DATAVILIJ™

Software Requirements Specification

DATAVILIJ™

Author: Siyuan Zou
The Dot Inc™

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

Table of Contents

1. Introduction

With the increase in demand and popularity of data-driven artificial intelligence (AI) in computer science, visualizing how AI algorithms work is becoming increasingly important. Java is among the most important programming languages used to implement these algorithms, however it lacks in availability of visualization libraries. More specifically visualization libraries that shows the how these algorithms learn from the data.

This document is the Software Design Description (SDD) for DataViLiJ™ application.

DataViLiJ (Data Visualization Library in Java) will be a deskt op application that will allow users to select an algorithm (from a set of standard AI algorithms) and dynamically show the user what changes, and how.

1.1 Purpose

The purpose of this document is to serve as the blueprint for the construction of DataViLiJ™ 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, the UML Sequence diagrams will be used in specifying how object interact with the user. Upon completing this document, the reader should understand how this application should look and operate.

1.2 Intended Audience

The intended audience for this document is the development team, including instructors, software designer, and software developers.

1.3 Product Scope

The goal of this project is for students and beginning professionals in AI to have a visual understanding of the inner workings of the fundamental algorithms. AI is a vast field, and this project is limited to the visualization of two types of algorithms that "learn" from data. These two types are called **clustering** and **classification**. The design and development of these algorithms is outside the scope of the project, and the assumption is that such algorithms will already be developed independently, and their output will comply with the data format specified in this document. DataViLiJ serves simply as a visualization tool for how those algorithms work. Both clustering and classification are, in theory, not limited to a fixed number of labels for the data, but this project will be limited to at most four labels for clustering algorithms, and exactly two labels for classification algorithms. Further, the design and development of this project will also assume that the data is 2-dimensional. As such, 3D visualization is currently beyond the scope of DataViLiJ.

As for the GUI interactions, touch screen capabilities are not within the scope of this application.

1.3 Definitions, acronyms, and abbreviations

1) **Algorithm:** In this document, the term 'algorithm' will be used to denote an AI algorithm that can "learn" from some data and assign each data point a label.
2) **Clustering:** A type of AI algorithm that learns to assign labels to instances based purely on the spatial distribution of the data points.
3) **Classification:** A type of AI algorithm that learns to assign new labels to instances based on how older instances were labeled. These algorithms calculate geometric objects that divide the *x*-*y* plane into parts. E.g., if the geometric object is a circle, the two parts are the *inside* and the *outside* of that circle; if the geometric object is a straight-line, then again, there two parts, one on each side of the line.
4) **Framework:** An abstraction in which software providing generic functionality for a broad and common need can be selectively refined by additional user-written code, thus enabling the development of specific applications, or even additional frameworks. In an object-oriented environment, a framework consists of interfaces and abstract and concrete classes.
5) **Graphical User Interface (GUI):** An interface that allows users to interact with the application through visual indicators and controls. A GUI has a less intense learning curve for the user, compared to text-based command line interfaces. Typical controls and indicators include buttons, menus, check boxes, dialogs, etc.
6) **IEEE:** Institute of Electrical and Electronics Engineers, is a professional association founded in 1963. Its objectives are the educational and technical advancement of electrical and electronic engineering, telecommunications, computer engineering and allied disciplines.
7) **Instance:** A 2-dimensional data point comprising a *x*-value and a *y*-value. An instance always has a name, which serves as its unique identifier, but it may be labeled or unlabeled.
8) **Software Design Description (SDD):** A written description of a software product, that a software designer writes in order to give a software development team overall guidance to the architecture of the project.
9) **Software Requirements Specification (SRS):** A description of a software system to be developed. It lays out functional and non-functional requirements and may include a set of use cases that describe user interactions that the software must provide. This document, for example, is a SRS.
10) **Unified Modeling Language (UML):** A general-purpose, developmental modeling language to provide a standard way to visualize the design of a system.
11) **Use Case Diagram:** A UML format that represents the user's interaction with the system and shows the relationship between the user and the different *use cases* in which the user is involved.
12) **User:** Someone who interacts with the DataViLiJ application via its GUI.
13) **User Interface (UI):** See *Graphical User Interface (GUI)*.

1.4 References

[1] IEEE Software Engineering Standards Committee. "IEEE Recommended Practice for Software Requirements Specifications." In *IEEE Std. 830-1998*, pp. 1-40, 1998.

[2] IEEE Software Engineering Standards Committee. “IEEE Standard for Information Technology – Systems Design – Software Design Descriptions.” In *IEEE STD 1016-2009*, pp.1-35, July 20 2009
[3] Rumbaugh, James, Ivar Jacobson, and Grady Booch. *Unified modeling language reference manual, The*. Pearson Higher Education, 2004.
[4] McLaughlin, Brett, Gary Pollice, and David West. *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D*. O’Reilly Media, Inc., 2006.
[5] Riehle, Dirk, and Thomas Gross. “Role model based framework design and integration.” In *ACM SIGPLAN Notices*, Vol. 33, No. 10, pp. 117-133. ACM, 1998.

1.5 Overview

This software design description (SDD), include design components that use UML or specify how to build the DataViLiJ application, the UML provide are the class diagrams which are in section 3 of the SDD and the sequence diagram which are in section 4. Note that all UML diagram are created using VioletUML

2. Package-level Design Viewpoint

This design will encompass the DataViLiJ application. In completing the application, we will heavily rely on Java API, more specifically JavaFX framework to provide a working application. In the next few sections we will provide description of the components needed to be built, as well as how we use the Java API in doing so.

2.1 DataViLiJ Overview

The DataViLiJ application will be design with the ViLiJ framework provided. Figure 2.1 specify all components to be develop placed inside their associated packages.

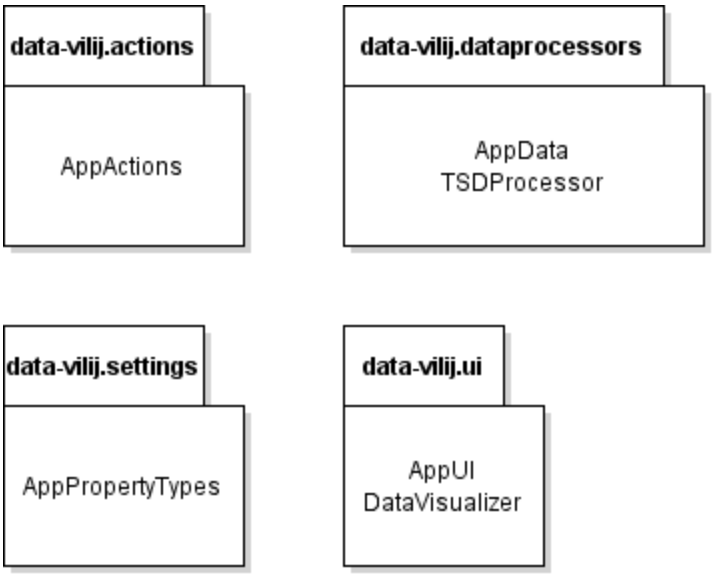
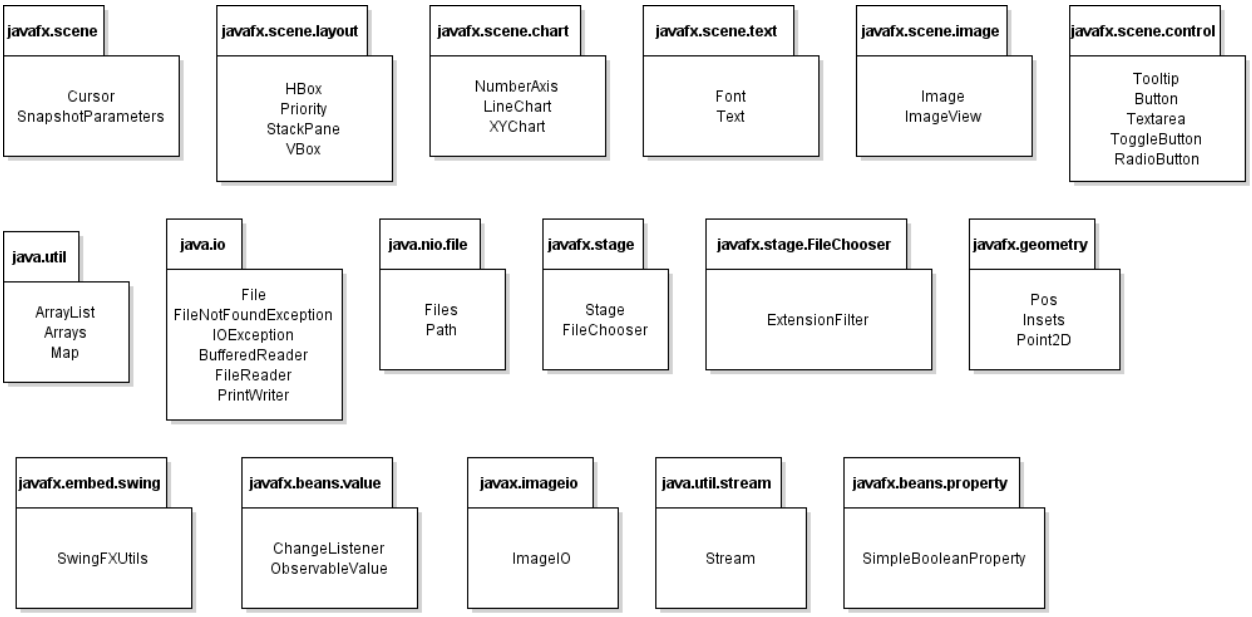


Figure 2.1

2.2 Java API Usage

The DataViLiJ application will be made with Java, hence the usage of Java API is necessary. Figure 2.2 shows all of the Java API that is used.

Figure 2.2 Java API



2.3 Java API Usage Descriptions

Tables 2.3.1-2.3. below summarize how each class will be used.

Table 2.3.1 classes in Java API's javafx.scene

Class/Interface	Usage
SnapshotParameters	For setting rendering attributes for the Chart.

Table 2.3.2 classes in Java API's javafx.scene.layout

Class/Interface	Usage
HBox	For the layout of the detail text of the data.
Priority	For setting the priority of the growth of the HBox and the VBox.
StackPane	For the layout of the whole UI.
VBox	For the layout of nodes on the left of the UI.

Table 2.3.3 classes in Java API's javafx.scene.chart

Class/Interface	Usage
NumberAxis	For creating the chart axis.
LineChart	The chart used to display the data.
XYChart	For creating series of the chart.

Table 2.3.4 classes in Java API's javafx.scene.text

Class/Interface	Usage
Font	For changing font of text.

Text	For creating details of data.

Table 2.3.5 classes in Java API's javafx.scene.image

Class/Interface	Usage
Image	For receiving the screenshot image of the chart.
ImageView	For saving the screenshot image.

Table 2.3.6 classes in Java API's javafx.scene.control

Class/Interface	Usage
Button	For creating the config button, or any other button necessary.
TextArea	For showing the instances of the data.
ToggleButton	For toggling the text area from done to edit or vice versa.
RadioButton	For selection of algorithm types and algorithmss
CheckBox	For the option of continuous run.

Table 2.3.7 classes in Java API's javafx.util

Class/Interface	Usage
ArrayList	For general data traversing.
Arrays	For general data traversing.
Map	For general data traversing.

Table 2.3.8 classes in Java API's java.io

Class/Interface	Usage
File	For writing file.
FileNotFoundException	For throwing exception when file is not founded.
IOException	For throwing IOException of FileChooser.
BufferedReader	For traversing through the loaded data.
FileReader	For reading file of the data that comes from load data.
PrintWriter	For saving data to the path of the data.

Table 2.3.9 classes in Java API's java.nio.file

Class/Interface	Usage
Files	For saving the data through PrintWriter
Path	For tracking the path of data to be save to.

Table 2.3.10 classes in Java API's javafx.stage

Class/Interface	Usage
Stage	For creating the main stage of the application.
FileChooser	For saving or loading data.

Table 2.3.11 classes in Java API's javafx.stage.FileChooser

Class/Interface	Usage
ExtensionFilter	For writing the tsd extension.

Table 2.3.12 classes in Java API's javafx.geometry

Class/Interface	Usage
Pos	For formatting the position of node in the stack pane
Insets	For setting padding between node in panes
Point2D	For saving data point to series.

Table 2.3.13 classes in Java API's javafx.embed.swing

Class/Interface	Usage
SwingFxUtils	For saving the screenshot image.

Table 2.3.14 classes in Java API's javafx.beans.value

Class/Interface	Usage
ChangeListener	For detecting changes of checkbox.
ObservableValue	For detecting changes of text area.

Table 2.3.15 classes in Java API's javax.imageio

Class/Interface	Usage
ImageIO	For saving the screenshot image by writing it to extension.

Table 2.3.16 classes in Java API's java.util.stream

Class/Interface	Usage
Stream	For traversing using lambda.

Table 2.3.17 classes in Java API's javafx.beans.property

Class/Interface	Usage
SimpleBooleanProperty	For creating a Boolean to check if data has any unsaved changes

3. Class-level Design Viewpoint

This design will encompass the DataViLiJ application. The following UML are the class diagram for the application. Due to the complexity of this program we will provide a series of diagram going from the basic overview of classes to the detailed ones that contain methods to be used.

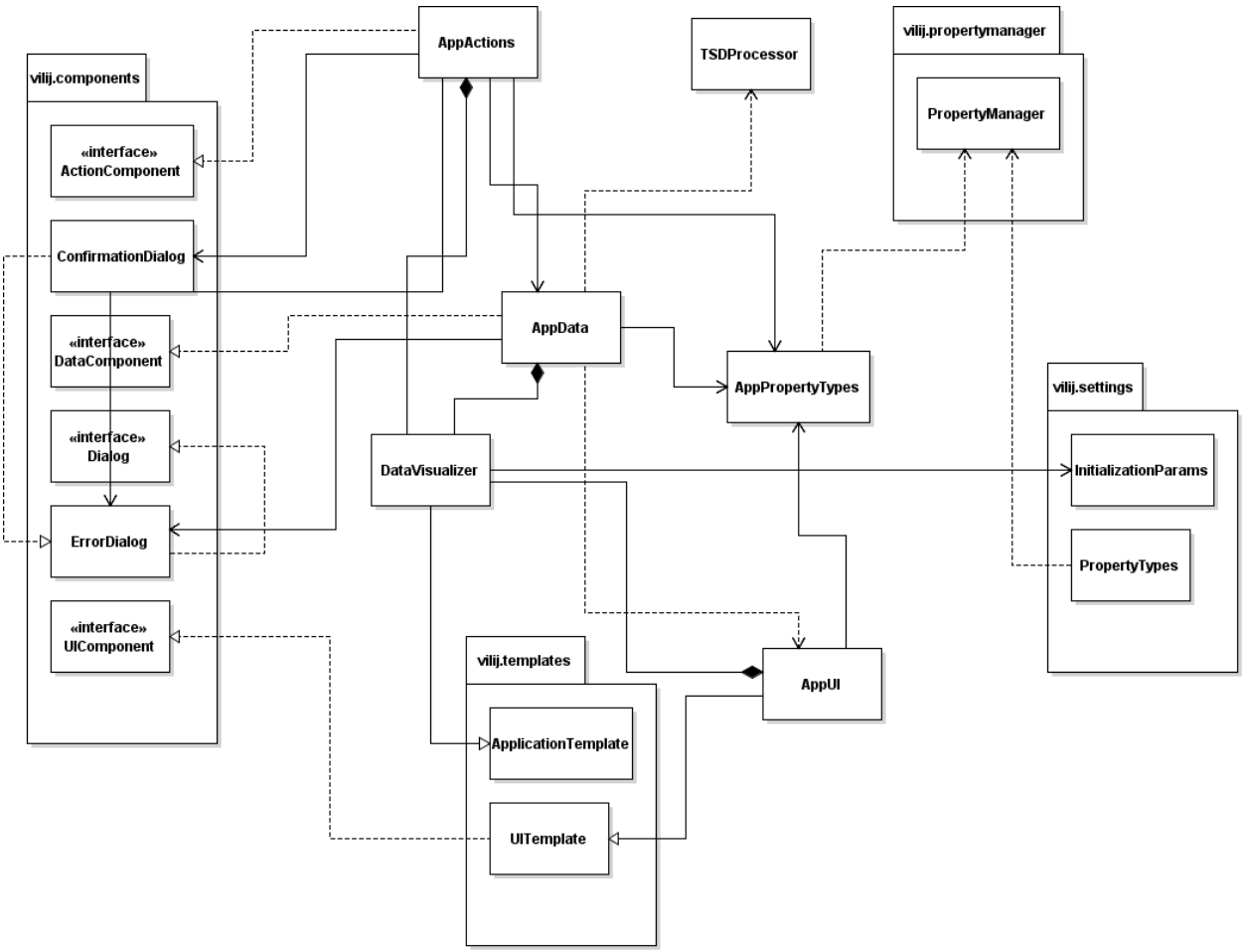


Figure 3.1 DataViLiJ Overview UML class diagram

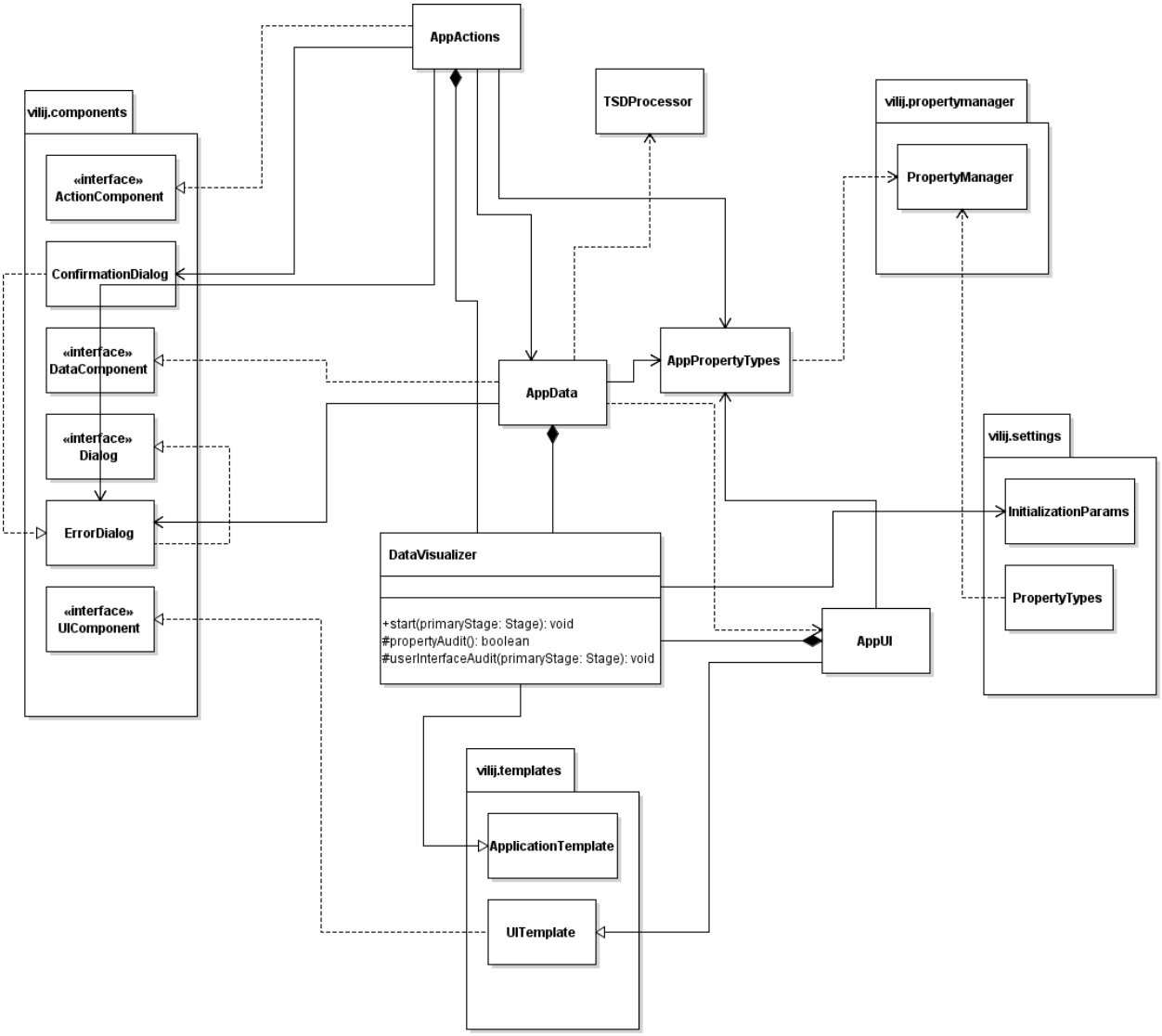


Figure 3.2 DataVisualizer detailed UML class diagram

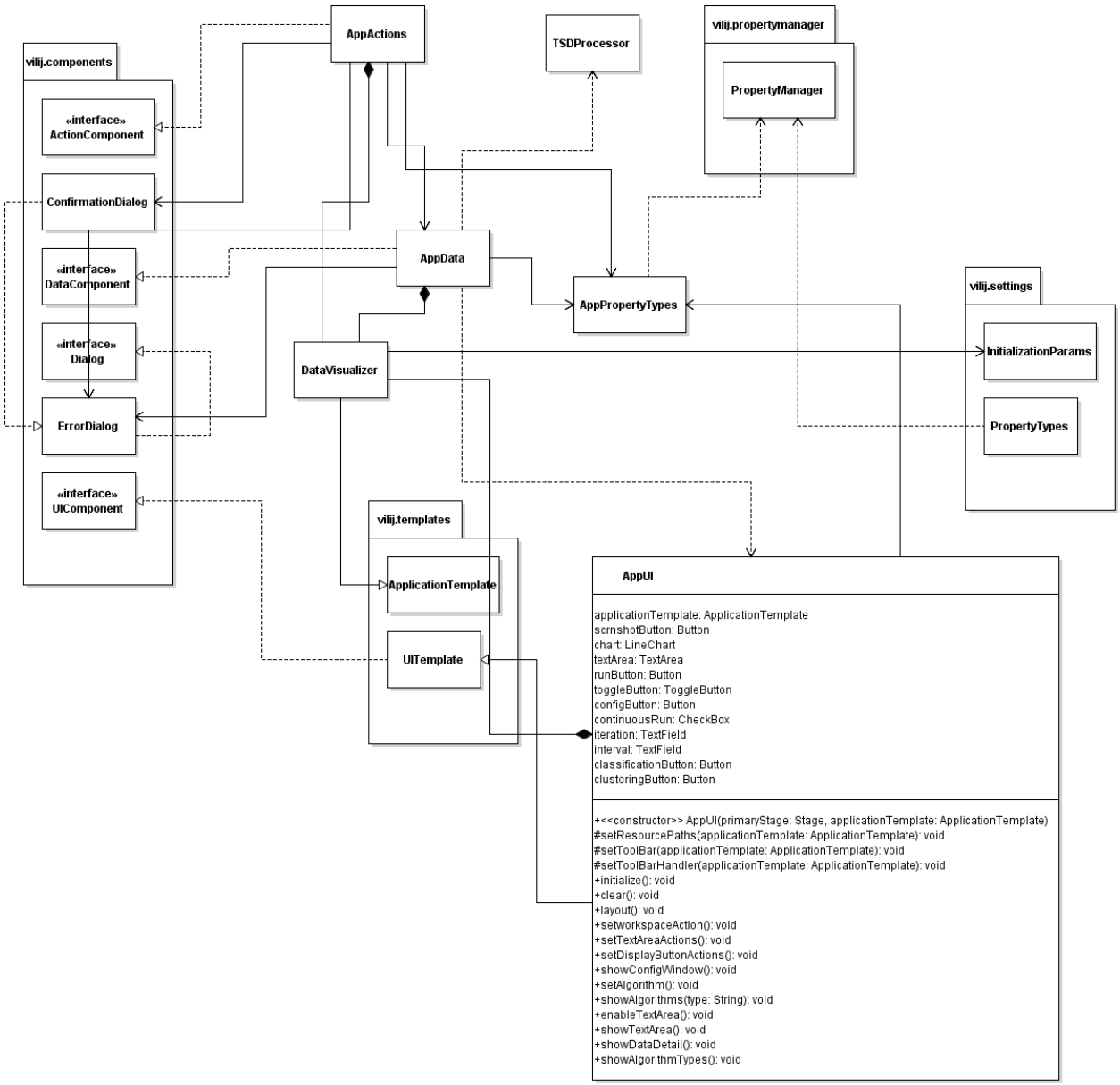


Figure 3.3 AppUI detailed UML class diagram

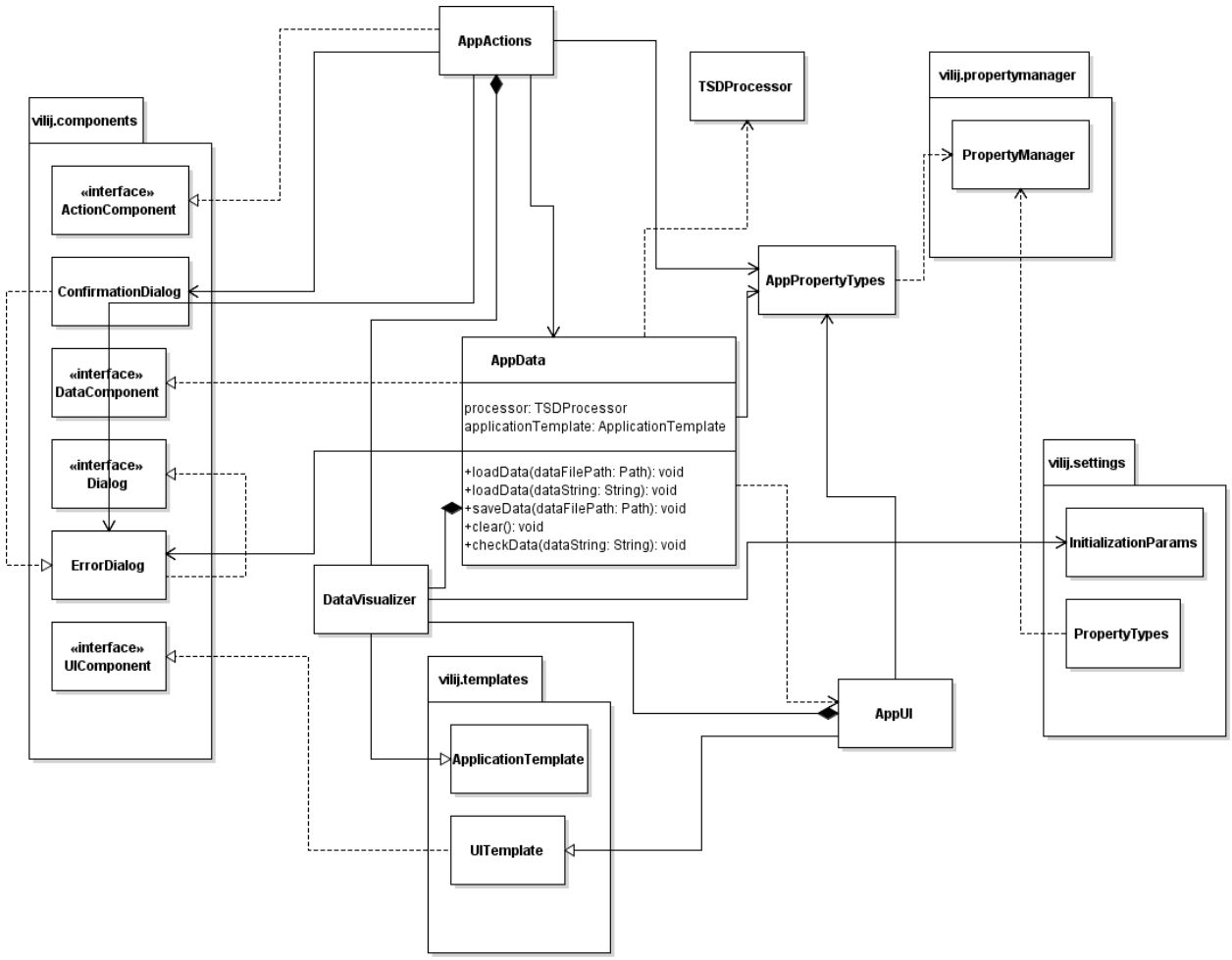


Figure 3.4 AppData detailed UML class diagram

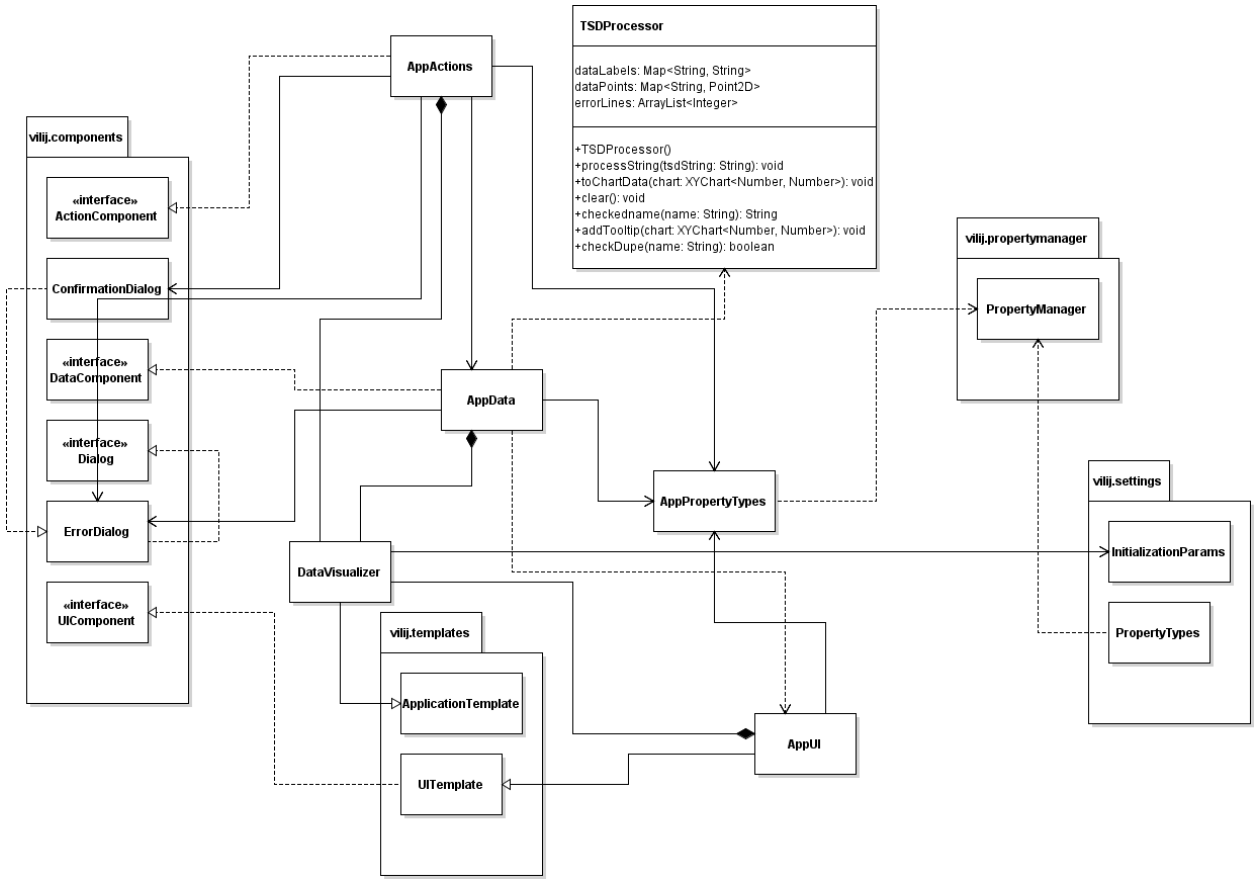


Figure 3.5 TSDProcessor detailed UML class diagram

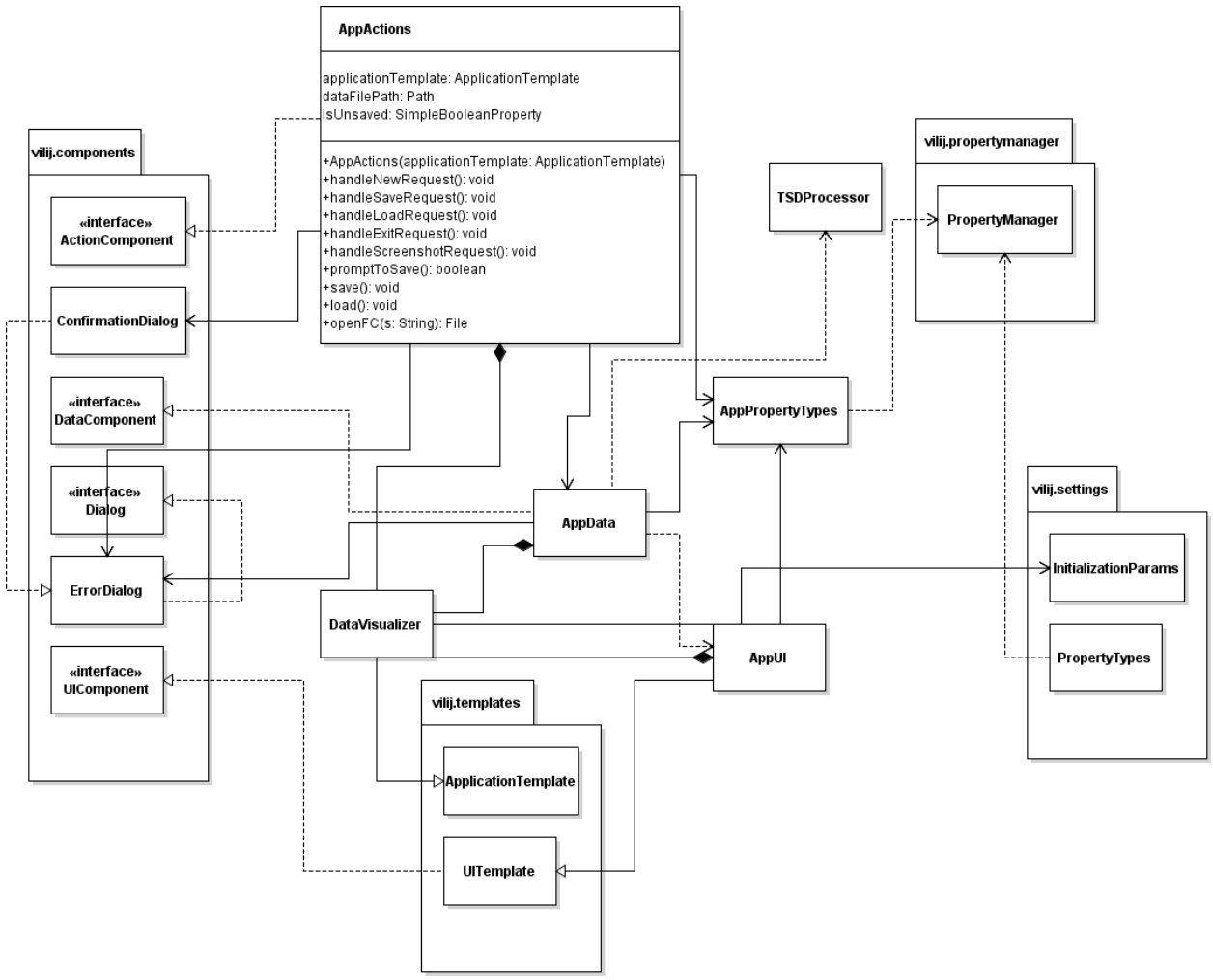


Figure 3.6 AppActions detailed UML class diagram

4. Method-level Design Viewpoint

In this section we will use UML sequence diagrams to describe how the application will response to user. We will cover the appropriate use cases.

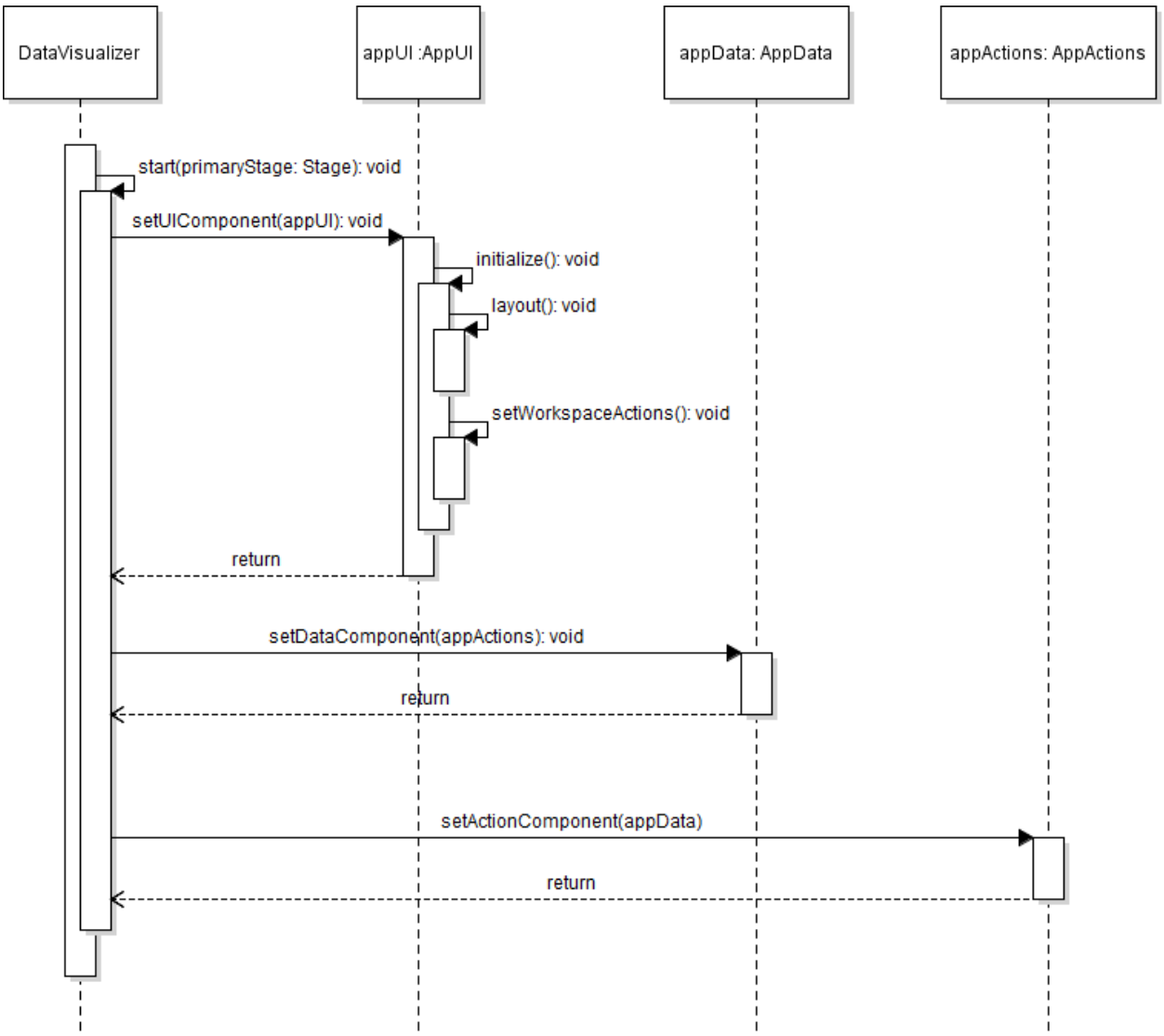


Figure 4.1 Start Application Use, UML Sequence Diagram (use case 1)

Upon user starting the program the start method will run, this will go through each of the components and layout the GUI of this application.

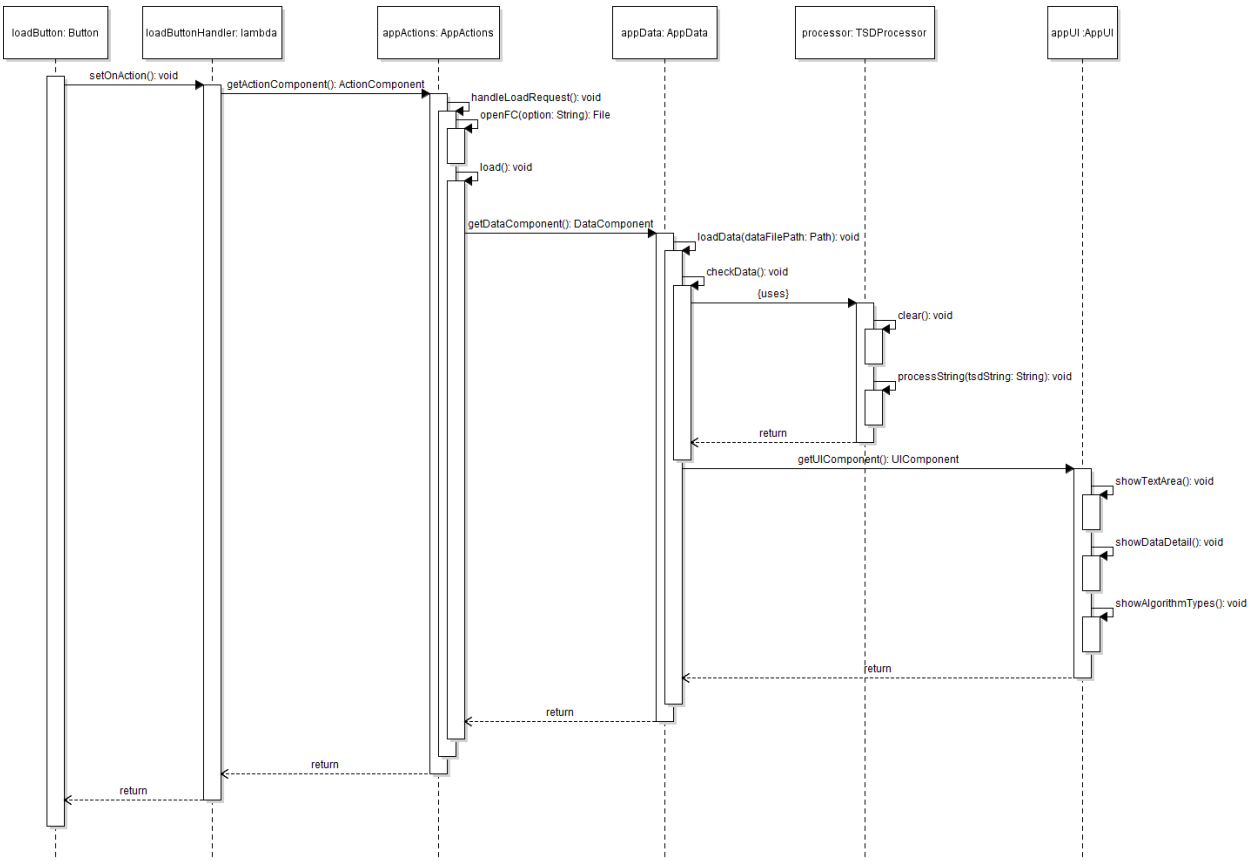


Figure 4.2.1 Load Data Part 1, UML Sequence Diagram (use case 2)

Upon clicking loadButton user then are granted to choose a tsd file, if said tsd file is formatted correctly with no error, the textarea will show up, follow with detail of the data and the algorithm types.

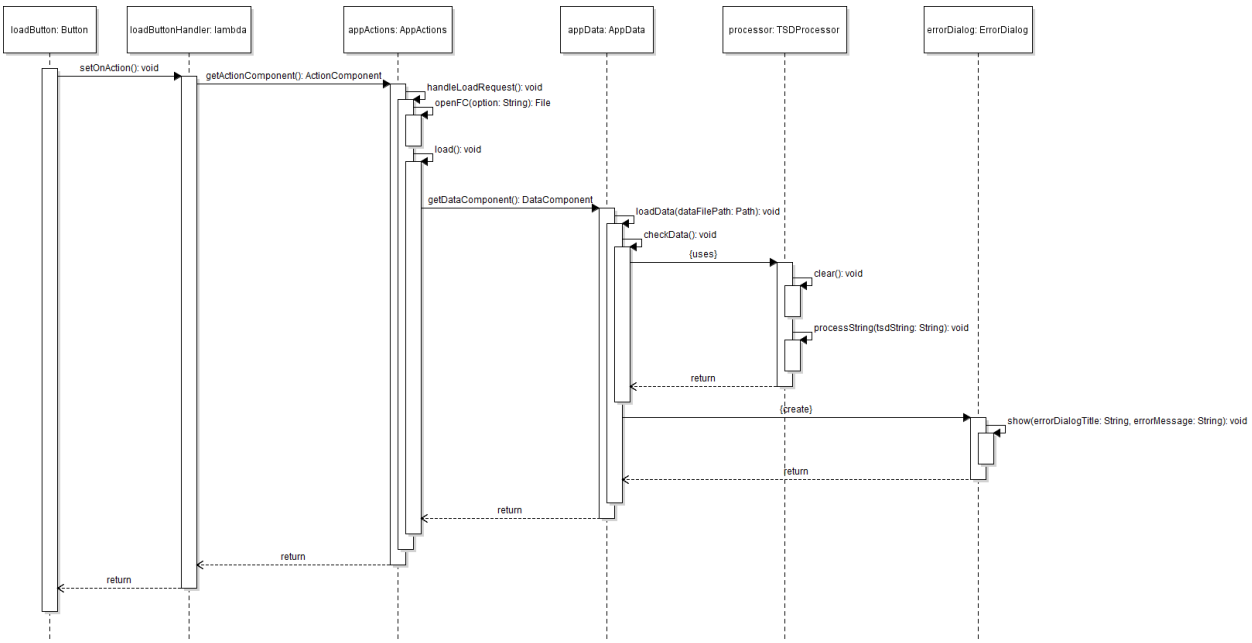


Figure 4.2.2 Load Data Part 2, UML Sequence Diagram (use case 2)

Upon selecting the tsd file, if the data is invalid an error dialog will pop up informing the user.

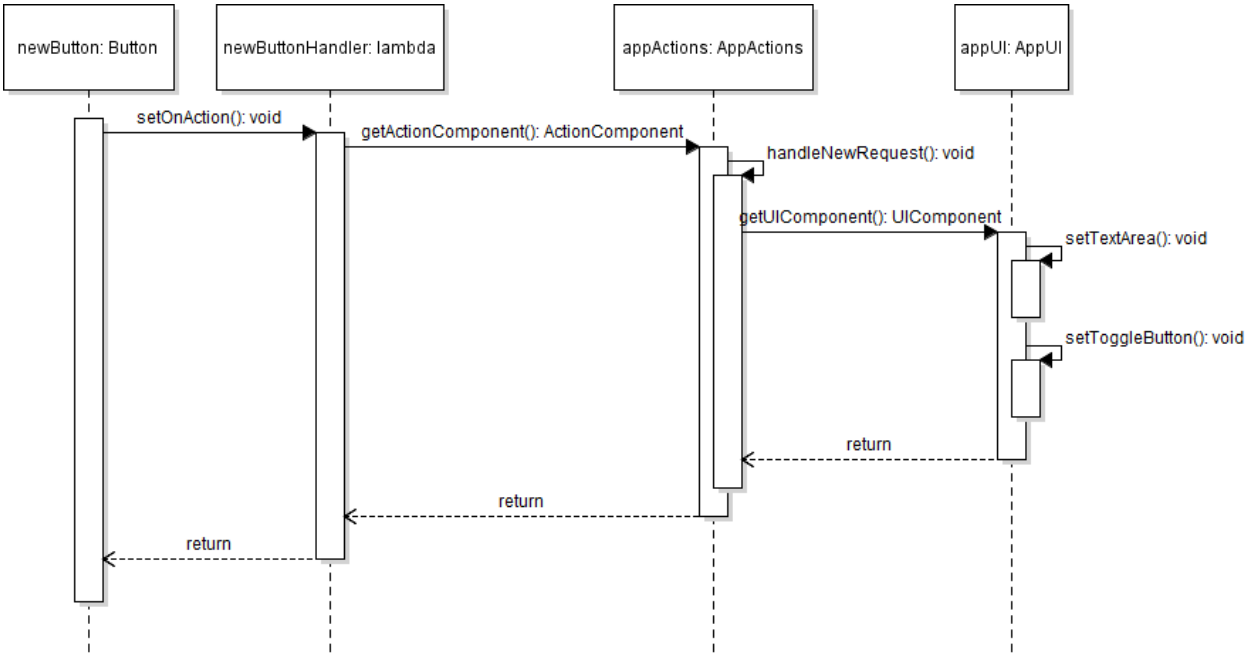


Figure 4.3.1 Create New Data, UML Sequence Diagram (use case 3)

Upon clicking newbutton the GUI will display an editable textarea for new data, along with a toggleButton the have states of Done and Edit.

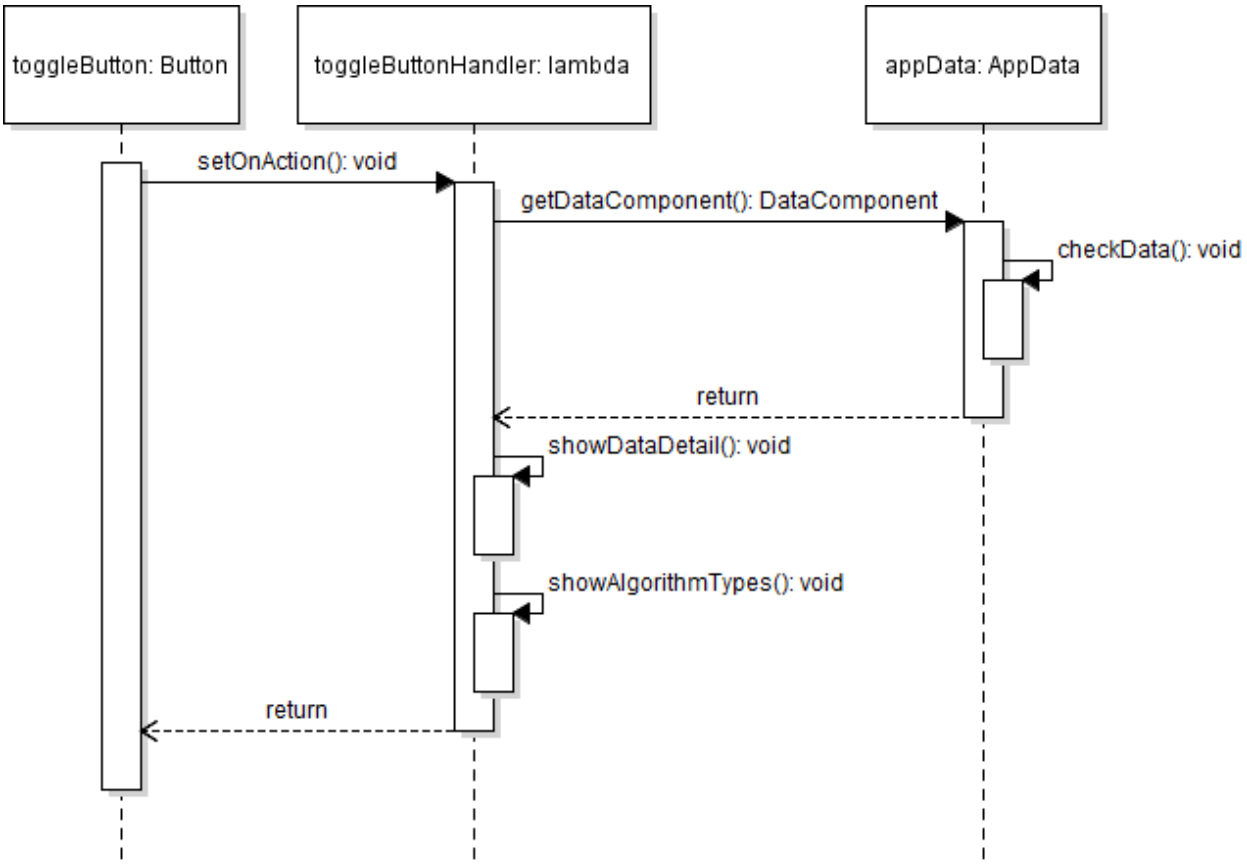


Figure 4.3.2 Done Switch Part 1, UML Sequence Diagram (use case 3)

Upon clicking Done on the toggle Switch, if data is valid the detail of data and the algorithm types will show up in GUI.

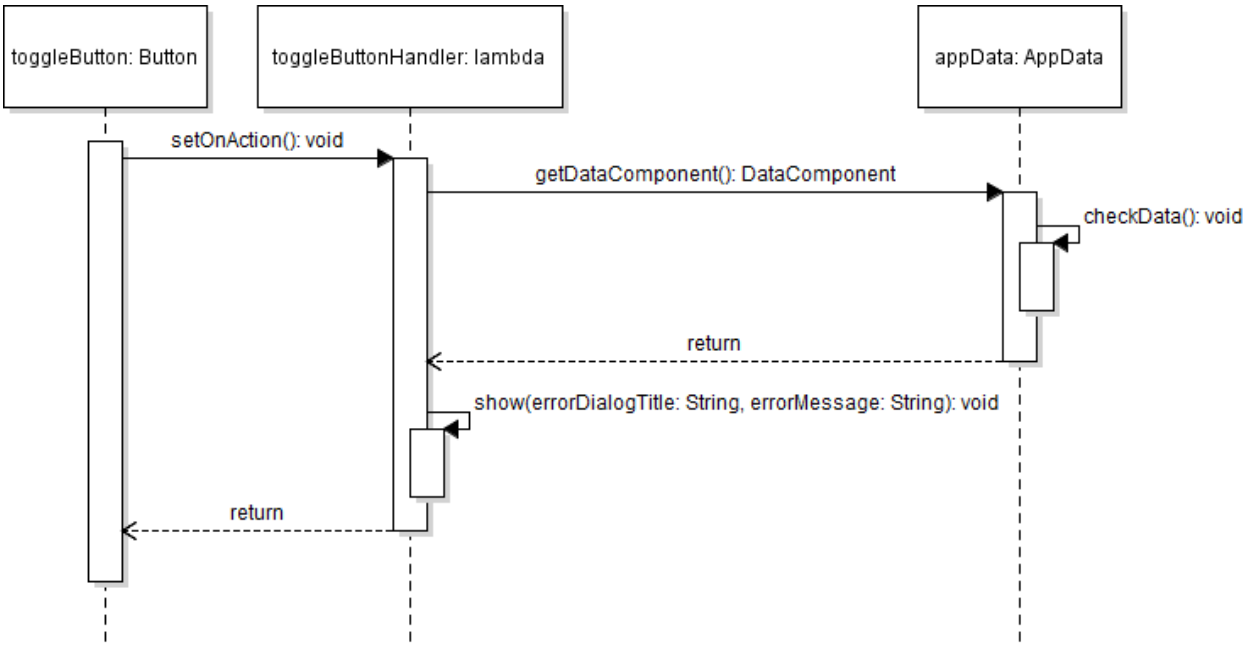


Figure 4.3.3 Done Switch Part 2, UML Sequence Diagram (use case 3)

Upon clicking Done on the toggle Switch, if data is invalid, an error dialog will show up.

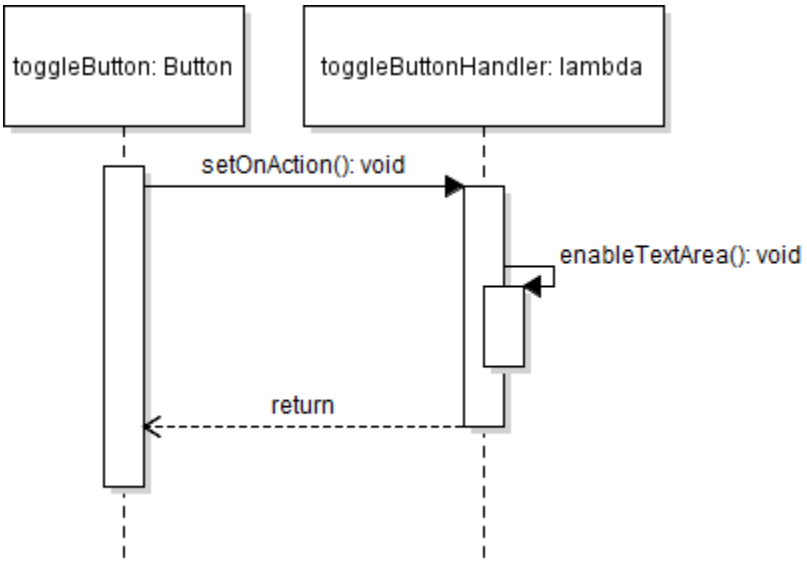


Figure 4.3.4 Edit Switch, UML Sequence Diagram (use case 3)

Upon clicking Edit on the toggle Switch, the textArea will become editable.

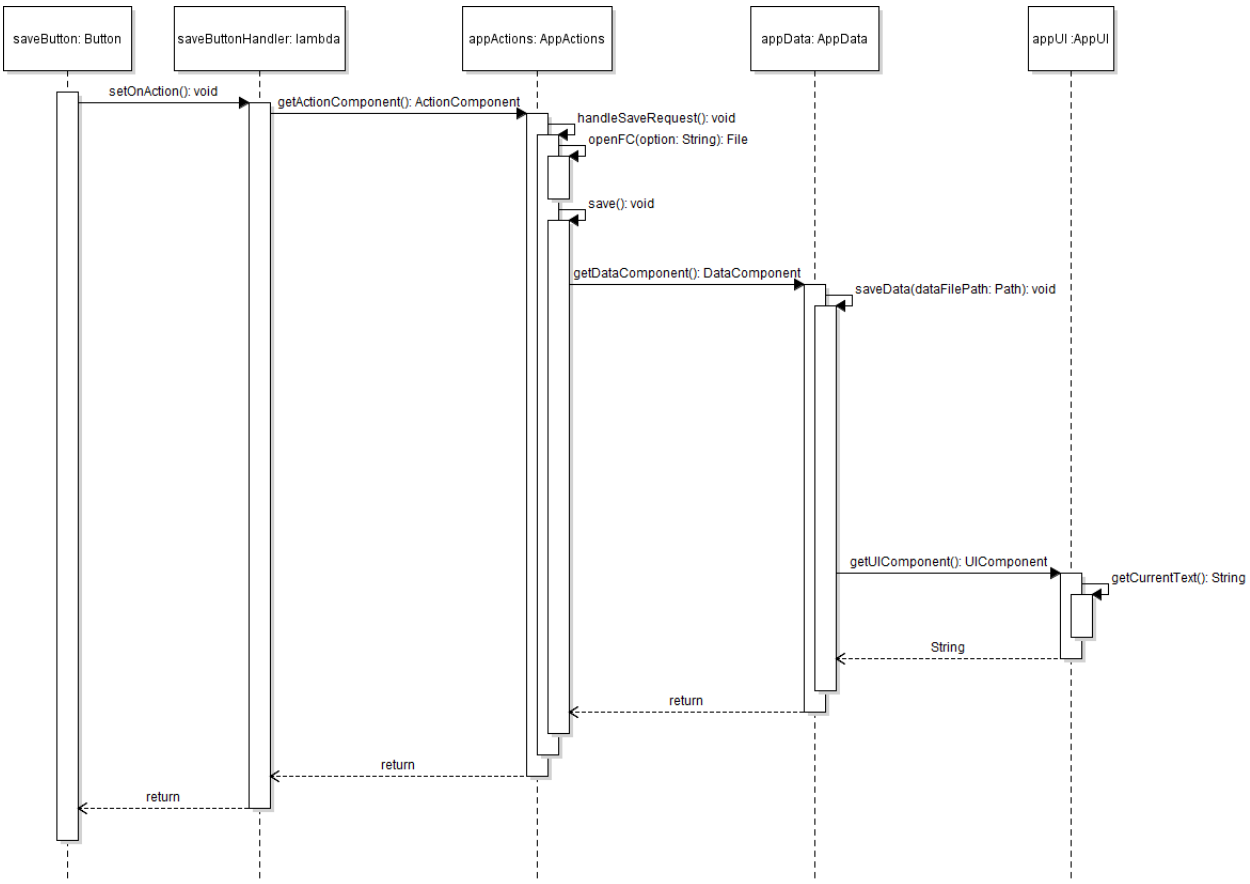


Figure 4.4.1 Save Data, UML Sequence Diagram (use case 4)

Upon clicking Save Button, a filechooser will show up allow the user to save the data as a tsd file.

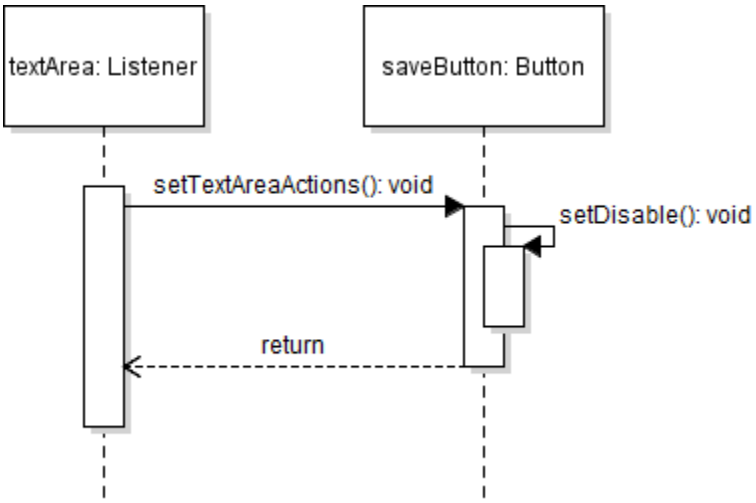


Figure 4.4.2 Save Button Disable Condition, UML Sequence Diagram (use case 4)

The save button will be disable if it is empty or there were no changes, this will be determined by the textArea's changeListener.

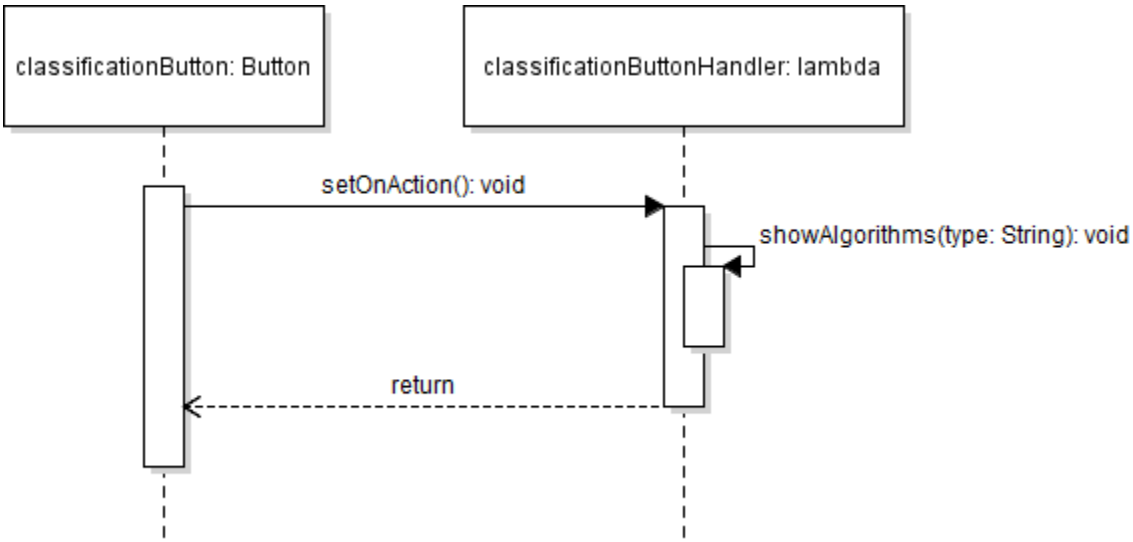


Figure 4.5.1 Selecting Classification Type, UML Sequence Diagram (use case 5)

Upon choosing the classification algorithm type, the action event will be run and the algorithms with the classification type will show.

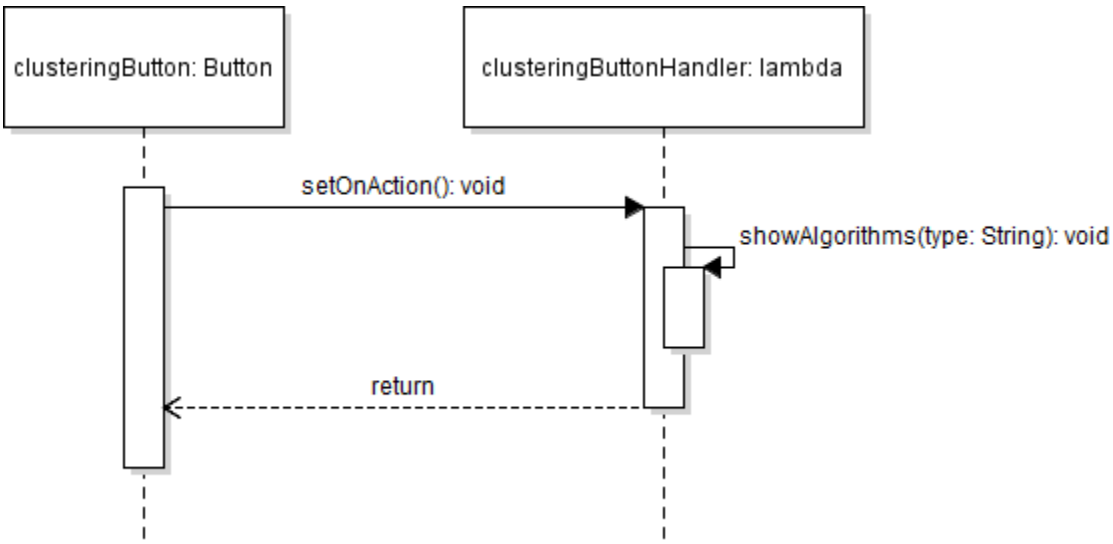


Figure 4.5.2 Selecting Clustering Type, UML Sequence Diagram (use case 5)

Upon choosing the clustering algorithm type, the action event will be run and the algorithms with the clustering type will show.

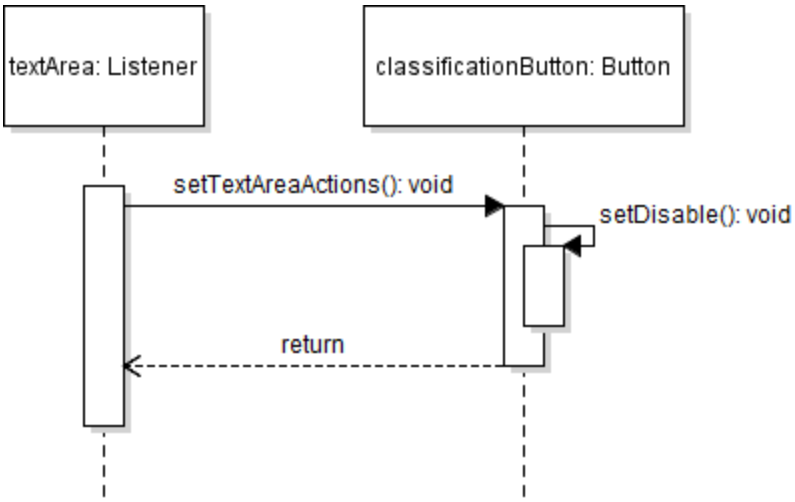


Figure 4.5.3 classificationButton Disable Condition, UML Sequence Diagram (use case 5)

The classificationButton will be disable if there are not exactly two labels from data.

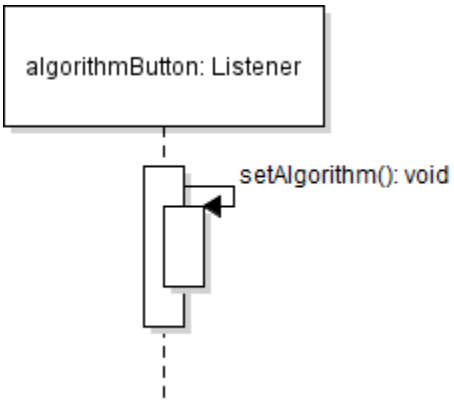


Figure 4.6 Selecting Algorithm, UML Sequence Diagram (use case 6)

Upon selecting a radio button, an algorithm will be selected, that can be run using the runButton that is further explain in use case 7.

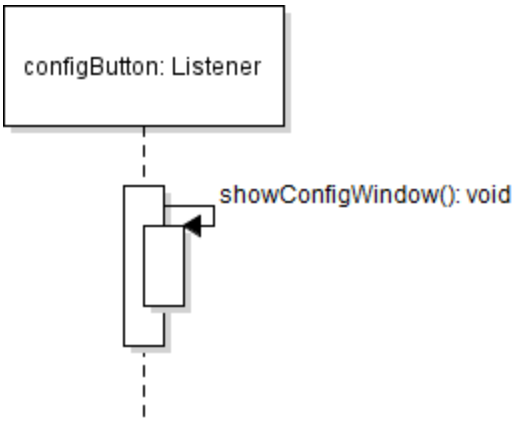


Figure 4.7 Configuration Window, UML Sequence Diagram (use case 7)

Upon click on the configButton a new window will pop up asking for the users input on iteration, interval, and if the program should continuously run till stop or not.

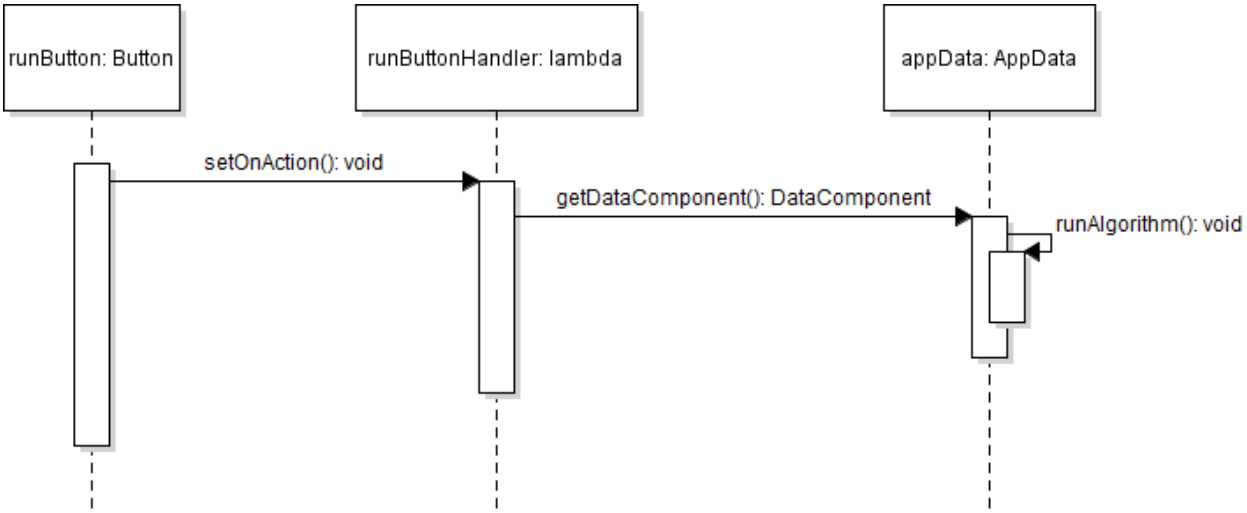


Figure 4.8 Run Algorithm, UML Sequence Diagram (use case 8)

Upon click on the run button, the runAlgorithm() method will run in AppData and in there its going to determine the intervals, iteration, as well as the should it continuously run or not.

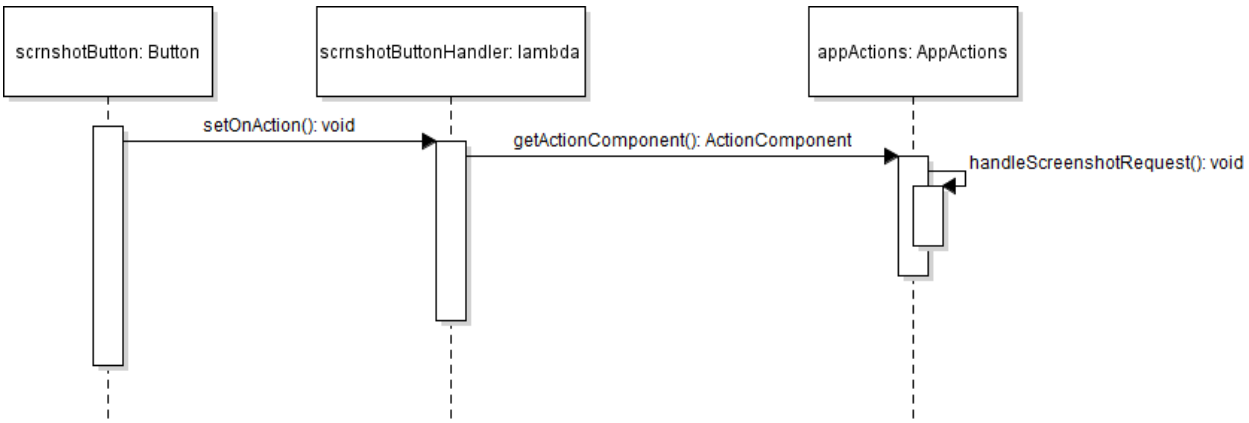


Figure 4.9 Take Screenshot, UML Sequence Diagram (use case 9)

Upon click the screenshot button on the toolbar, the screenshot of the chart will be taken, and a filechooser will open allowing the user to save the image.

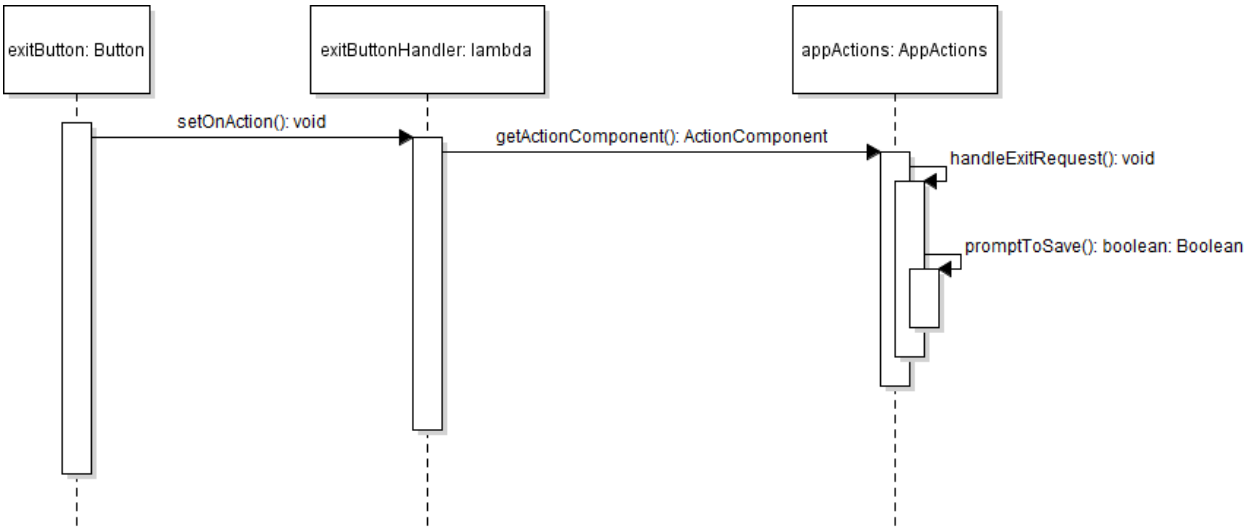


Figure 4.10 Take Screenshot, UML Sequence Diagram (use case 10)

Upon click exit button, a prompt to save window will popup asking the user to save the data or not.

5. File Structure and Formats

Note that the framework of the DataViLiJ application is already provide, as well as the propertiesManagers. Figure 5.1 specifies the necessary file structure the application use in order for it to launch.

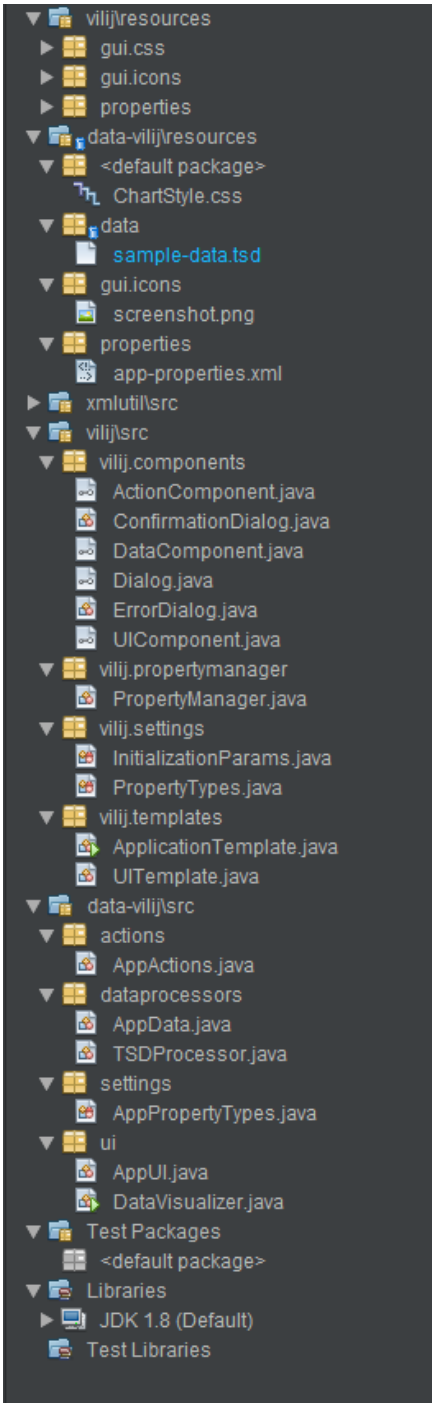


Figure 5.1 Structure of application.

```
<property name="DATA_RESOURCE_PATH" value="data-vilij/resources/data"/>

<!-- USER INTERFACE ICON FILES -->
<property name="SCREENSHOT_ICON" value="screenshot.png"/>

<!-- TOOLTIPS FOR BUTTONS -->
<property name="SCREENSHOT_TOOLTIP" value="Screenshot"/> <!-- will print current view of image

<!-- WARNING MESSAGES -->
<property name="EXIT_WHILE_RUNNING_WARNING"
          value="An algorithm is running. If you exit now, all unsaved changes will be lost. A

<!-- ERROR MESSAGES -->
<property name="RESOURCE_SUBDIR_NOT_FOUND" value="Directory not found under resources."/>

<!-- APPLICATION-SPECIFIC MESSAGE TITLES -->
<property name="SAVE_UNSAVED_WORK_TITLE" value="Save Current Work"/>

<!-- APPLICATION-SPECIFIC MESSAGES -->
<property name="SAVE_UNSAVED_WORK" value="Would you like to save current work?"/>

<!-- APPLICATION-SPECIFIC PARAMETERS -->
<property name="DATA_FILE_EXT" value="*.tsd"/>
<property name="DATA_FILE_EXT_DESC" value="Tab-Separated Data File"/>
<property name="TEXT_AREA" value="text area"/>
```

Figure 5.2 app-properties.xml format

Note: This is just a look of what the xml file should look like.

6. Supporting Information

Note that the algorithm will be provided by the instructors.