) : JsonObject
+loadData(data : AppDataComponent, filePath : String) : void
-getDataAsDouble(json : JsonObject, dataName : String) : double
-loadNode(jsonNode : JsonObject) : Shape
-loadColor(json : jsonObject, colorToGet : String) : Color
-loadJSONFile(jsonFilePath : String) : JsonObject
+exportData(data : AppDataComponent, filePath : String) : void
+importData(data : AppDataComponent, filePath : String) : void
mmm.css
mmmStyle
+$CLASS_MAX_PANE : String
+$CLASS_RENDER_CANVAS : String
+$CLASS_BUTTON : String
+$CLASS_EDIT_TOOLBAR : String
+$CLASS_EDIT_TOOLBAR_ROW : String
+$CLASS_COLOR_CHOOSER_PANE : String
+$CLASS_COLOR_CHOOSER_CONTROL : String
+$EMPTY_TEXT : String
+$BUTTON_TAG_WIDTH : int
Connections with other classes are not considered in this detailed diagram. Please check the overview diagram to see connections.

Figure 3.4: Detailed mmm.file and mmm.css UML Class Diagram

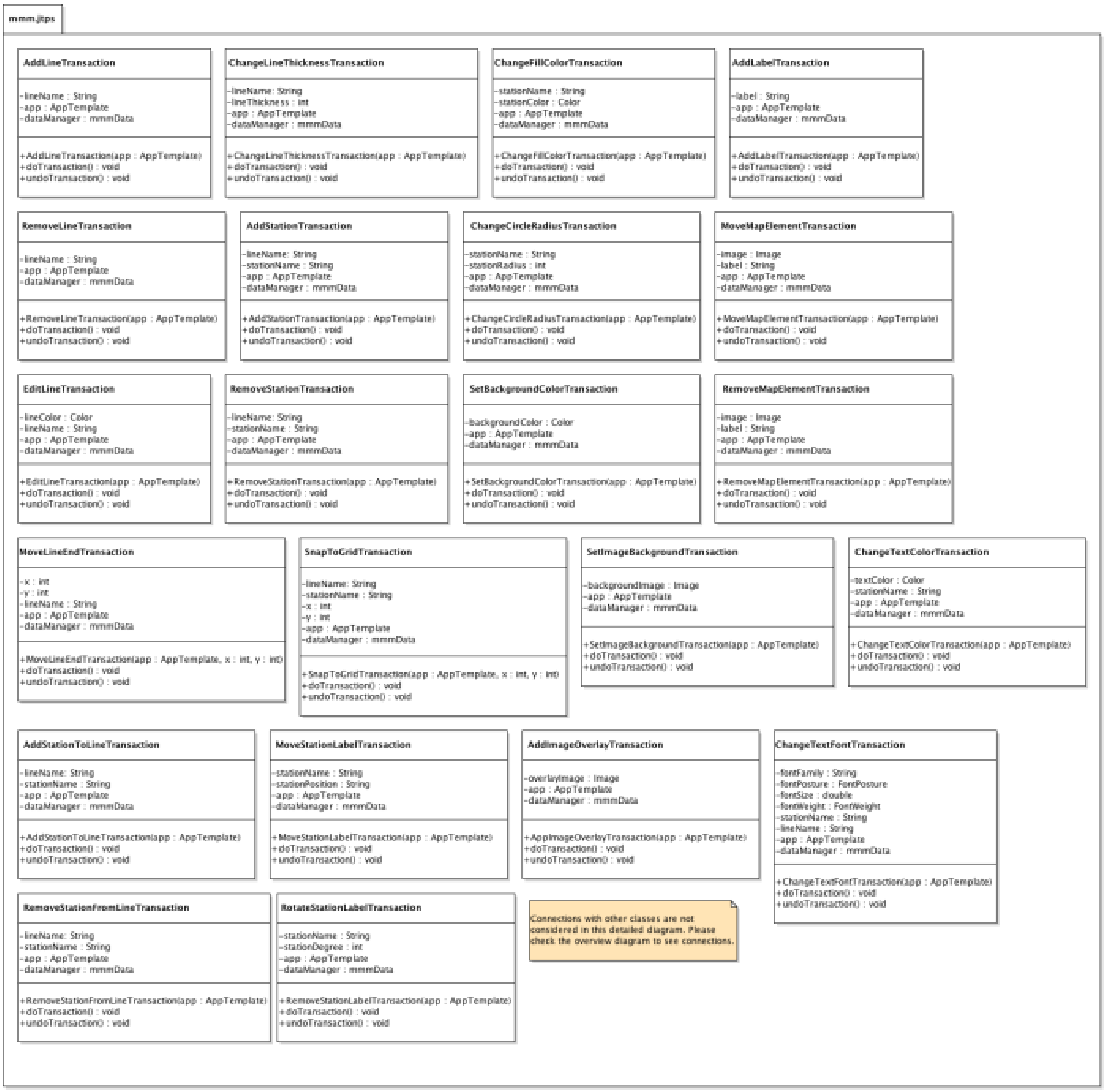


Figure 3.5: Detailed mmm.jtps UML Class Diagram

4 Method-Level Design Viewpoint

Now that the general architecture of the classes has been determined, it is time to specify how data will flow through the system. The following UML Sequence Diagrams describe the methods called within the code to be developed in order to provide the appropriate event responses.

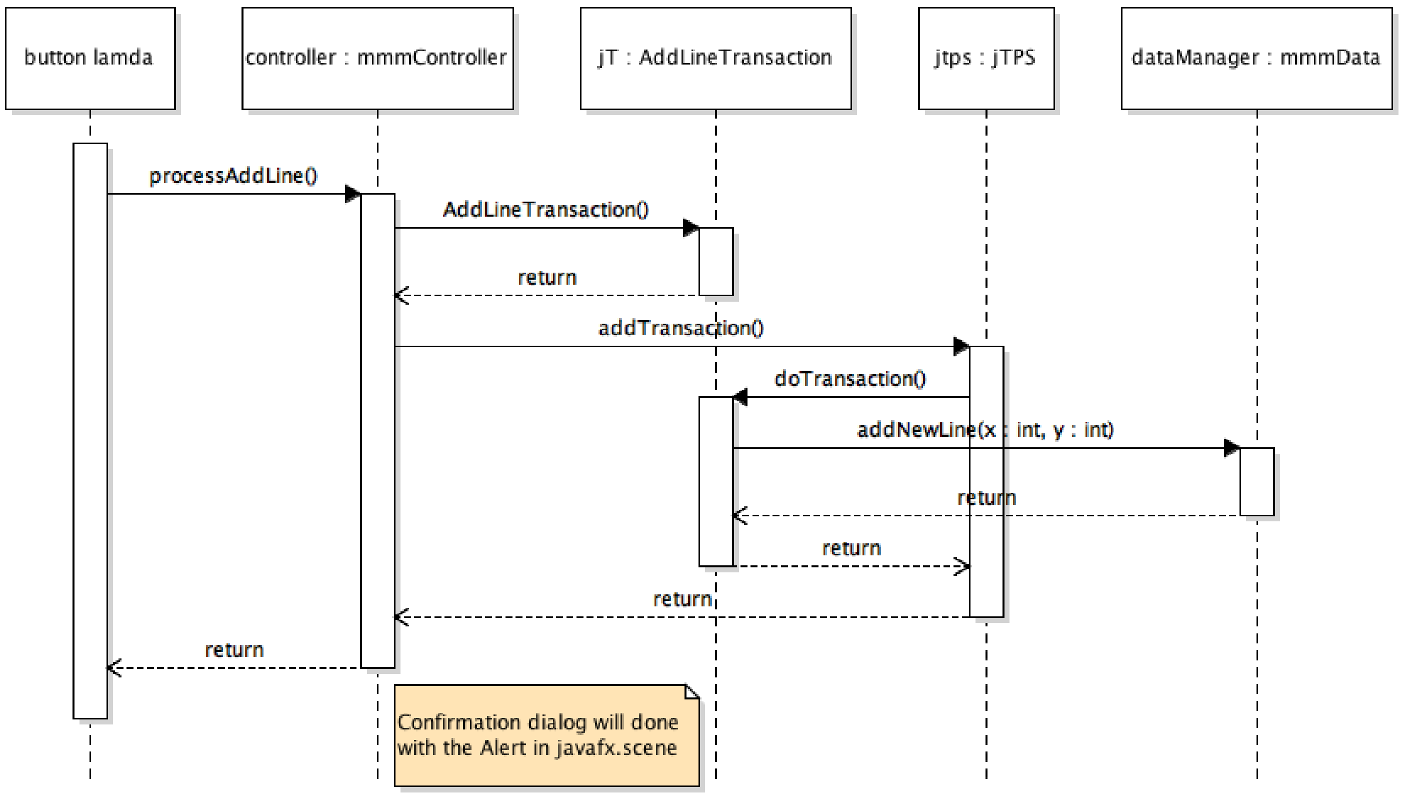


Figure 4.1: Add Line UML Sequence Diagrams

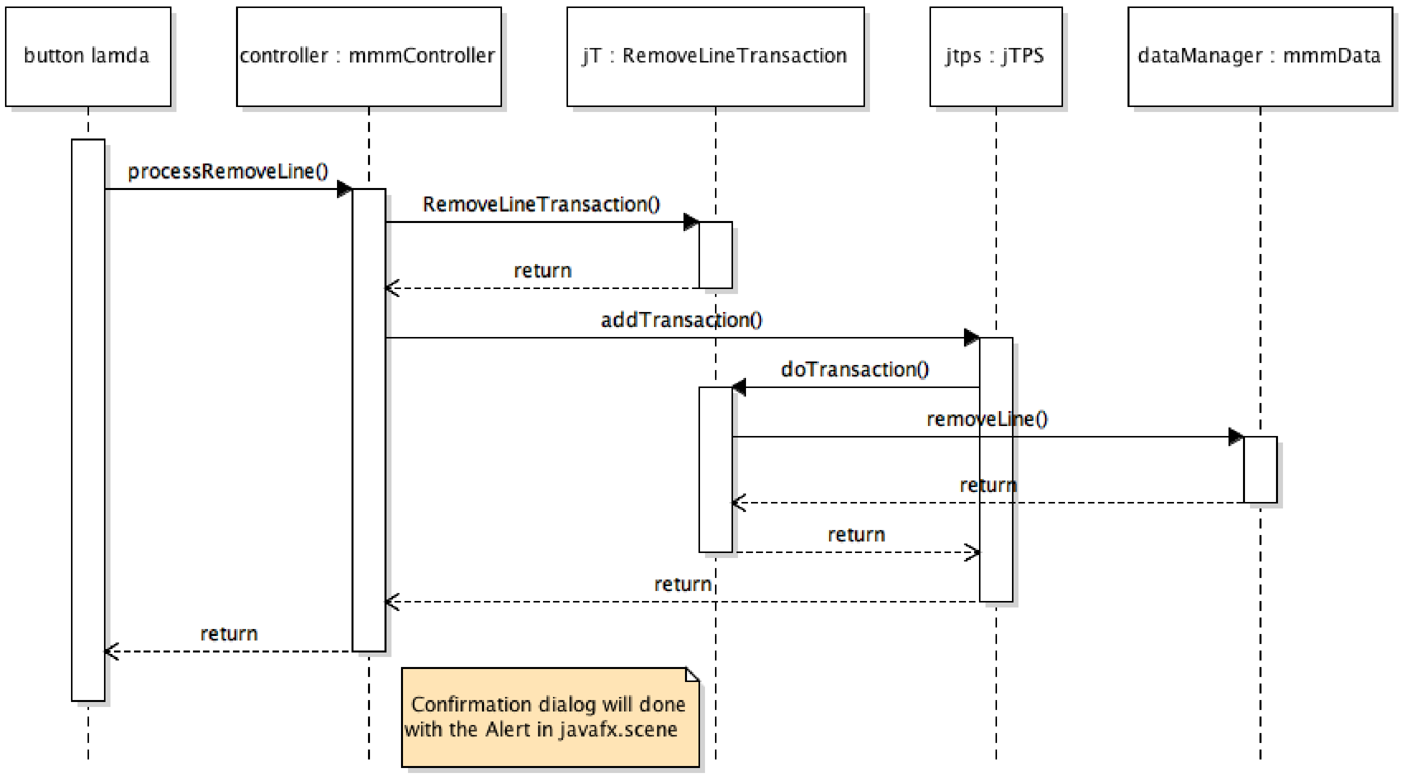


Figure 4.2: Remove Line UML Sequence Diagrams

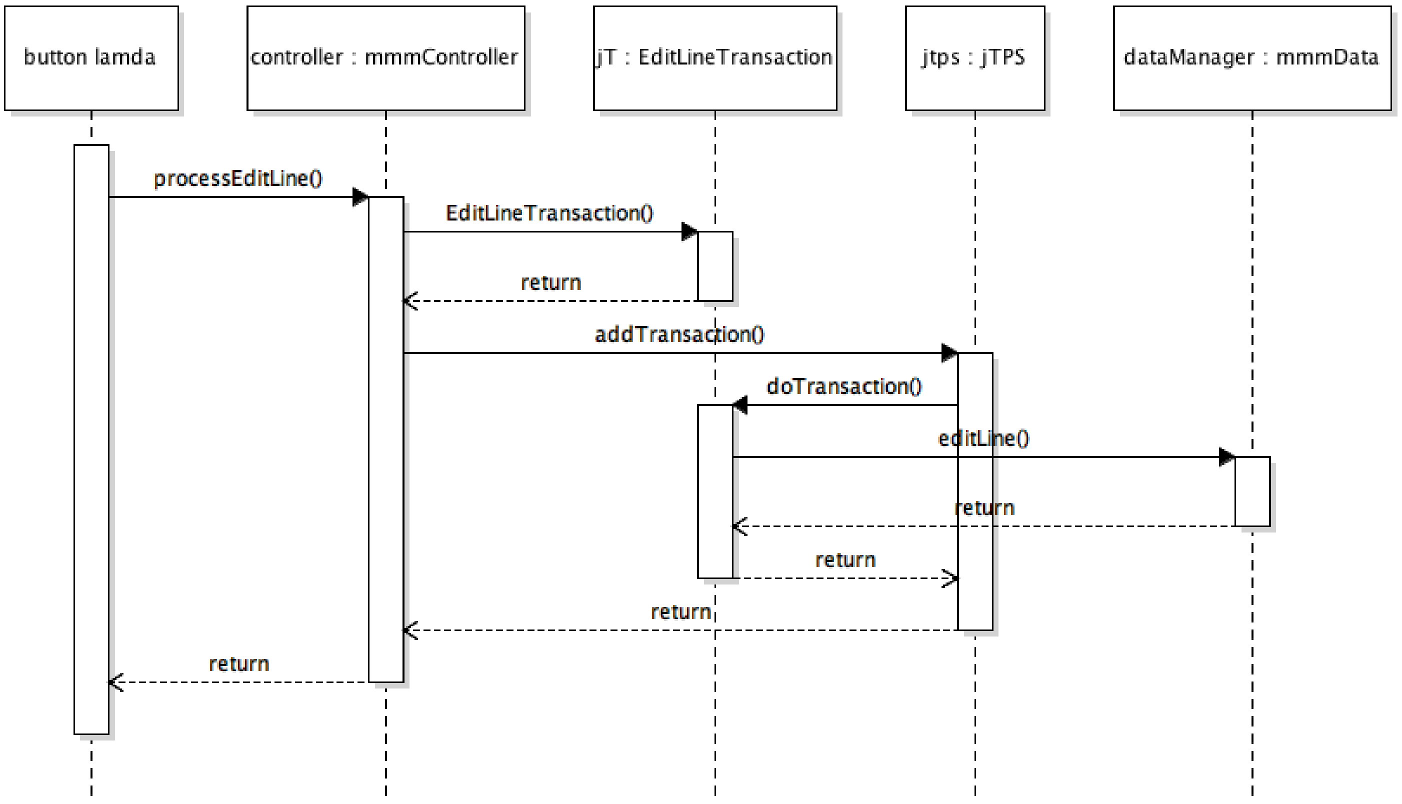


Figure 4.3: Edit Line UML Sequence Diagrams

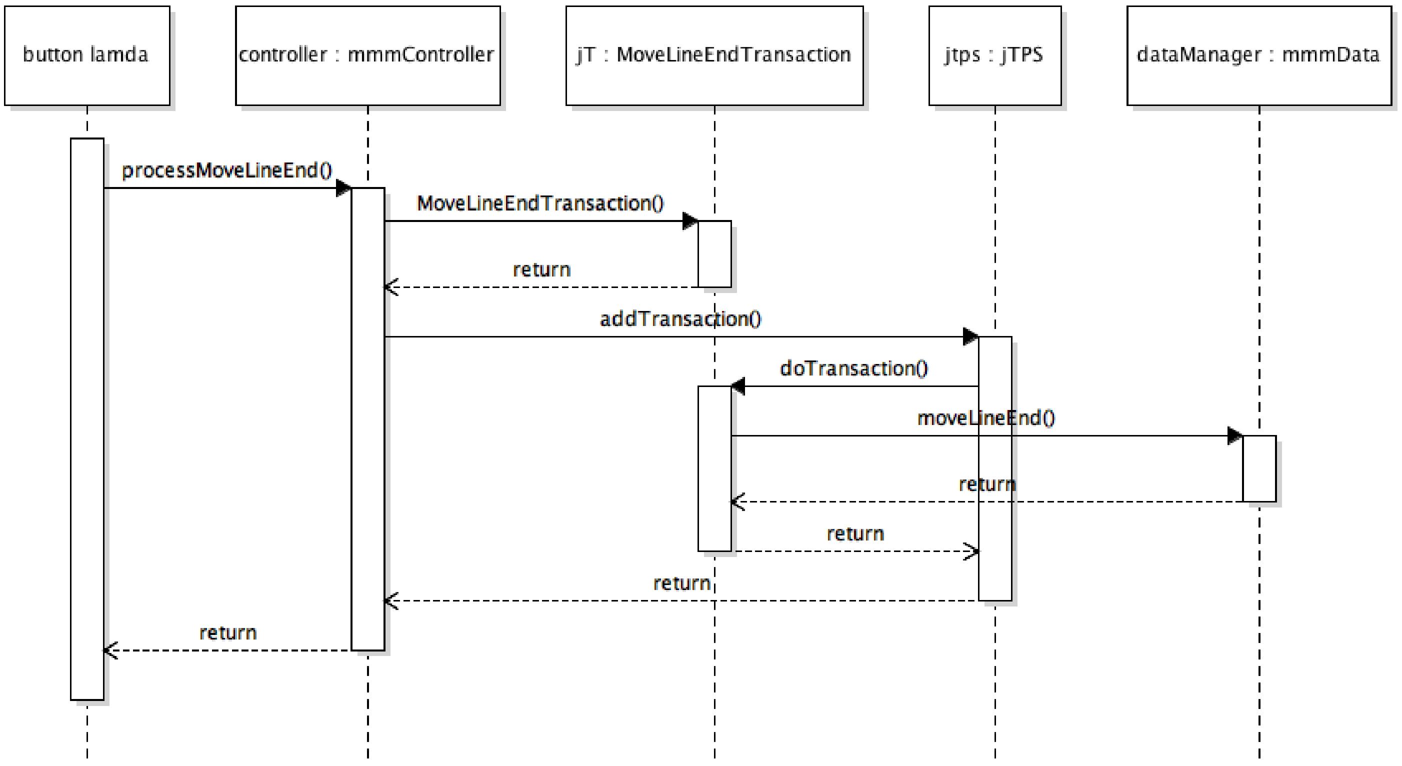


Figure 4.4: Move Line End UML Sequence Diagrams

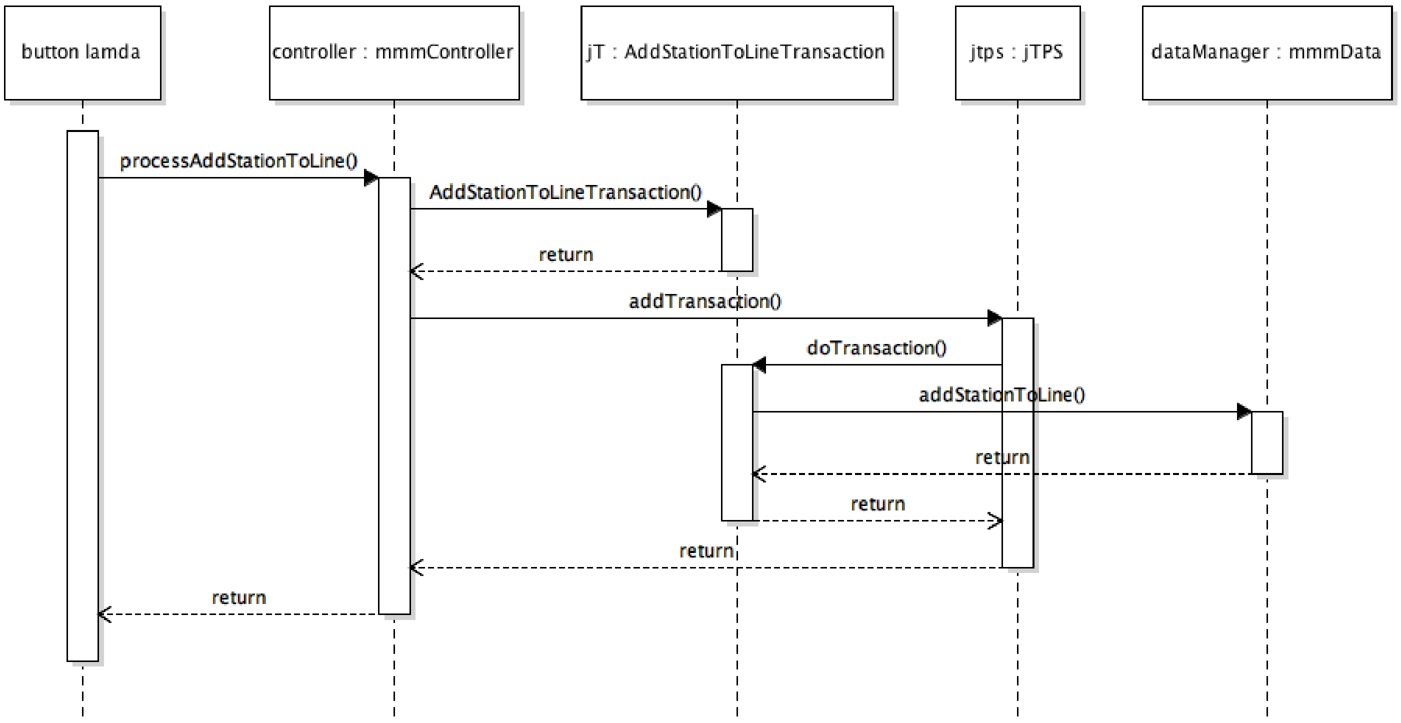


Figure 4.5: Add Station to Line UML Sequence Diagrams

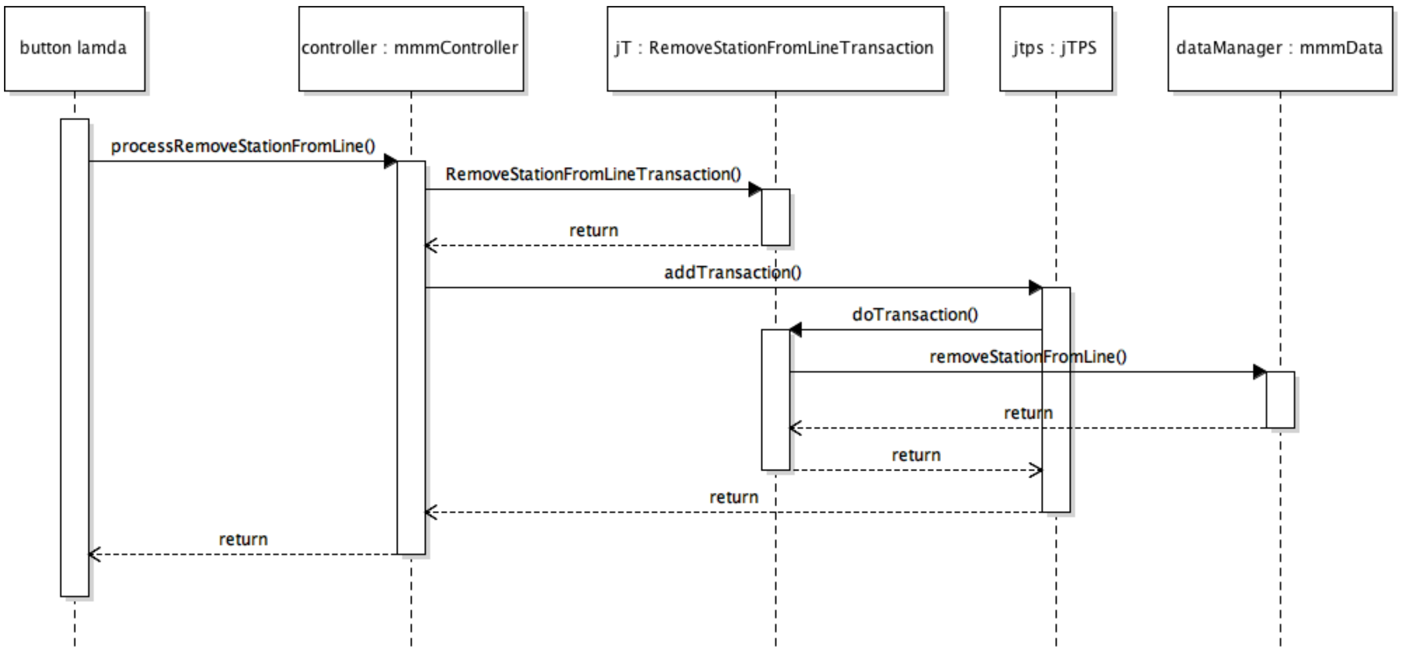


Figure 4.6: Remove Station from Line UML Sequence Diagrams

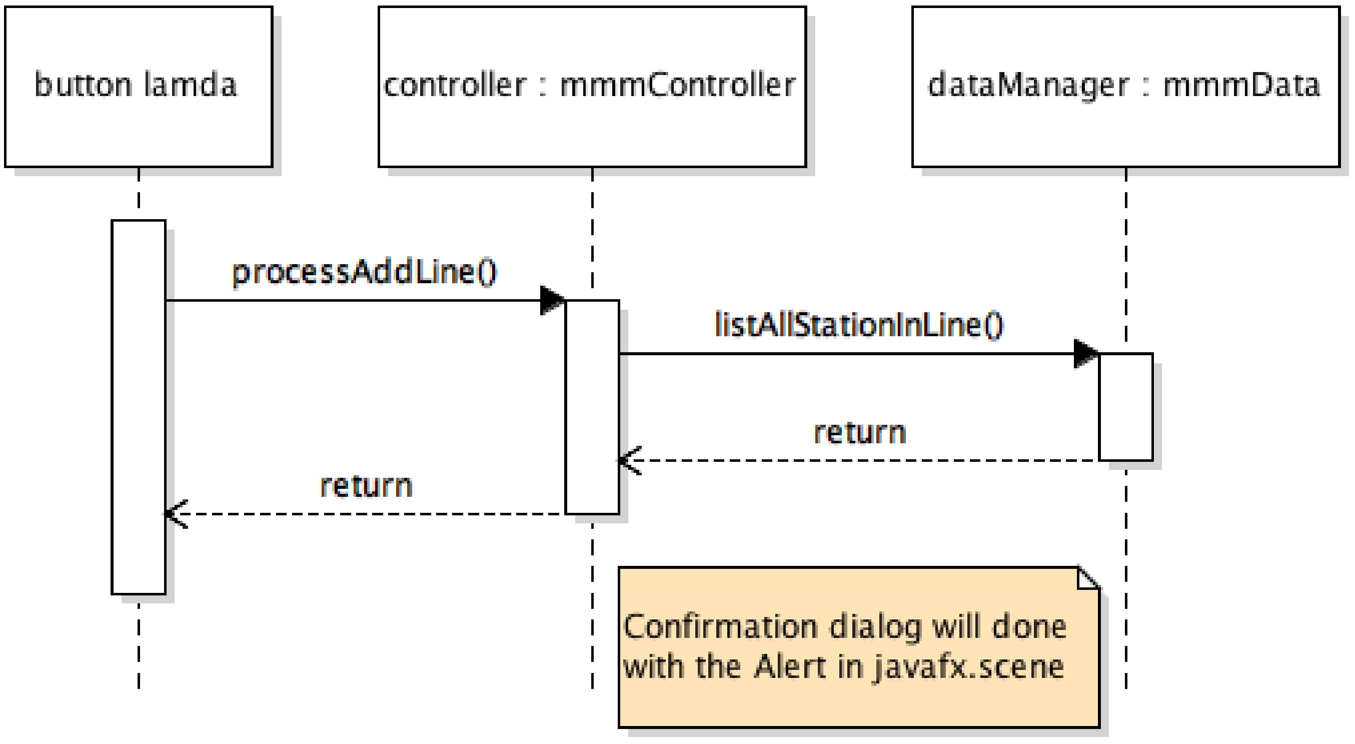

button lamda
controller : mmmController
dataManager : mmmData
processAddLine()
listAllStationInLine()
return
return
Confirmation dialog will done with the Alert in javafx.scene

Figure 4.7: List All Stations in Line UML Sequence Diagrams

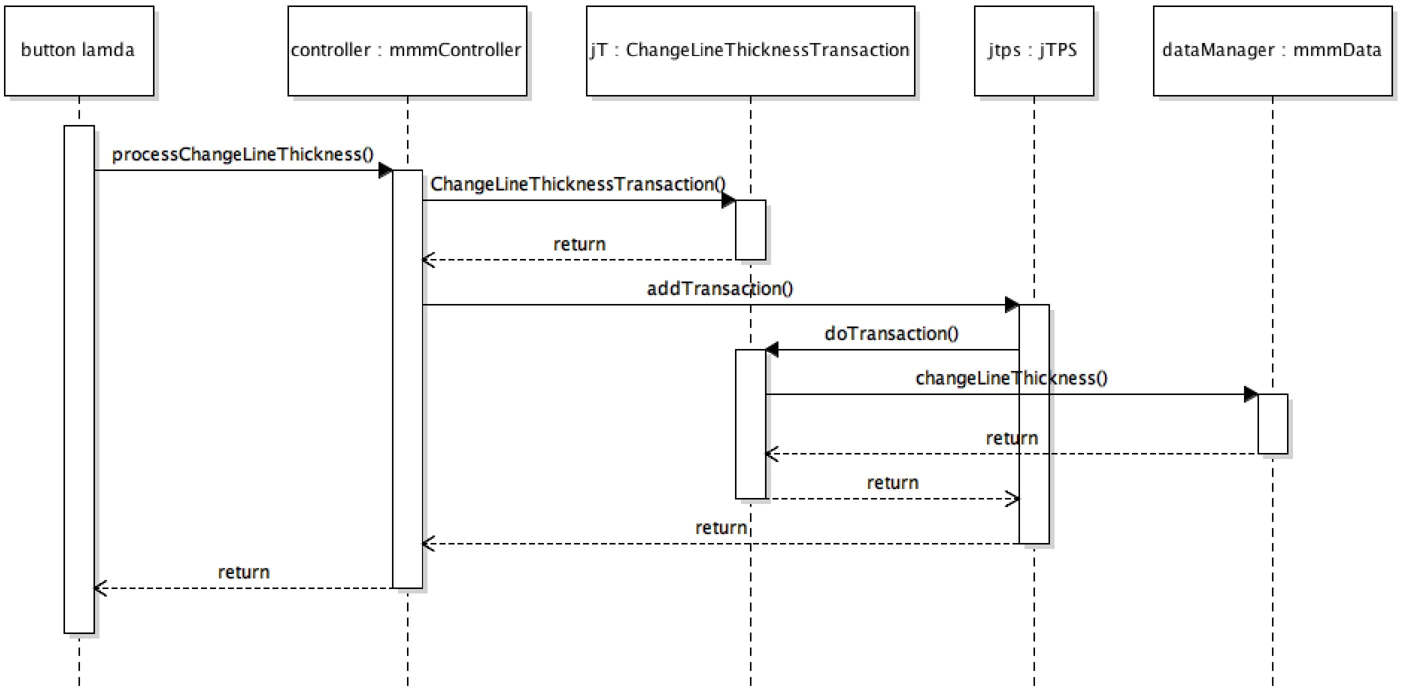


Figure 4.8: Change Line Thickness in LineUML Sequence Diagrams

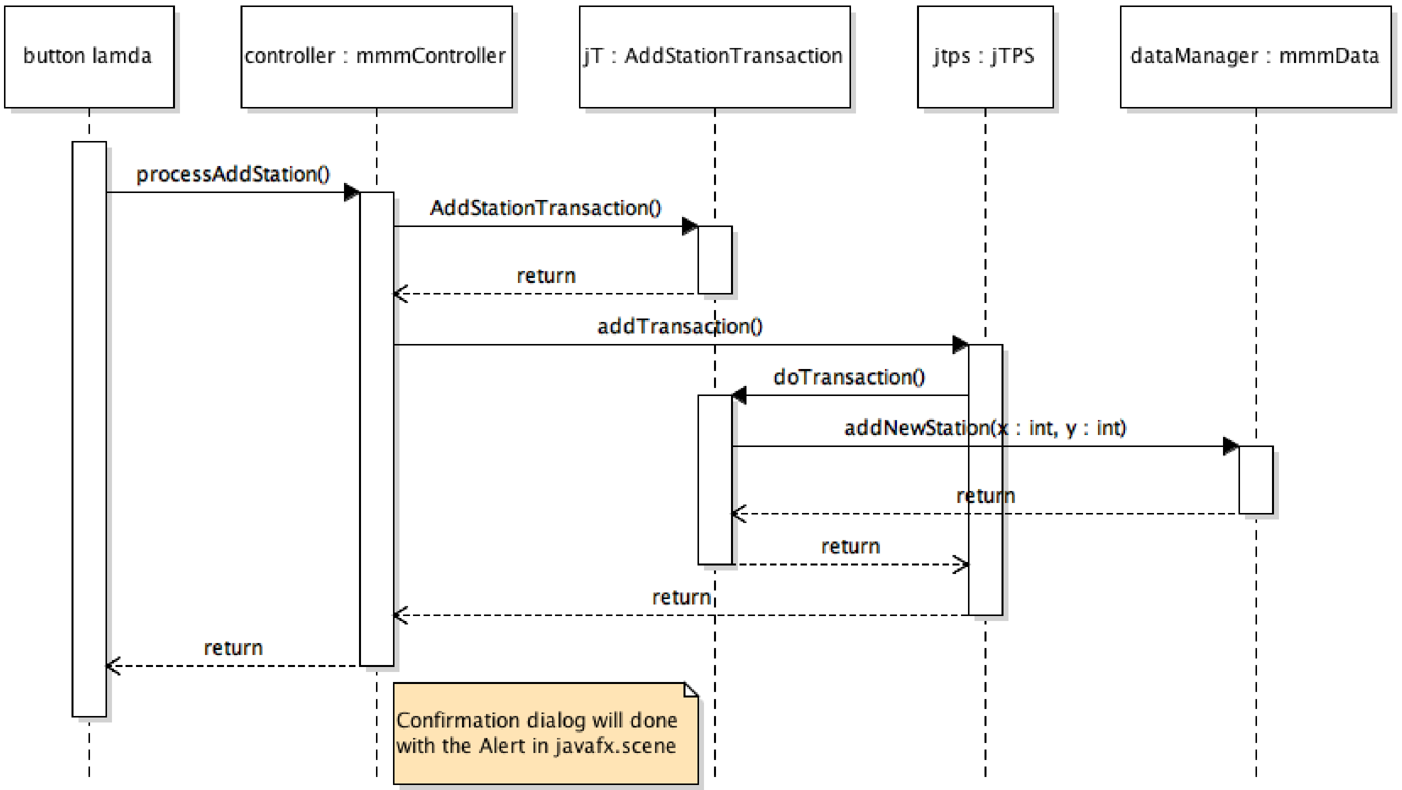


Figure 4.9: Add New Station UML Sequence Diagrams

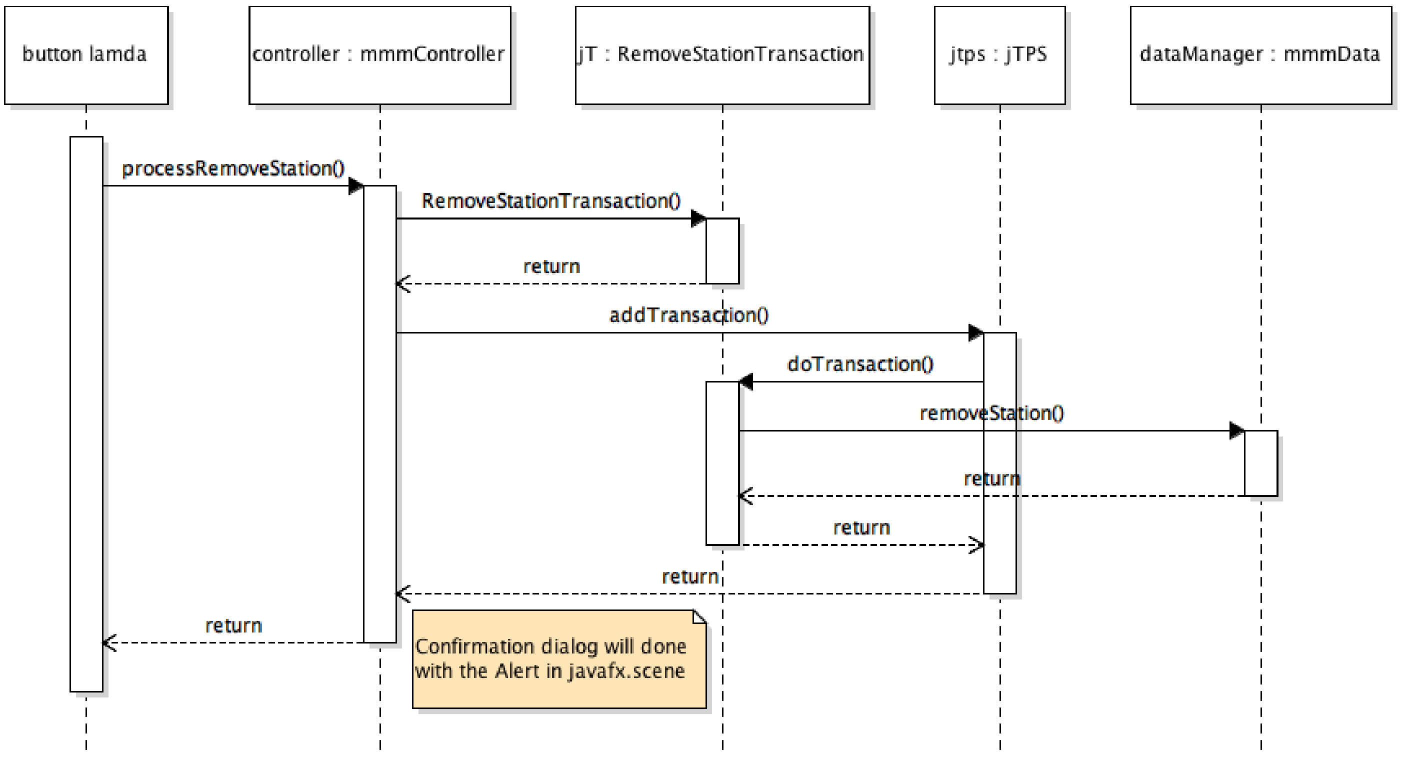


Figure 4.10: Remove Station UML Sequence Diagrams

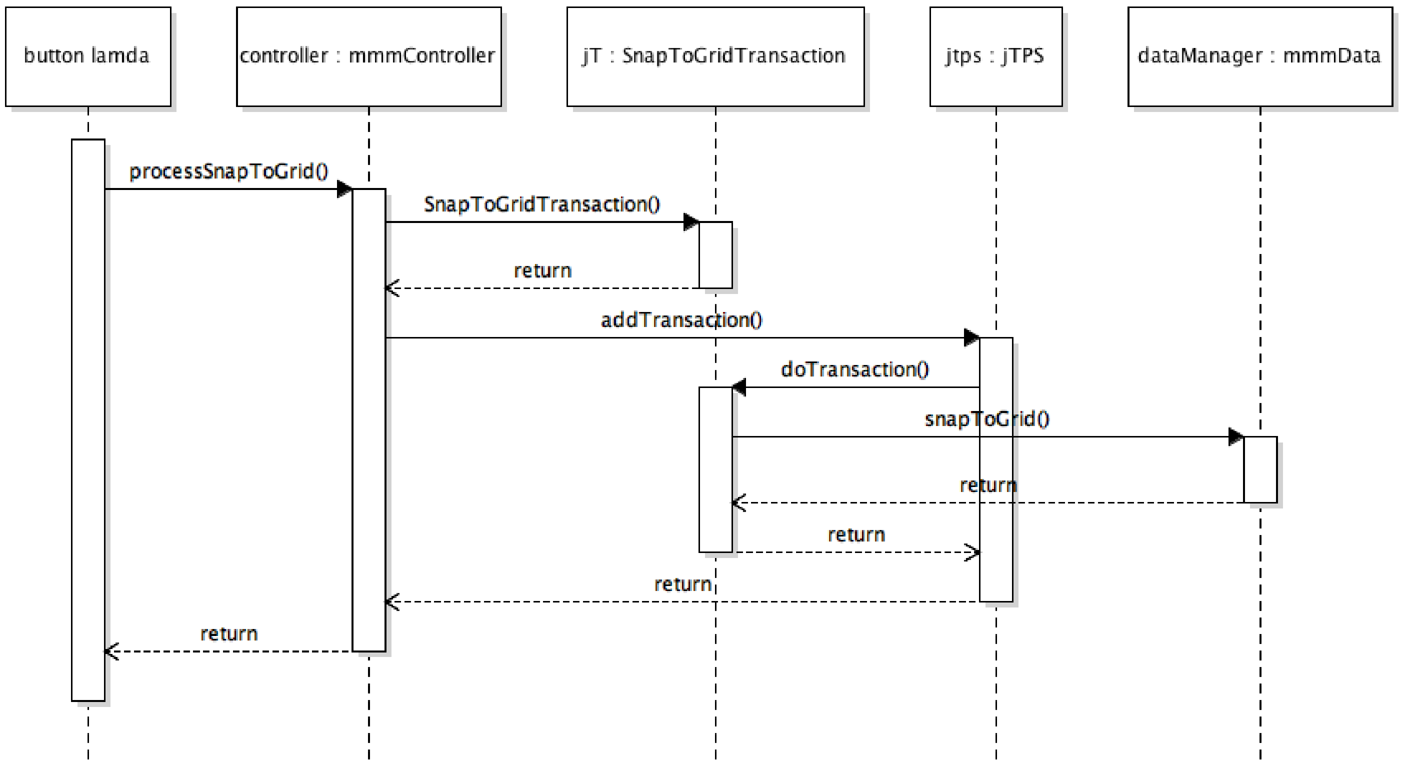


Figure 4.11: Snap to Grid UML Sequence Diagrams

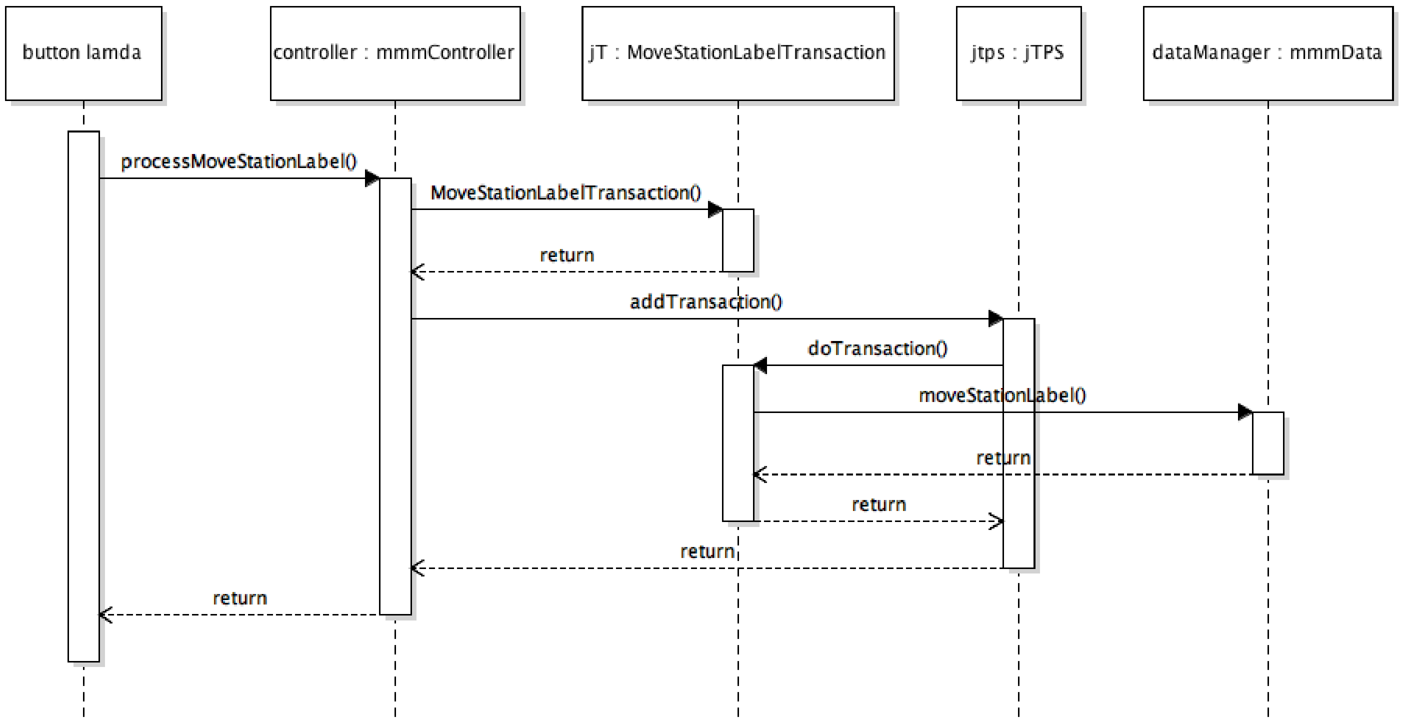


Figure 4.12: Move Station Label UML Sequence Diagrams

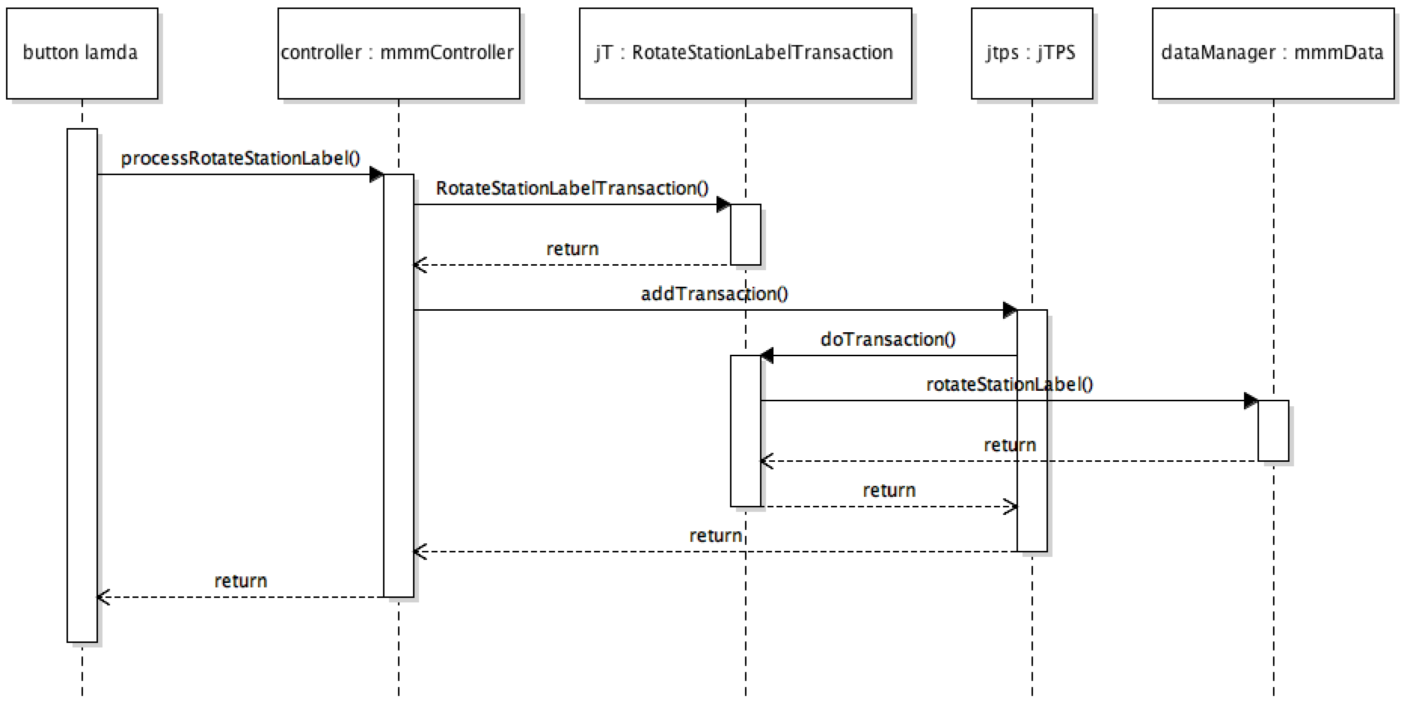


Figure 4.13: Rotate Station Label UML Sequence Diagrams

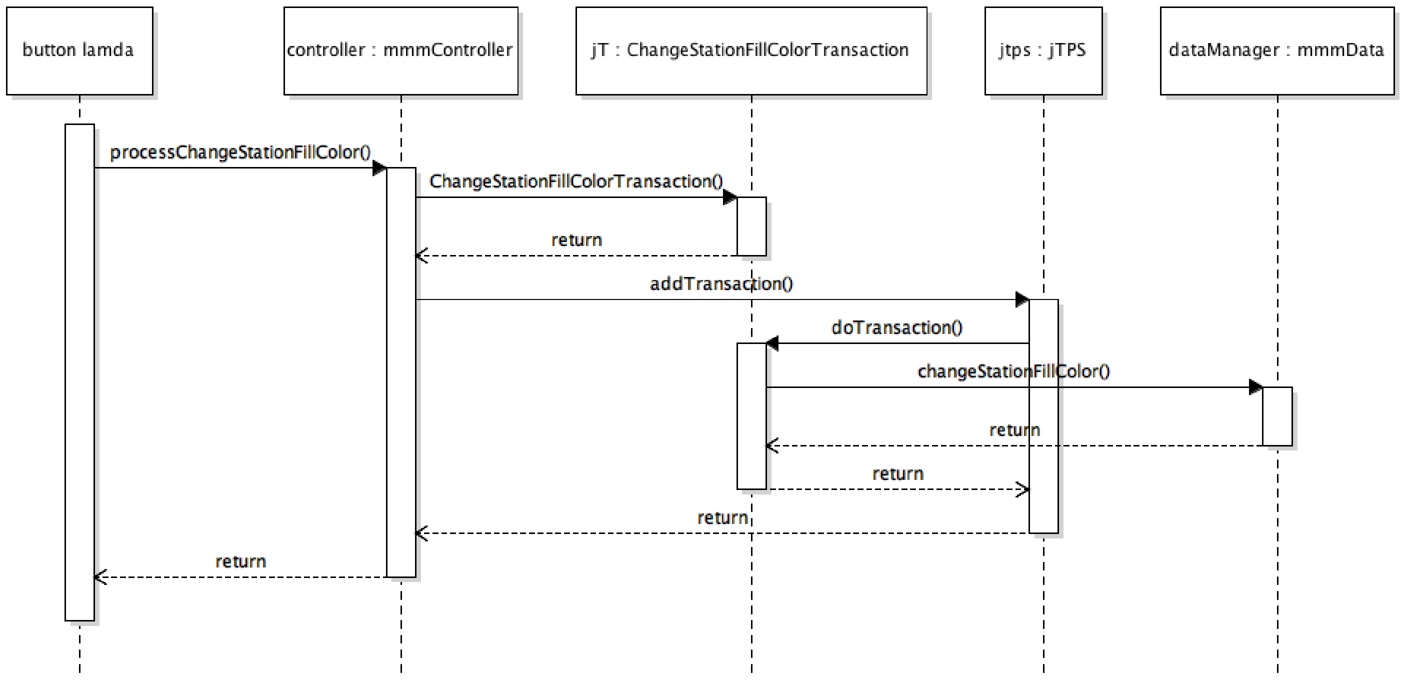

button lamda
controller : mmmController
jT : ChangeStationFillColorTransaction
jtps : jTPS
dataManager : mmmData
processChangeStationFillColor()
ChangeStationFillColorTransaction()
return
addTransaction()
doTransaction()
changeStationFillColor()
return
return
return
return

Figure 4.14: Change Station Fill Color UML Sequence Diagrams

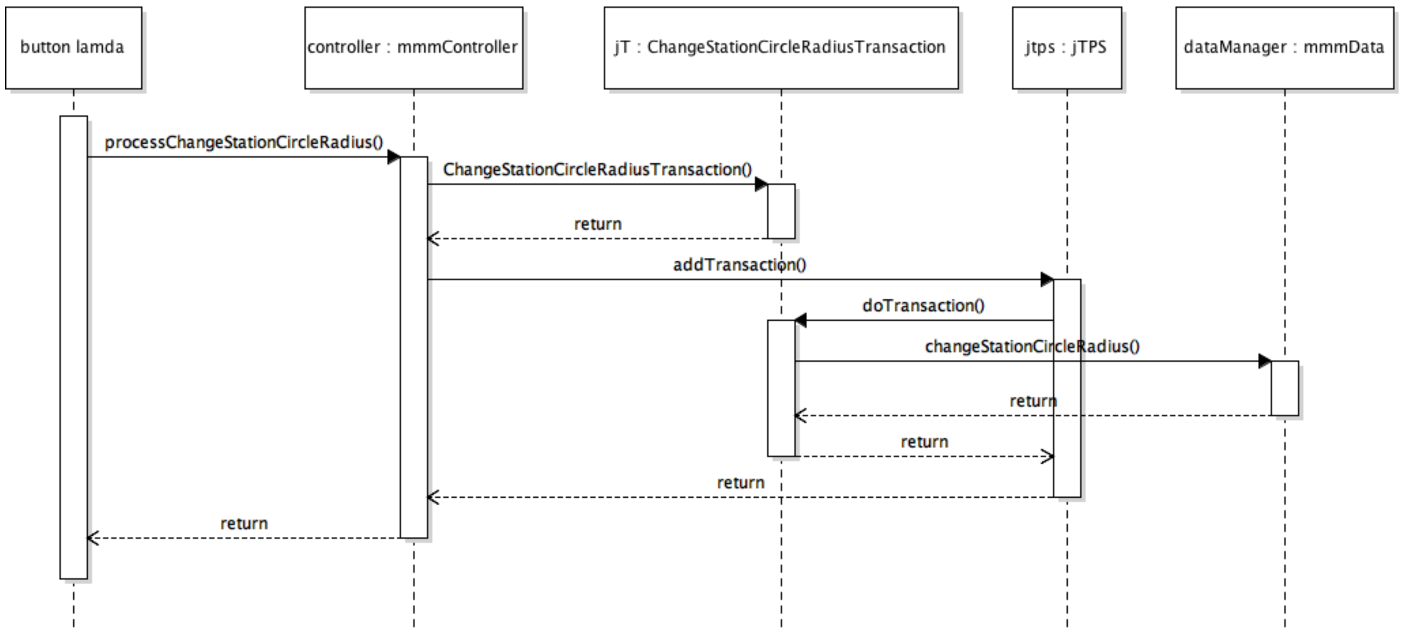

button lamda
controller : mmmController
jT : ChangeStationCircleRadiusTransaction
jtps : jTPS
dataManager : mmmData
processChangeStationCircleRadius()
ChangeStationCircleRadiusTransaction()
return
addTransaction()
doTransaction()
changeStationCircleRadius()
return
return
return
return

Figure 4.15: Change Station Circle Radius UML Sequence Diagrams

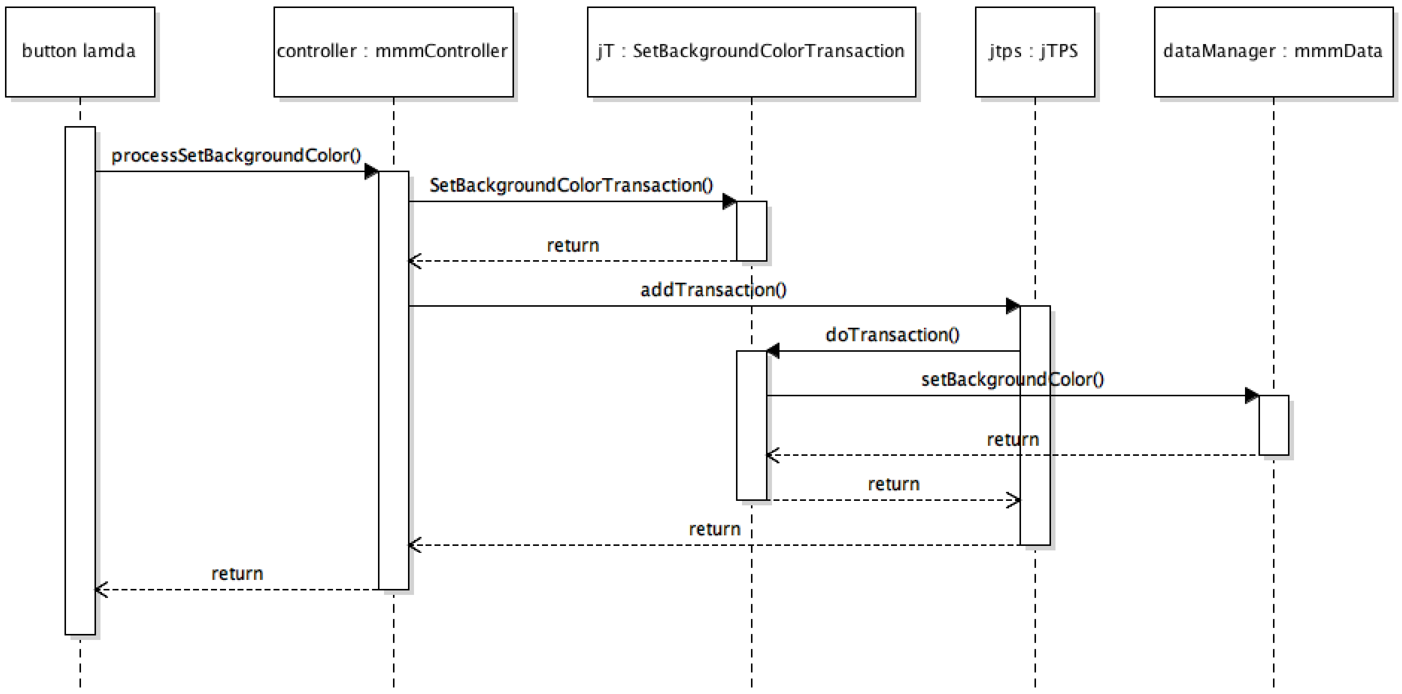


Figure 4.16: Set Background Color UML Sequence Diagrams

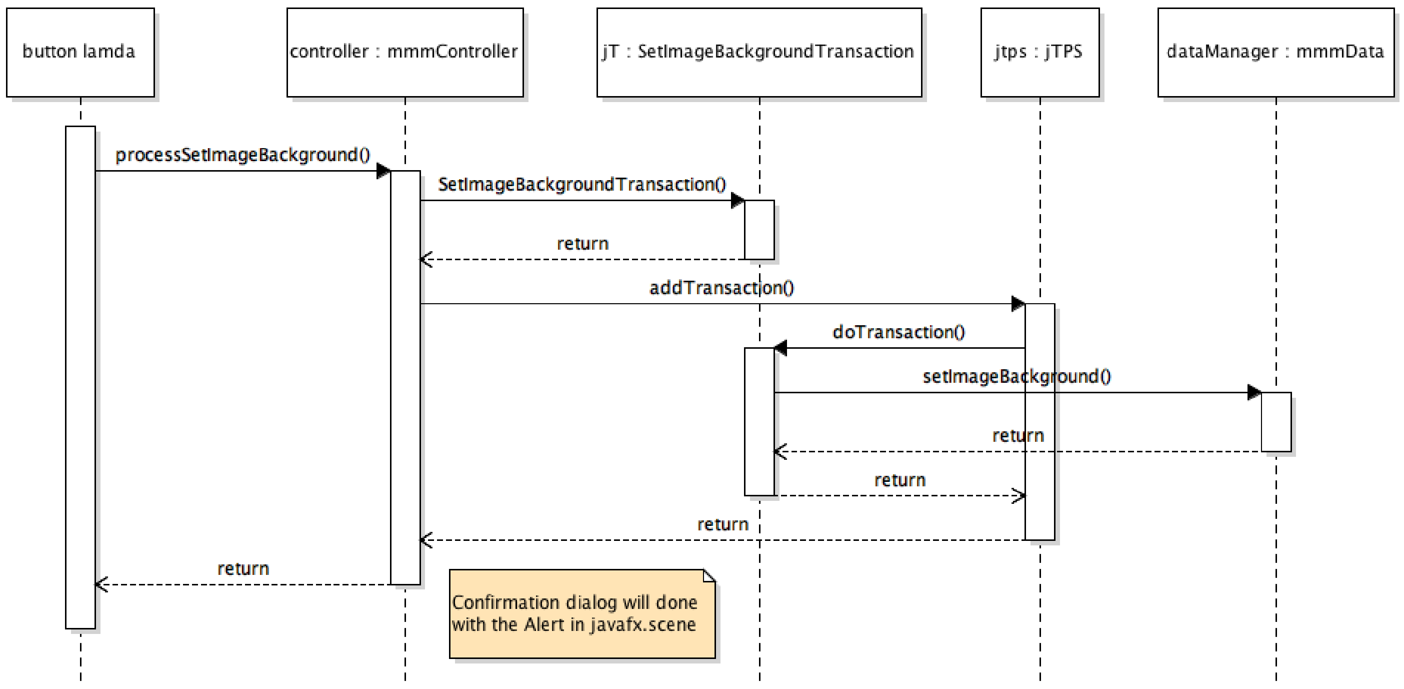

button lamda
controller : mmmController
jT : SetImageBackgroundTransaction
jtps : jTPS
dataManager : mmmData
processSetImageBackground()
SetImageBackgroundTransaction()
return
addTransaction()
doTransaction()
setImageBackground()
return
return
return
return
Confirmation dialog will done with the Alert in javafx.scene

Figure 4.17: Set Image Background UML Sequence Diagrams

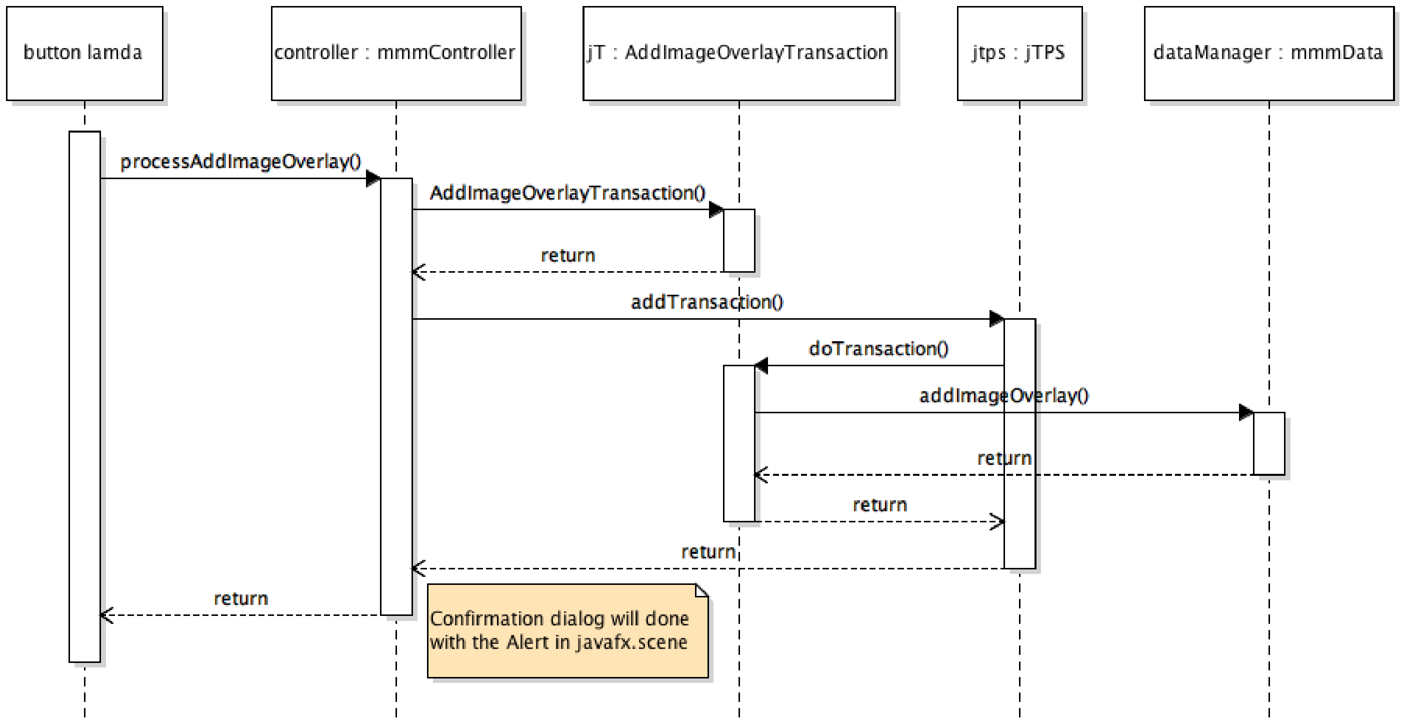


Figure 4.18: Add Image Overlay UML Sequence Diagrams

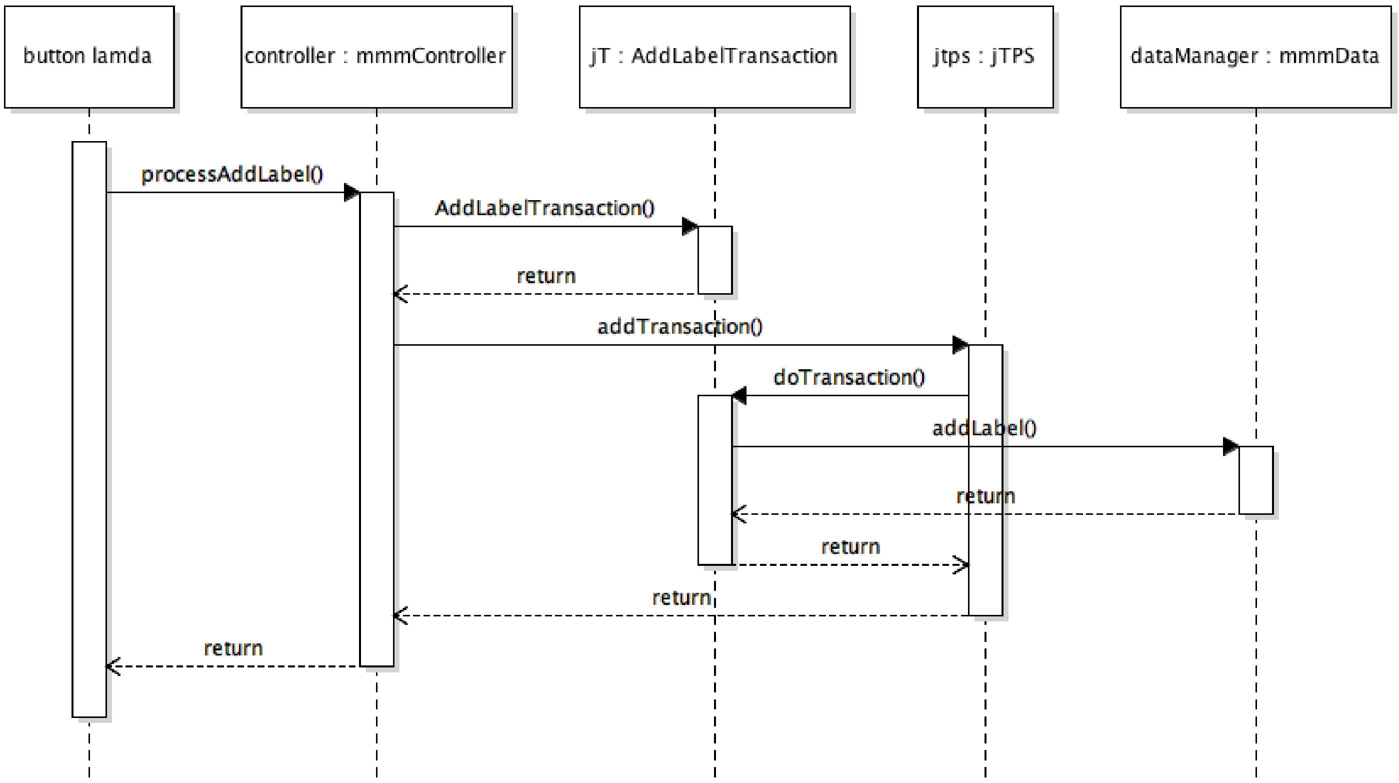


Figure 4.19: Add Label UML Sequence Diagrams

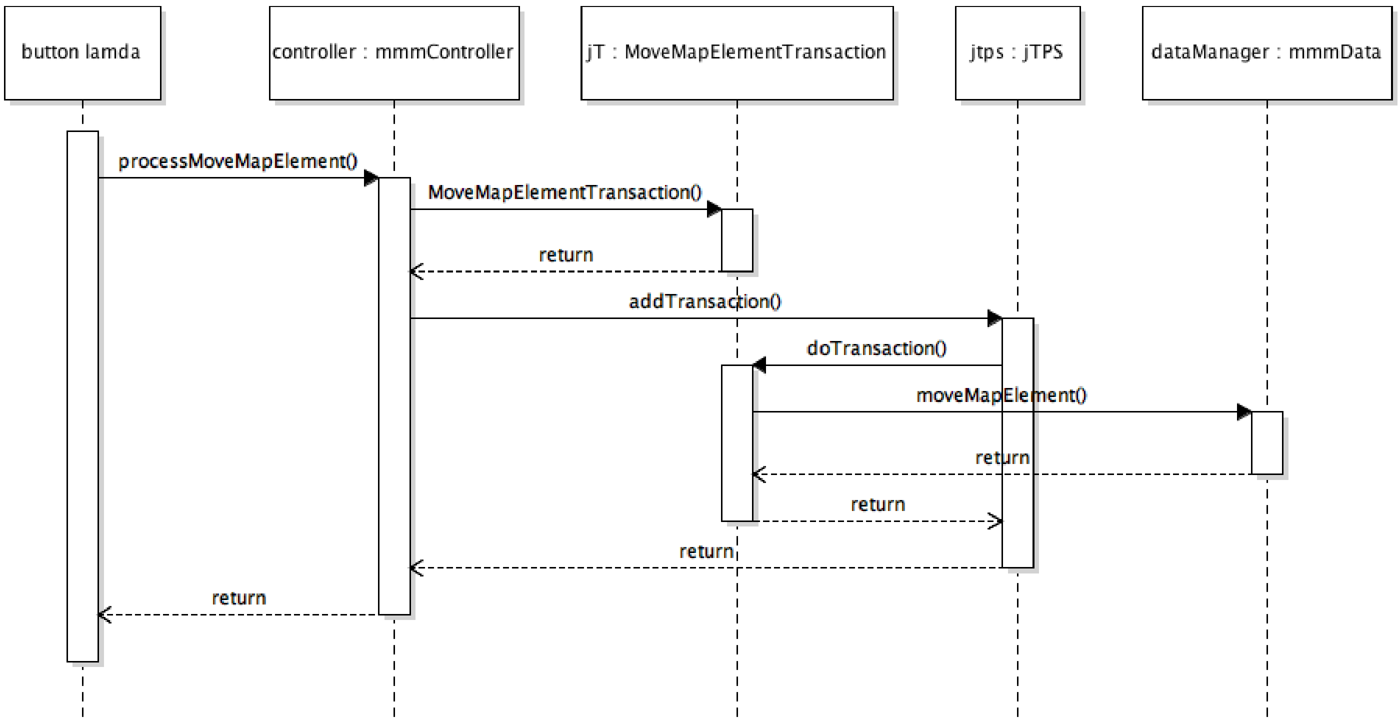


Figure 4.20: Move Map Element UML Sequence Diagrams

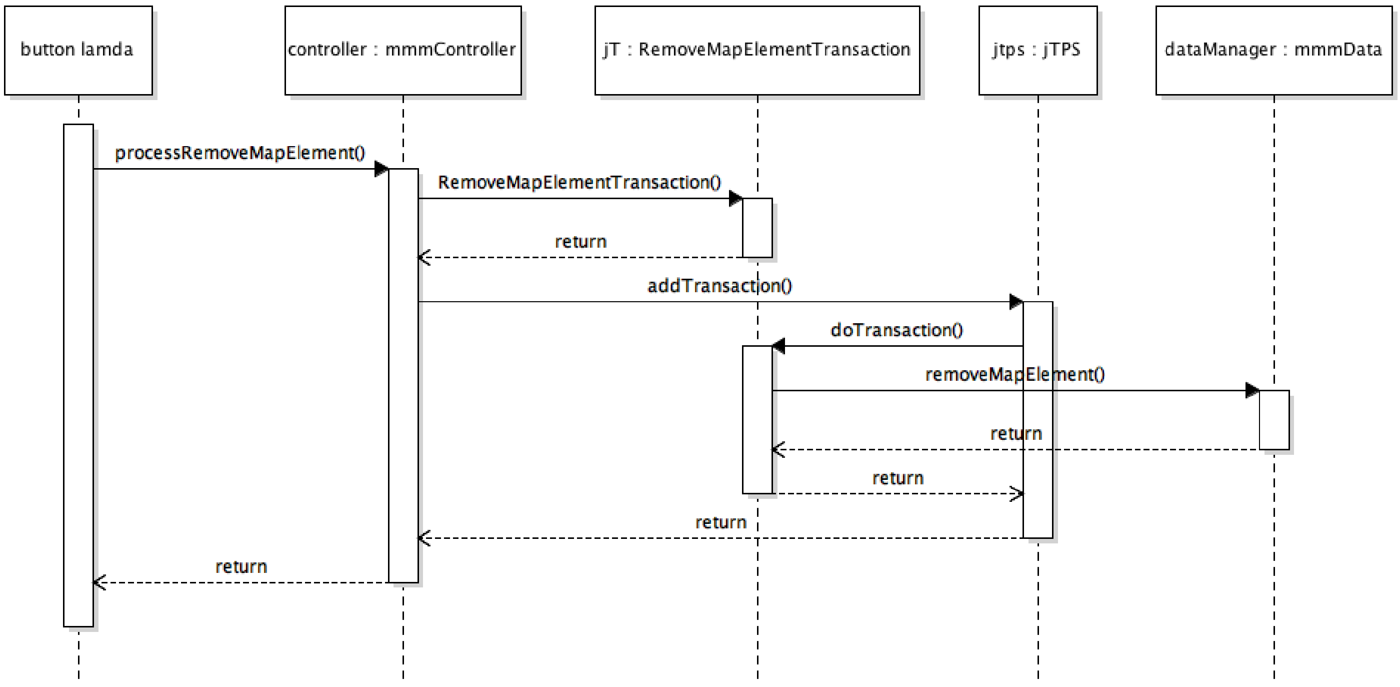


Figure 4.21: Remove Map Element UML Sequence Diagrams

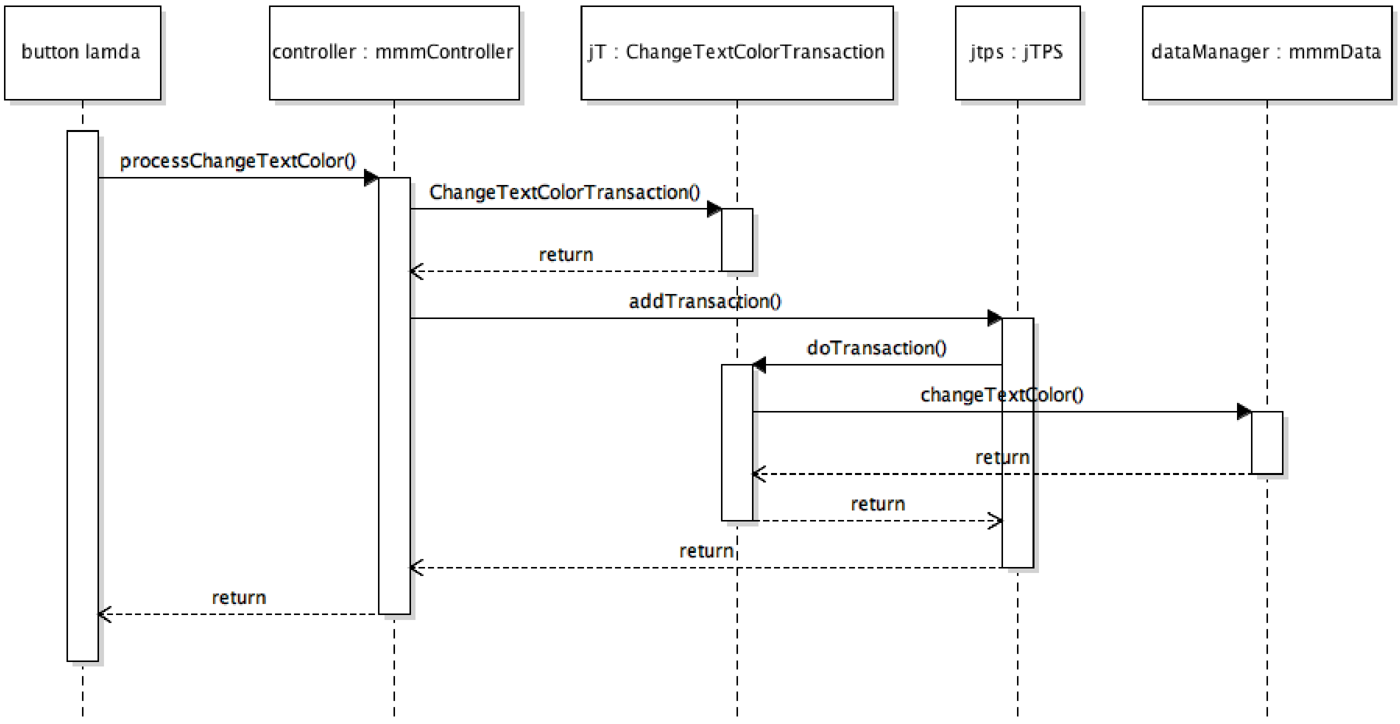


Figure 4.22: Change Text Color UML Sequence Diagrams

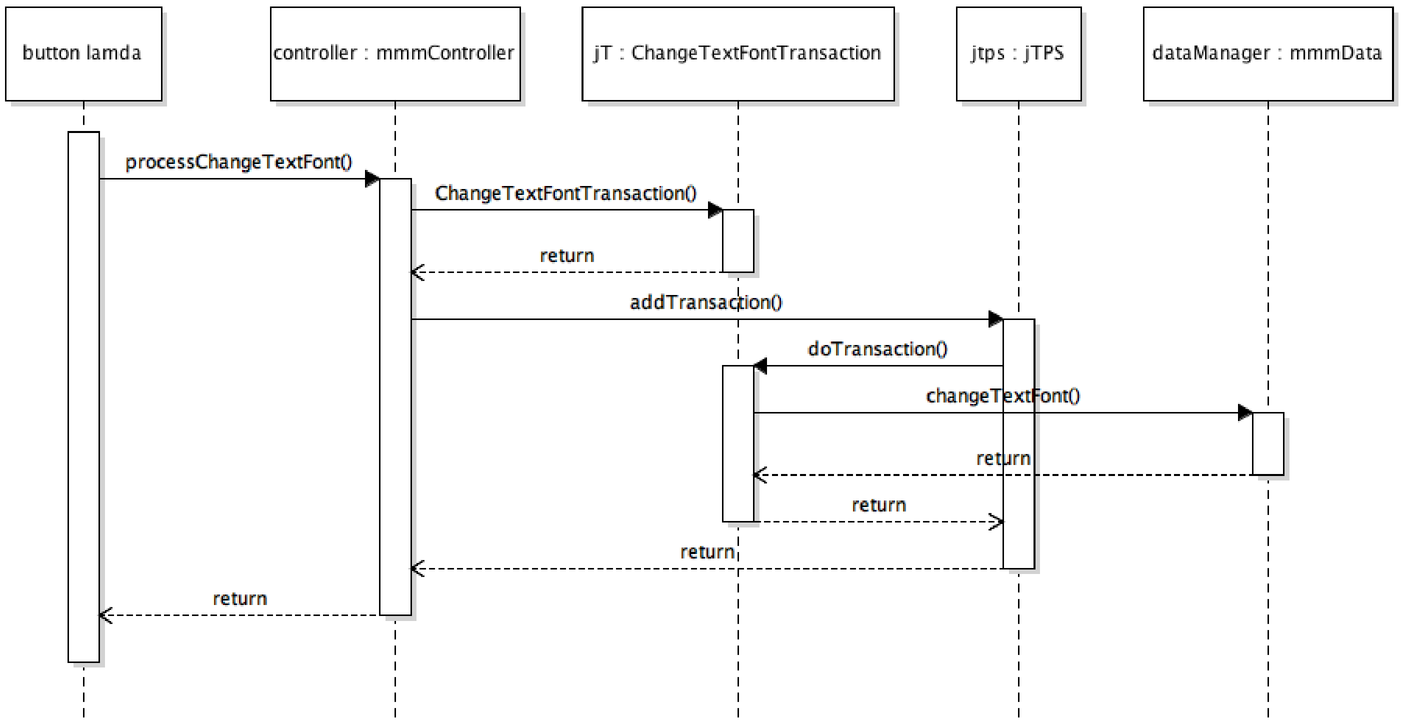
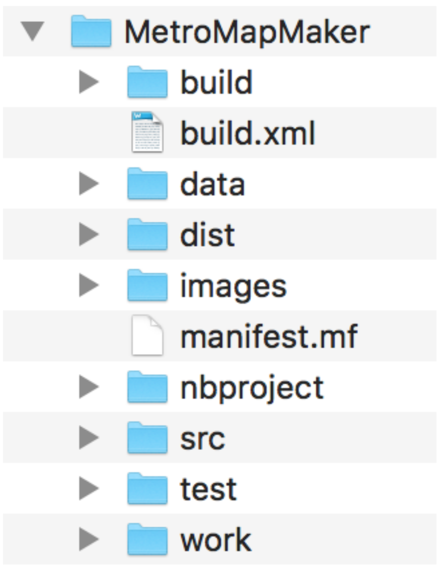


Figure 4.23: Change Text Font UML Sequence Diagrams

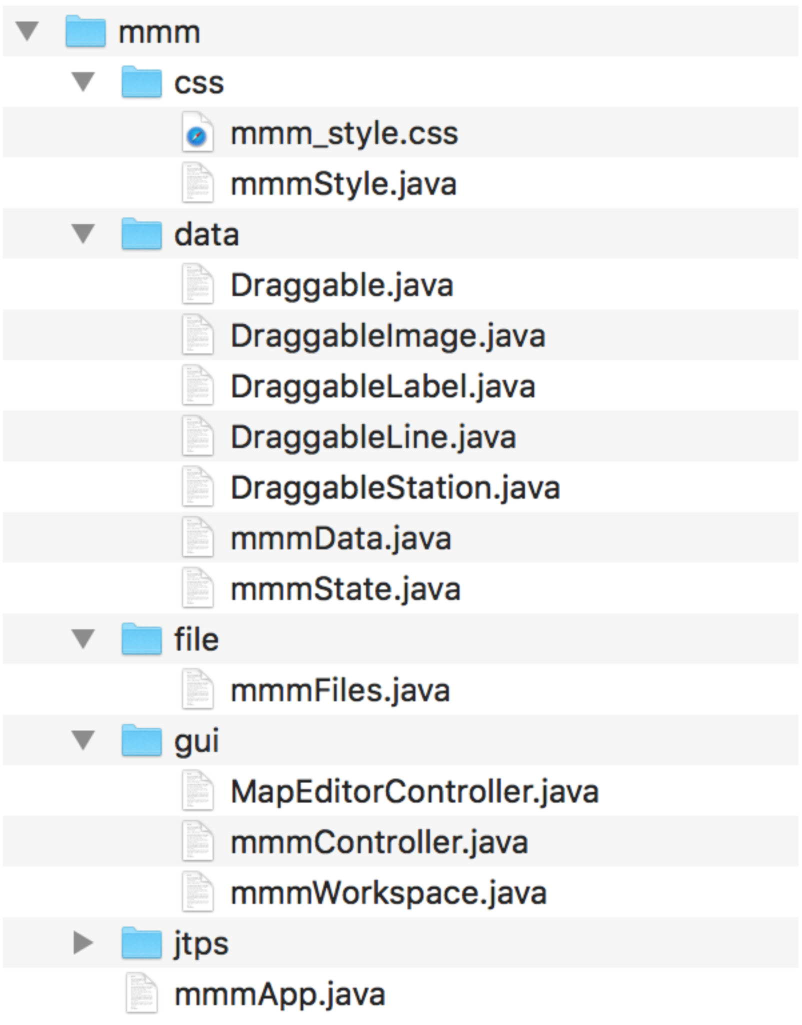
5 File Structure and Formats

The Metro Map Maker application will be deployed of a single, executable JAR file titled MetroMapMaker.jar with including the DesktopJavaFramework, PropertiesManager and jTPS. 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. Figure 5.2 specifies the necessary file structure the src/mmm directory should have. Note that all necessary images should of course go in the image directory.



```
MetroMapMaker
    build
    build.xml
    data
    dist
    images
    manifest.mf
    nbproject
    src
    test
    work
```

Figure 5.1: MetroMapMaker File Structure



```
mmm
    css
        mmm_style.css
        mmmStyle.java
    data
        Draggable.java
        DraggableImage.java
        DraggableLabel.java
        DraggableLine.java
        DraggableStation.java
        mmmData.java
        mmmState.java
    file
        mmmFiles.java
    gui
        MapEditorController.java
        mmmController.java
        mmmWorkspace.java
    jtps
    mmmApp.java
```

Figure 5.2: MetroMapMaker/src/mmm File Structure

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