DATAViLiJ™
Software Design Description

Author: Sudhin Domala

Abstract: This document describes the software design for DataViLiJ, that will be a desktop 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.

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

Table of Contents

1 Introduction

This is the Software Design Description (SDD) for the DataViLi™ application. Note that this document format is based on the IEEE Std 830TM-1998 (R2009) recommendation for software design.

1.1 Purpose

The purpose of this document is to specify how the DataViLiJ application should look and operate. This document serves as an agreement among all parties and as a reference for how the data visualization application should ultimately be constructed. Upon reading of this document, one should clearly understand the visual aesthetics and functionalities of application's user interface as well as understand the way AI algorithms are incorporated..

1.2 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.

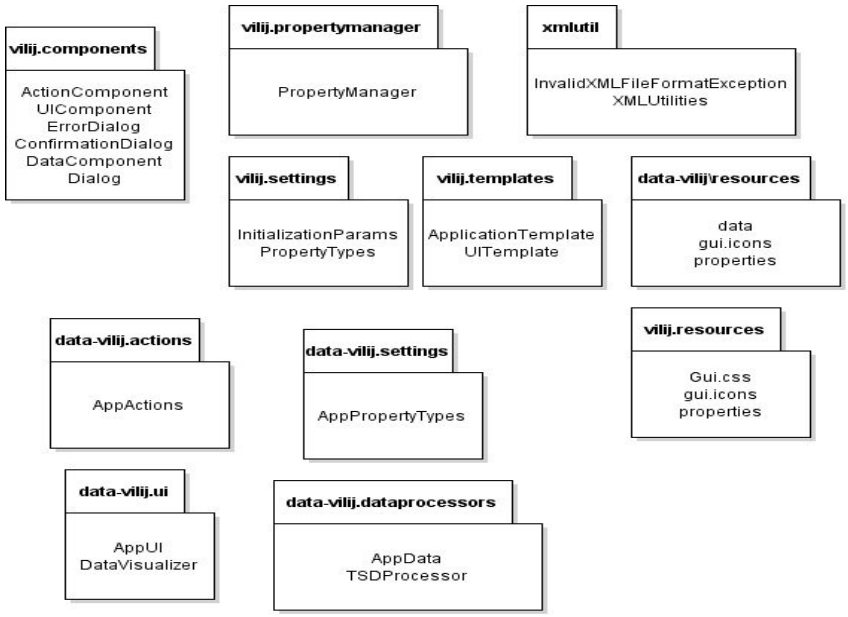
1.5 Overview

This software requirements specification (SRS) document will clearly define the operational capabilities of DataViLiJ and its UI functionalities and aesthetics. Note that this document is not a software design description (SDD), and it does not include design components that use UML or specify how to build the appropriate technologies. It is simply an agreement concerning what to build. The remainder of this document consists of two chapters, appendices, and the index. Chapter 2 provides the general factors affecting the application requirements. It also provides the background for these requirements, putting the application in perspective with other related applications such as the algorithms that will be used. Chapter 3 provides the software requirements to a level of detail sufficient to allow the design of the application to satisfy those requirements. This includes the GUI and the core operations. Here, every input into the application and every output from the application, along with all the functions, are described. The appendices include sample data that illustrates the data format,and background information on the type of algorithms.

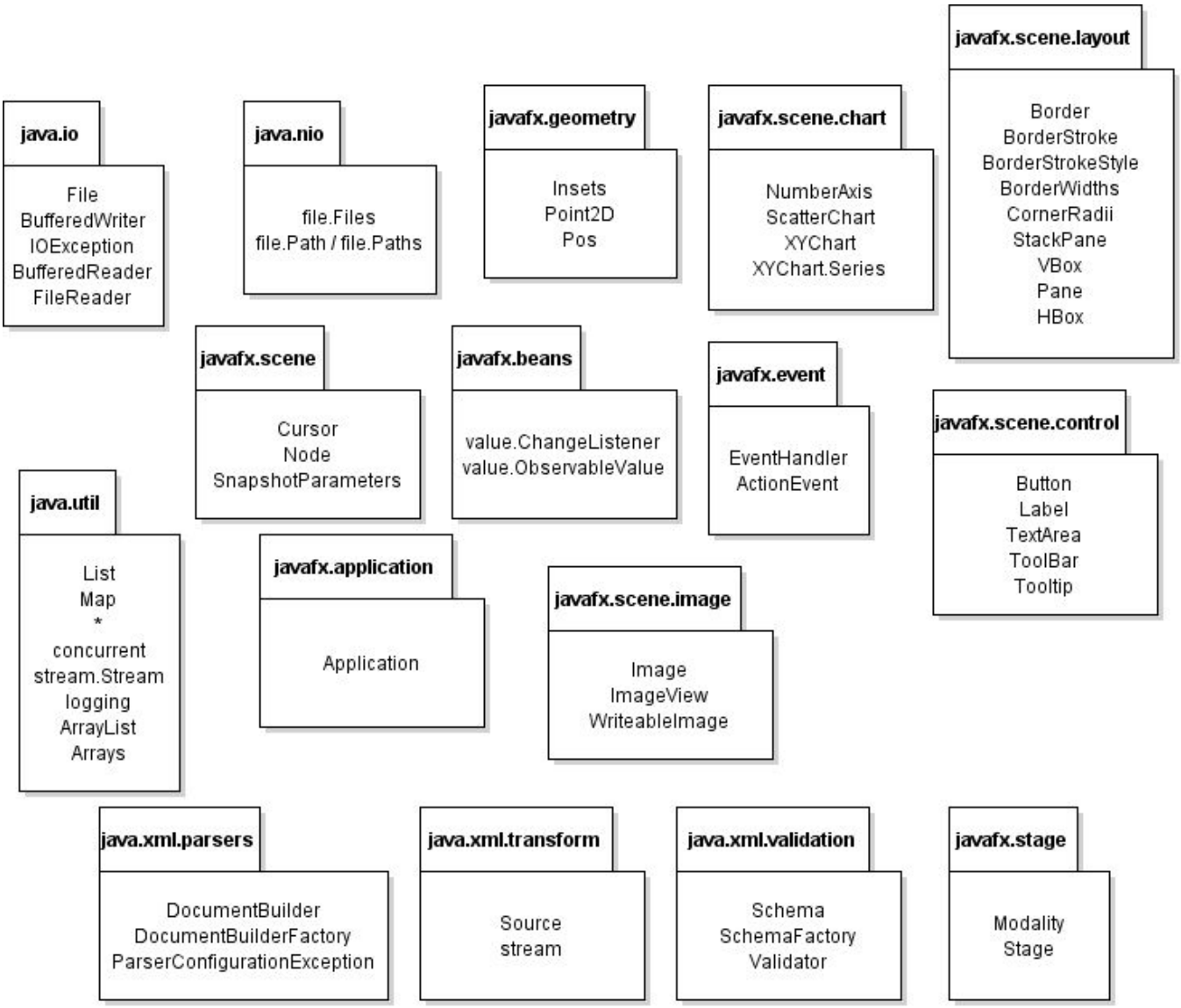
2 Package-Level Design Viewpoint

As mentioned, this design will encompass both the DataViLiJ application and the Desktop Java 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 Software Overview



2.1 Java API Usage



2.3 Java API Usage Descriptions

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

Class/Interface	Use
Cursor	A class to encapsulate the bitmap representation of the mouse cursor.
Node	A scene graph is a set of tree data structures where every item has zero or one parent, and each item is either a "leaf" with zero sub-items or a "branch" with zero or more sub-items.
SnapshotParameters	Parameters used to specify the rendering attributes for Node snapshot.

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

Class/Interface	Use

List	A `List` is an ordered collection of objects, represented by the `java.util.List` interface.
Map	For responding to an action event, like a button press. We will provide our own custom implementation of this interface.
*****	For getting information about a key event, like which key was pressed.
concurrent	For responding to a key event, like a key press. We will provide our own custom implementation of this interface.
stream	For getting information about a mouse event, like where was the mouse pressed?
logging	For responding to a mouse event, like a mouse button press. We will provide our own custom implementation of this interface.
ArrayList	A list of array elements with properites beyond normal array
Arrays	A set of indexed elements that is a data structure to handle values

Table 2.3: Uses for classes in Java API's java.io

Class/Interface	Use
File	Enables you to use work with groups of objects.
BufferedWriter	Let's the JVM know which language to use based on the geographical region
IOException	Used to associate a key with a specific value. HashMaps are efficient for locating a value based on a key and inserting or deleting. An example would be the officeHoursGrid that has the when TAs have office hours.
BufferedReader	Can be used to traverse through a collection of objects
FileReader	Used for a unique collection of objects that are unsorted. Example,

Table 2.4: Uses for classes in Java API's javafx.geometry

Class/Interface	Use
insets	Contains static methods to create JSON readers, writers, builders, and their factory objects.

Point2D	Represent data types for values in JSON data as a Json Object.
Pos	Reads JSON data from a stream and creates an object model in memory.

Table 2.5: Used for class in Java API's javafx.beans.value

Class/Interface	Use
ChangeListener	An event which indicates that a keystroke occurred in a node. Generated when a key is pressed, released, or typed.
ObservableValue	This class represents a key combination in which the main key is specified by its KeyCode.

Table 2.6: Used for Java API's javafx.scene.image

Class/Interface	Use
Image	BorderPane lays out children in top, left, right, bottom, and center positions.
Imageview	Lays out its children within a flexible grid of rows and columns.
WriteableImage	A editable image processed out for ui purposes

Table 2.7: Used for javafx.scene.layout

Class/Interface	Use
Border	A button control that can contain text and/or a graphic.
BorderStroke	Lays out its children within a flexible grid of rows and columns. When a button is pressed and released an ActionEvent is triggered
BorderStrokeStyle	Label is a non-editable text control.
BorderWidths	A HBox lays out its children in a single horizontal row.
CornerRadii	A VBox lays out its children in a single vertical column.
StackPane	A control that has two or more sides, each separated by a divider, which can be dragged by the user to give more space to one of the sides, resulting in the other side shrinking by an equal amount.

VBox	A VBox lays out its children in a single vertical column.
Pane	For the canvas (center) of the workspace.
HBox	A HBox lays out its children in a single horizontal row.

Table 2.8: Used for Java API's javafx.scene.control

Class/Interface	Use
Button	It provides the implementation of the property interface while wrapping it as a string
Label	A button control that can contain text and/or a graphic.
TextArea	Label is a non-editable text control.
ToolBar	Lays out its children within a flexible grid of rows and columns. When a button is pressed and released an ActionEvent is triggered
ToolTip	For the canvas (center) of the workspace.

Table 2.9: Used for Java API's javafx.event

Class/Interface	Use
EventHandler	For taking action when a certain action occurs
ActionEvent	For getting info about an action event

Figure 2.10: Used for Java API's javafx.scene.chart

Class/Interface	Use
NumberAxis	attempts to match the entire input sequence against the pattern.
Scatter Chart	Compiles a representation of a regular expression. Matcher and pattern are used to check validity of user input
XYChart	A compiled x-y axis driven data structure that the user can declare and graph

XYChart.Series	A multiple of compiled x-y axis driven data structure that the user can declare and graph

Figure 2.11: Used for Java API's javafx.application

Class/Interface	Use
Application	Launches the application

Figure 2.12: Used for Java API's java.io

Class/Interface	Use
Files	Access to the other files and searches are possible with this
Path	A object that lets you set the end data placement via a path or initialization

Figure 2.13: Used for Java API's java.xml.parsers

Class/Interface	Use
DocumentBuilder	A java template/ structure to build txt files of said printable data
DocumentBuilderFactory	A automated version of documentbuilder
ParserConfigurationException	A exception handler for xml compiling issues

Figure 2.14: Used for Java API's java.xml.transform

Class/Interface	Use
Source	Manipulate of the xml placement
stream	Concurrent the change of the final destination of said xml file

Figure 2.15: Used for Java API's java.xml.validation

Class/Interface	Use

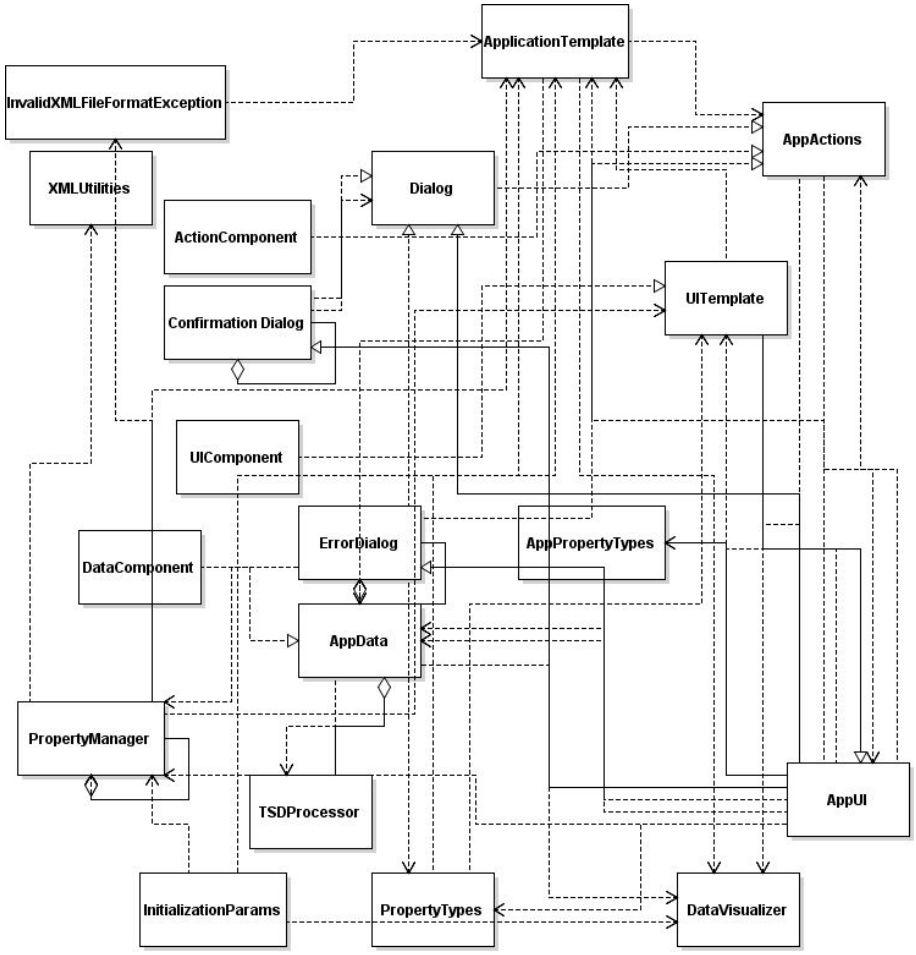
Schema	A class that lets you comply validation of layout
SchemaFactory	A generator of multiple or duplicated schemas
Validator	A automated check of sad xml file

Figure 2.16: Used for Java API's java.stage

Class/Interface	Use
Modality	Launches the application
Stage	Launches the first ui boxed layout

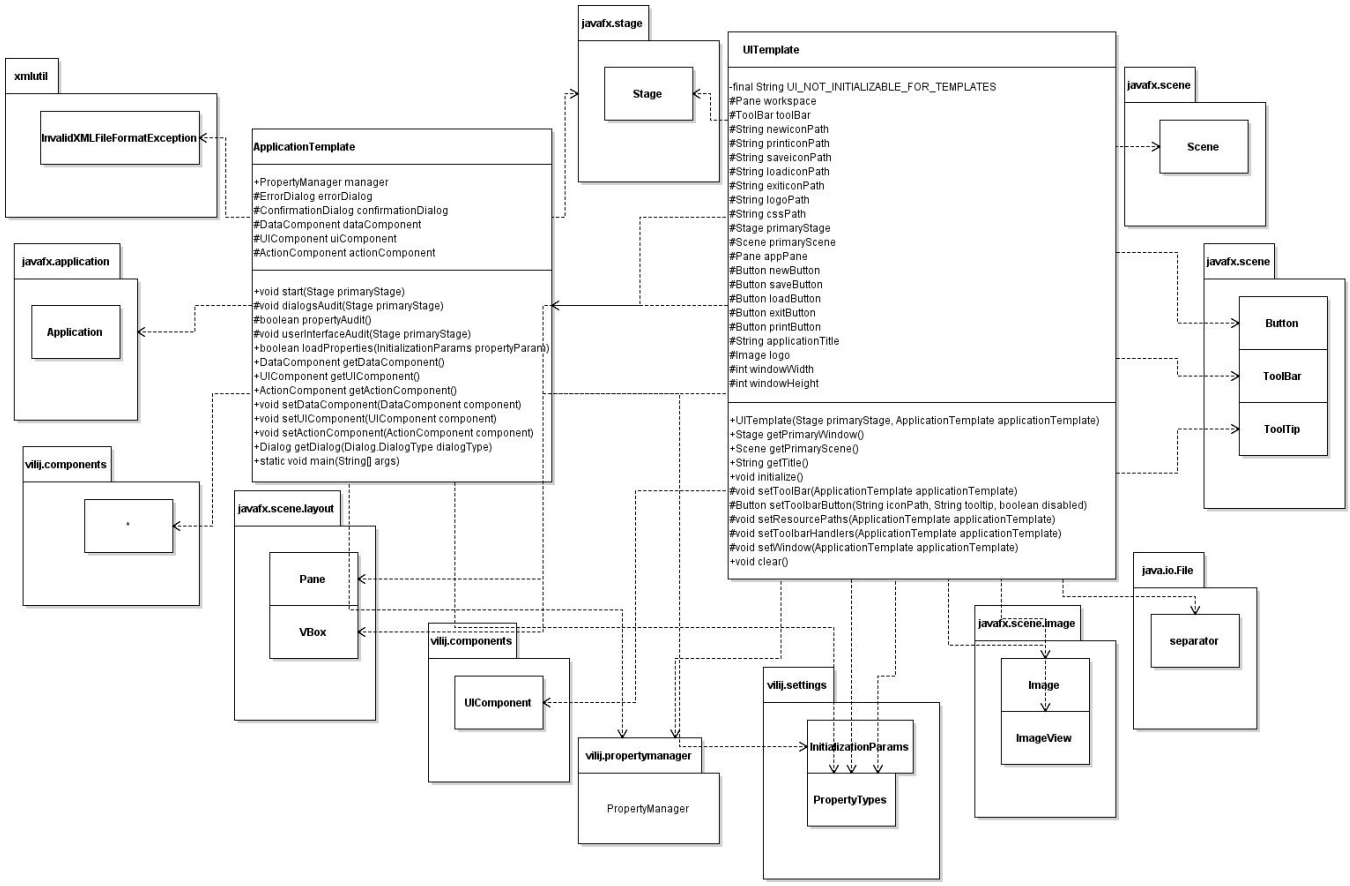
3 Class-Level Design ViewPoint

Figure 3.1 Desktop Java Framework Overview UML Class Diagram



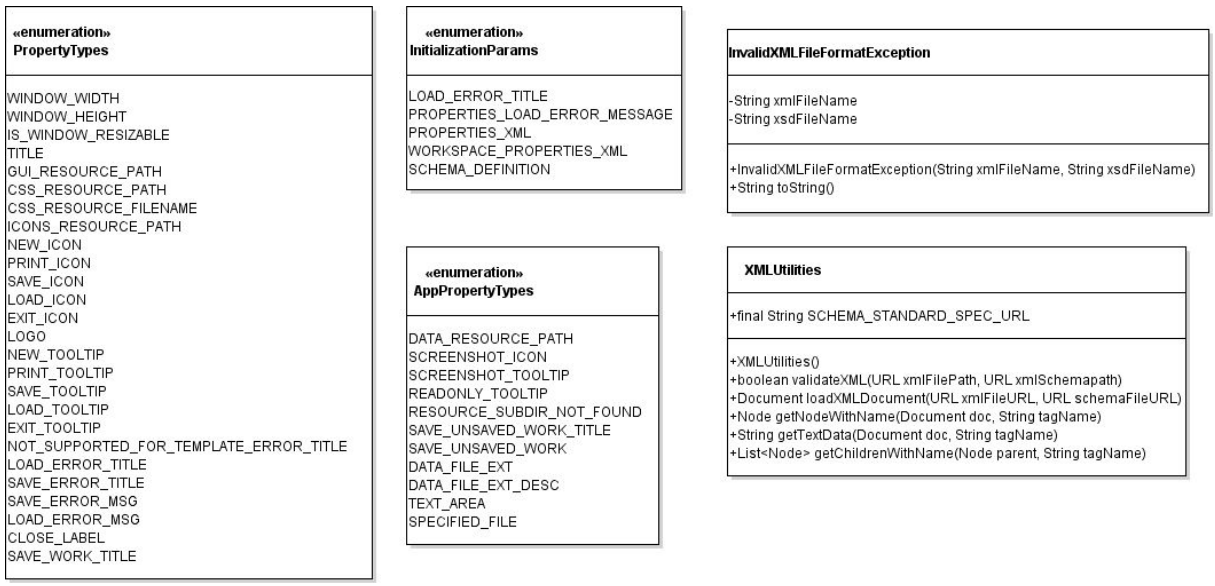
Note: This is how all classed interact with each other. For more details, please go to the UML Class Diagram for that class.

Figure 3.2 Detailed ApplicationTemplate & UITemplate UML class diagram



Note: ApplicationTemplate takes and organizes Dialog, Component, and Data Properties of the loading procedure. The UITemplate contains the paths and declarations of said UI aspect (buttons) and comply with the ApplicationTemplate to handle a button and a response in the DataVilij module.

Figure 3.3 detailed "Enums & XML" UML class diagrams



Note: This subsystem contains property types, enums and other backend data (XML,icons) from the data-vilij\src.settings , vilij.settings, xmlutil packages. These give declarations, set-icons, parameters to declare buttons, make XML, and mainly initialize the compilation of the UI's selected button functions

Figure 3.4 Detailed vilij.components class diagram

Note: The contains the Action, Data, and UI component on DataViLiJ that partakes in the handling of button functions and the methods to which a path of the inputted data to be displayed, saved, algo compiled depending on the user. The Dialogs (including Error,Confirmation) are exception handling if algos don't correspond with inputted data or inputted data can't be loaded or is erroneous in the button functions.These are in commication with AppActions, AppUI, and AppData.

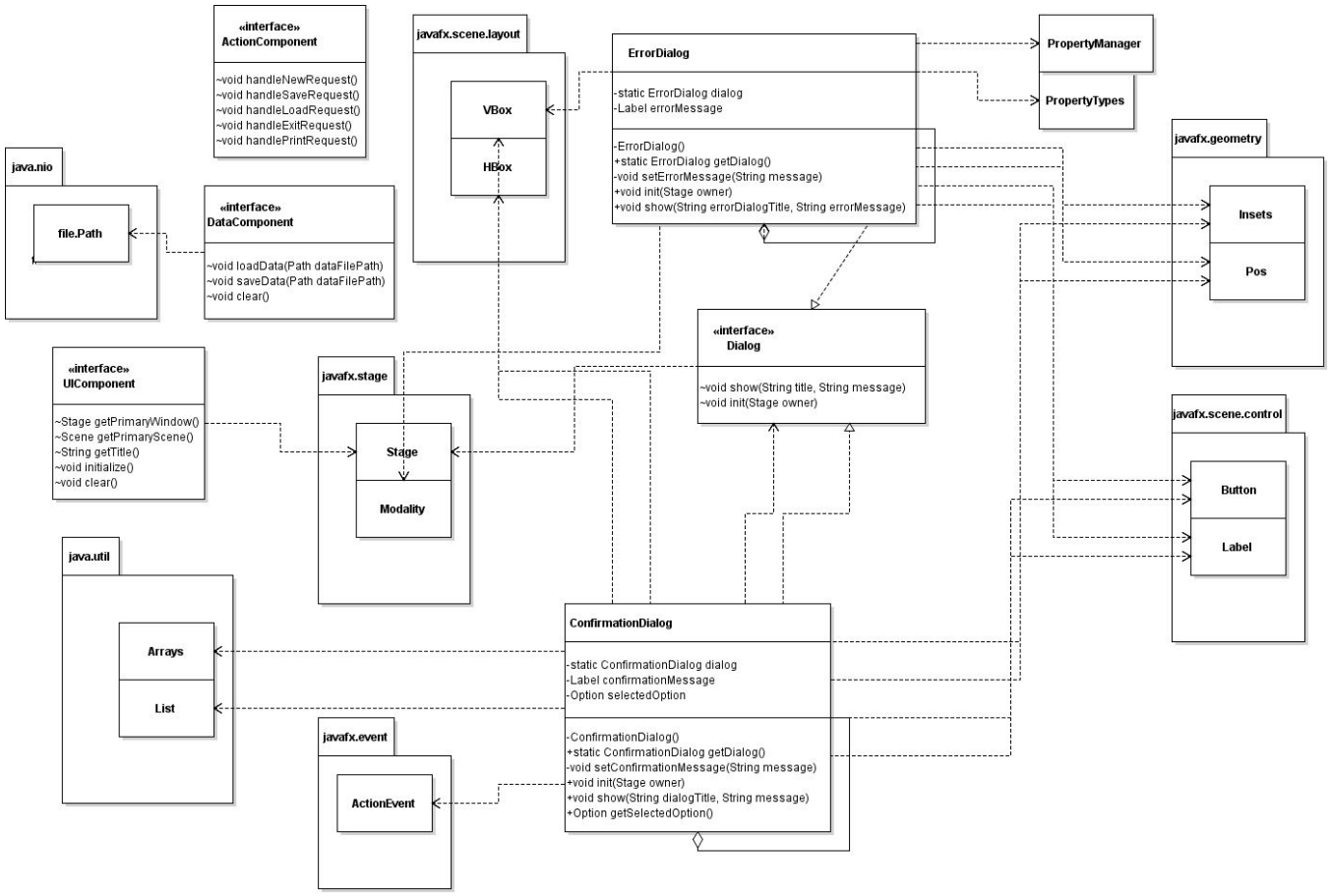


Figure 3.5 Detailed PropertyManager class diagram

Note: This is the Property Manager of DataViLiJ that sustains and hold methods to which to derive property types, load inputted data on to a map/display, and initialize elements/attributes. The subsystems communicates with AppUI, and UITemplate, ErrorDialog, and InitializationParams to achieve these said objectives.

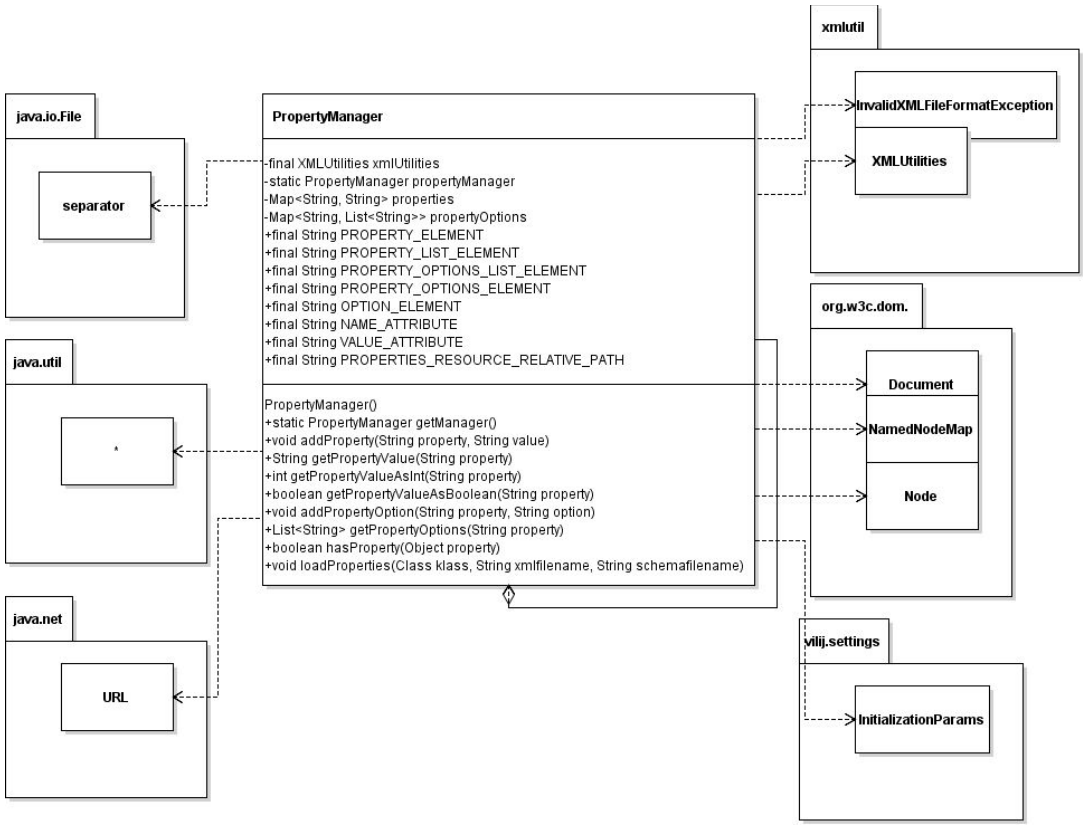


Figure 3.6 Detailed AppUI class diagram

Note: This subsystem contains of the “main class” of DataViLiJ and is communicated with every other Dialog, Template, and Data defined classes to display desired UI. This is where Buttons are initialized and derives many java API methods or parameters to specify the data of said UI. The chart, algo

methods/buttons, general workspace are also initialized.

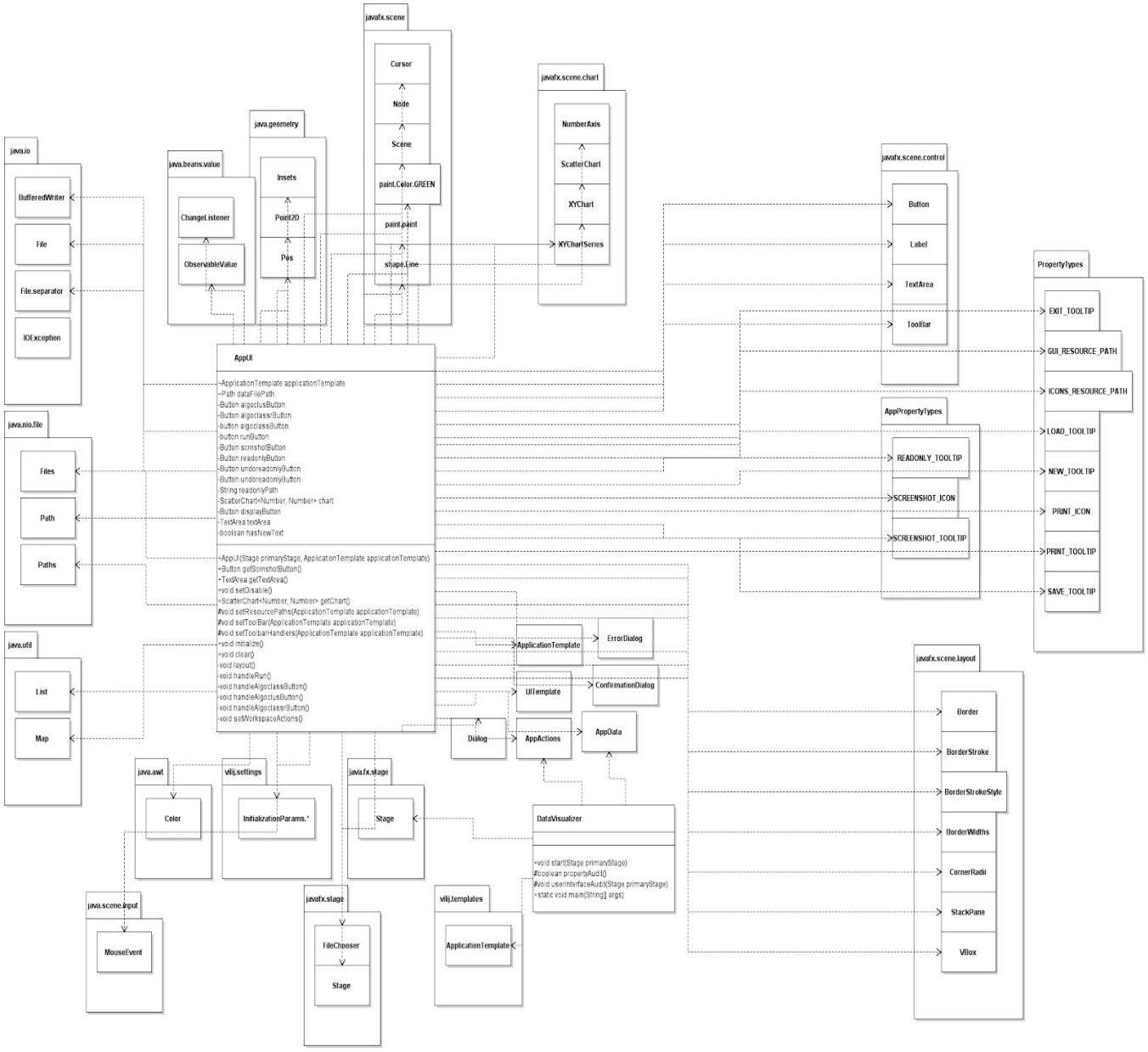


Figure 3.6 Detailed AppActions class diagram

Note: This subsystem is event-handling class with lambda expressions of print,save,exit,load, and sshot Buttons that UI displays and goes through the communication of this subclass to implement the given button function (algo or not).

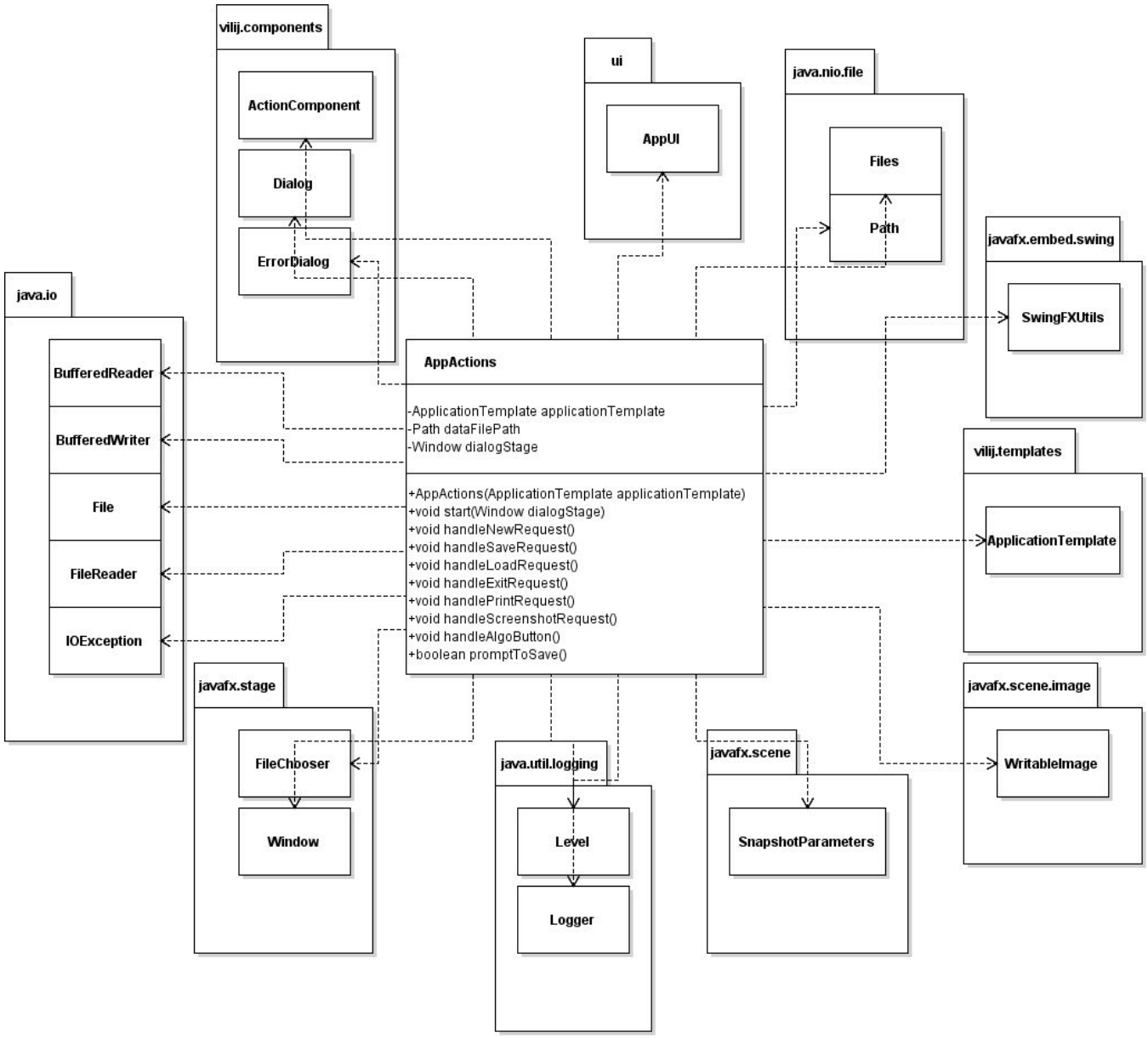
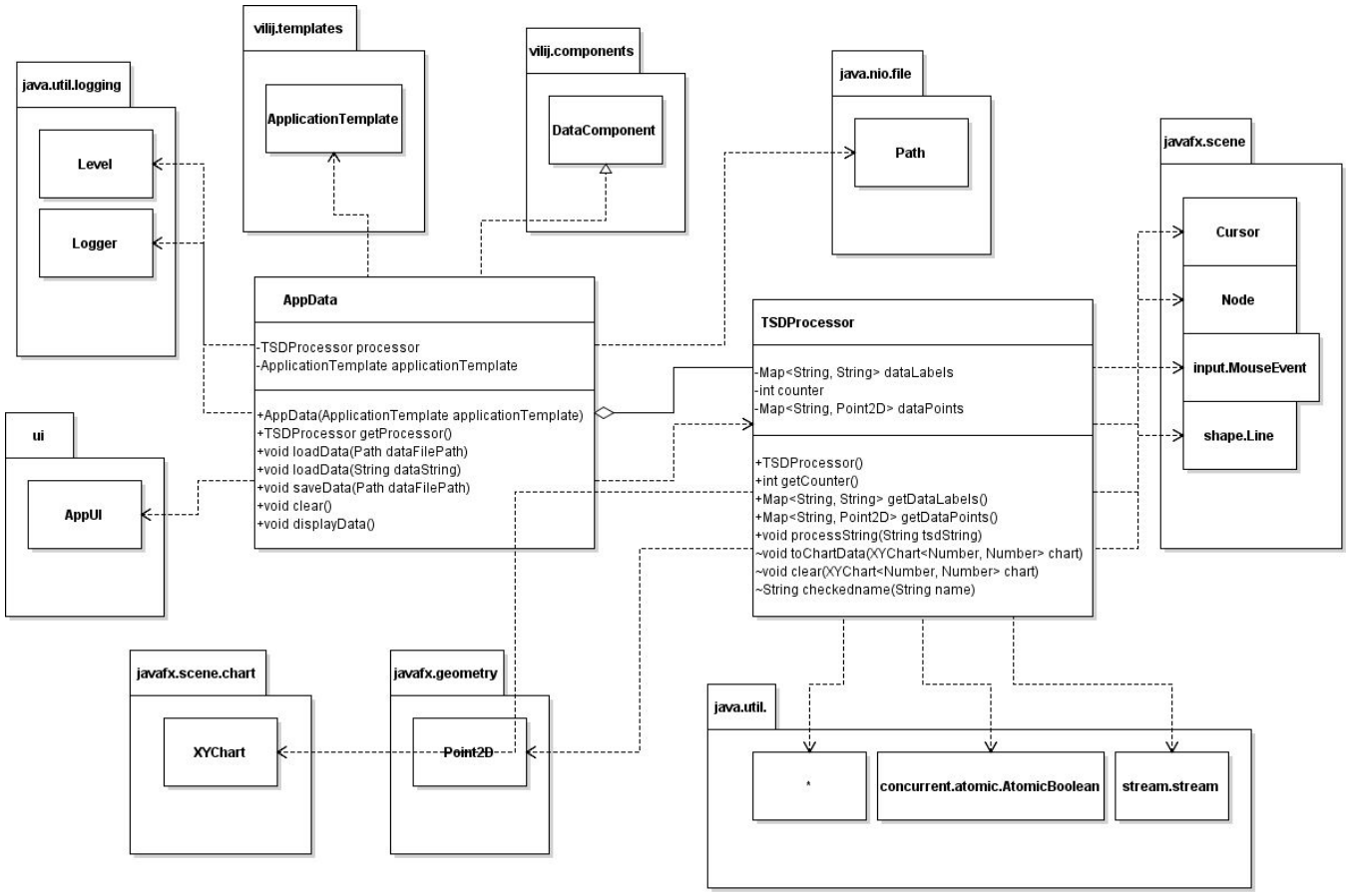


Figure 3.6 Detailed AppData/TSDProcessor class diagram

Note: This subsystem contains classes that go with data-handling to load,save,clear (newbutton sub function), initialize data values on said chart via Map<String,String> declarations. This compiled with the javafx API to communicate with Data Component, AppUI, Data Visualizer.



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 new button UML sequence diagram

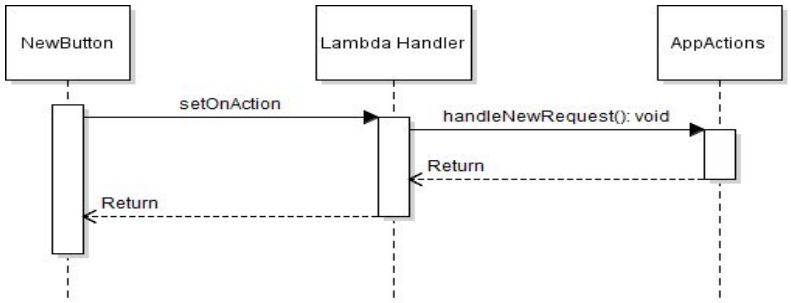


Figure 4.2 load button UML sequence diagram

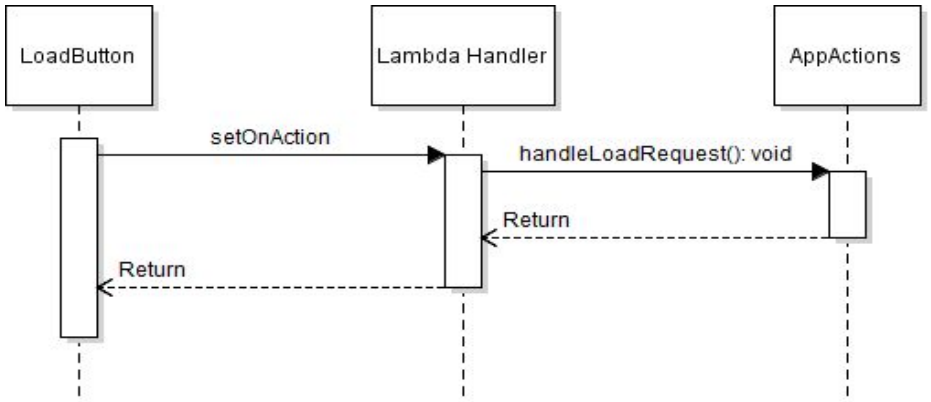


Figure 4.3 save as button UML sequence diagram

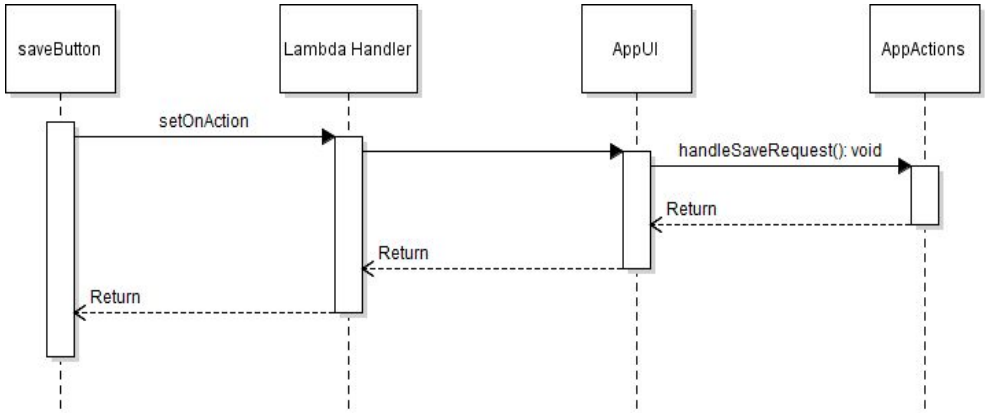


Figure 4.4 ScreenShot button UML sequence diagram

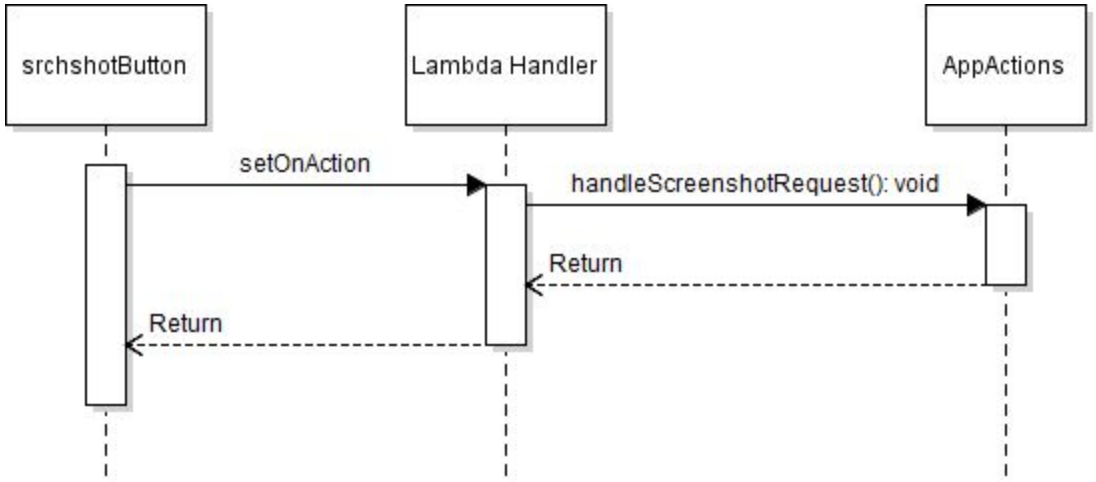


Figure 4.5 Exit button UML sequence diagram

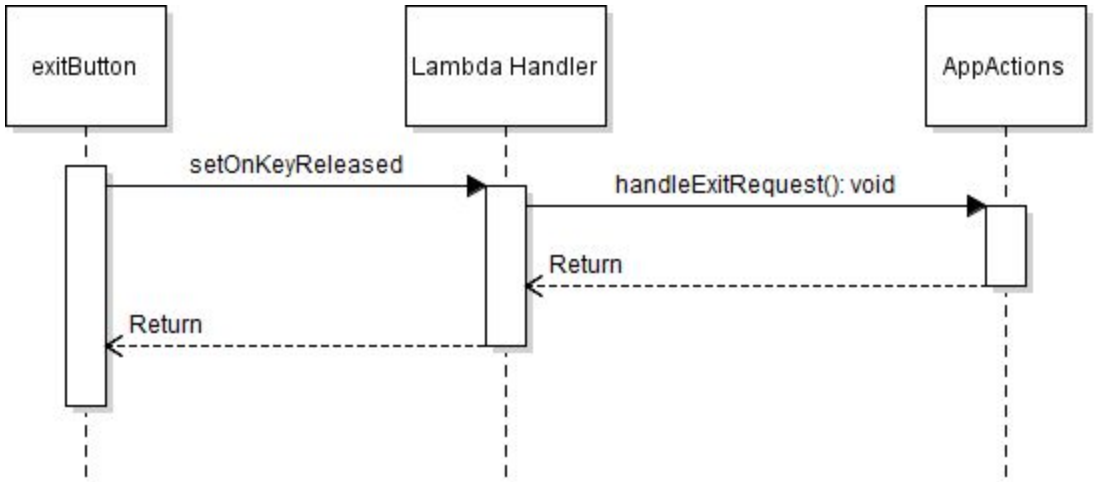


Figure 4.6 Export button UML sequence diagram

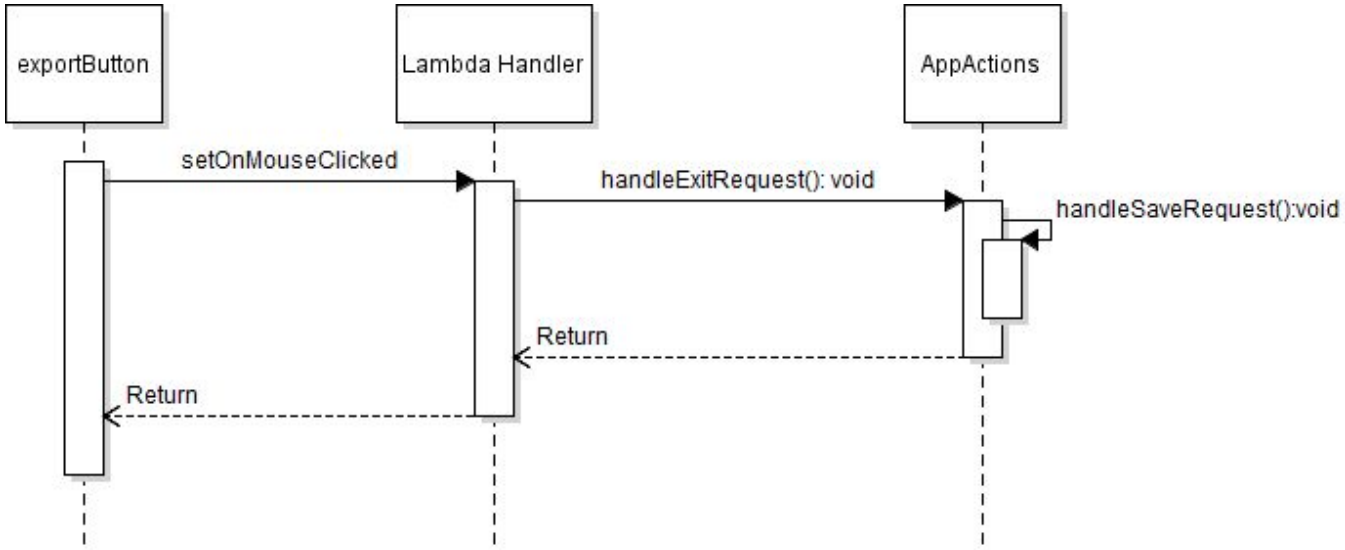


Figure 4.7 Run button UML sequence diagram

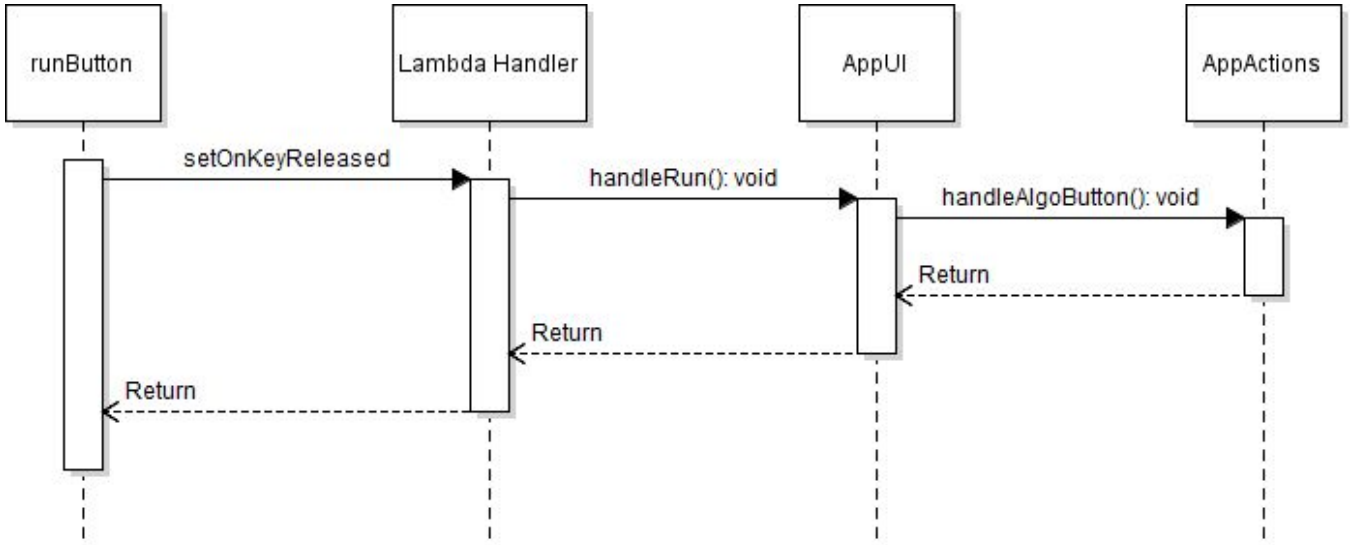


Figure 4.8 Classification button UML sequence diagram

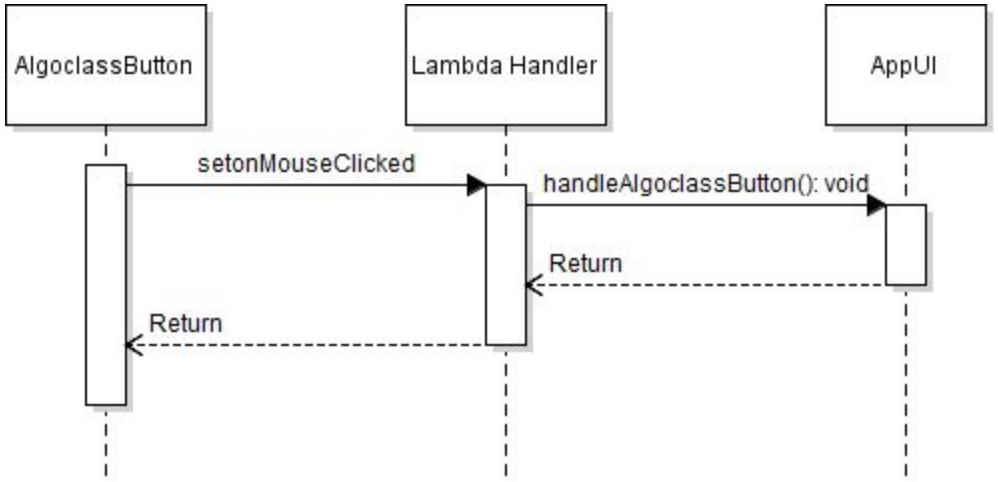


Figure 4.9 Clustering button UML sequence diagram

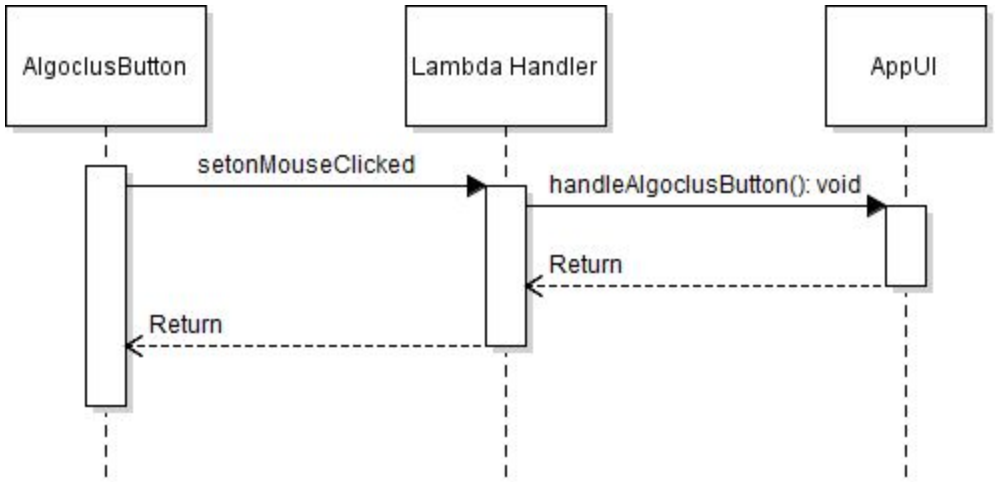


Figure 4.10 AlgoType button UML sequence diagram

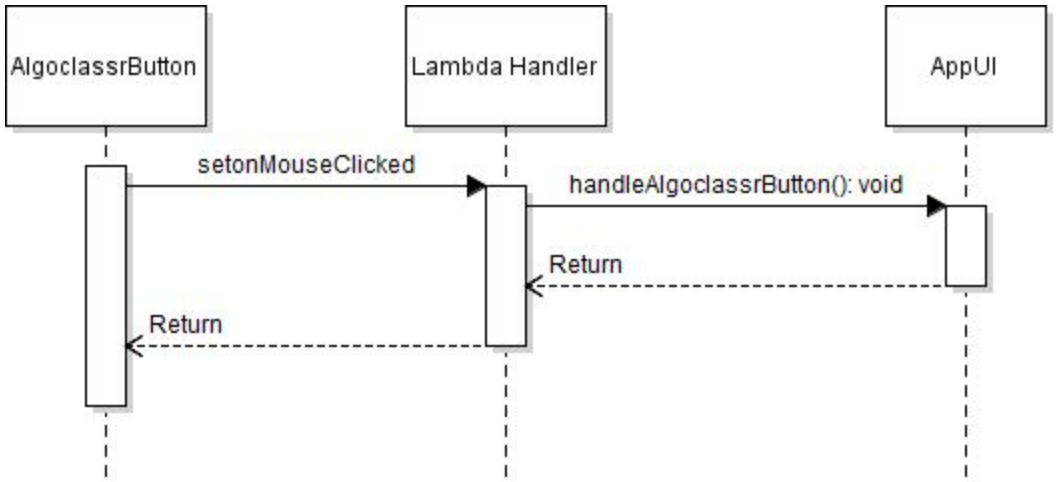
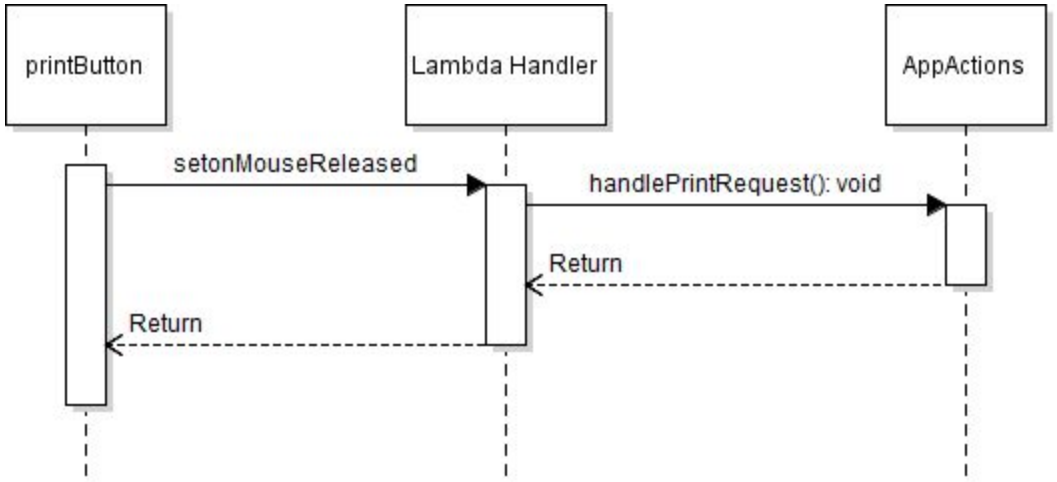


Figure 4.11 Print button UML sequence diagram



5 File/Data structures and formats

Appendix A: Tab-Separated Data (TSD) Format

The data provided as input to this software as well as any data saved by the user as a part of its usage must adhere
to the tab-separated data format specified in this appendix. The specification are as follows:

1. A file in this format must have the ".tsd" extension.
2. Each line (including the last line) of such a file must end with '\n' as the newline character.
3. Each line must consist of exactly three components separated by '\t'. The individual components are
    a. Instance name, which must start with '@'
    b. Label name, which may be null.
    c. Spatial location in the x-y plane as a pair of comma-separated numeric values. The values must be no more specific than 2 decimal places, and there must not be any whitespace between them.
4. There must not be any empty line before the end of file.
5. There must not be any duplicate instance names. It is possible for two separate instances to have the same spatial location, however.

6 Supporting Information

This document should serve as a reference for the designers and developers in the future stages of the development process. Since this product involves the use of a specific data format and specialized algorithms,we provide the necessary information in this section in the form of three appendixes.

Appendix A: Characteristics of Learning Algorithms

As mentioned before, the algorithms and their implementation, or an understanding of their internal workings, are not within the scope of this software. For the purpose of design and development, it

suffices to understand some basic characteristics of these algorithms. There are provided in this appendix.

A.1 Classification Algorithms

These algorithms 'classify' or categorize data into one of two known categories. The categories are the labels provided in the input data. The algorithm learns by trying to draw a straight line through the x-y 2-dimensional plane that separates the two labels as best as it can. Of course, this separation may not always be possible, but the algorithm tries nevertheless. An overly simplistic example where an algorithm is trying to separate two genders based on hair length (x-axis) and count (y-axis) is shown in Fig. 5 (picture courtesy dataaspirant.com). The output of a classification algorithm is, thus, a straight line. This straight line is what is updated iteratively by the algorithm, and must be shown as part of the dynamically updating visualization in the GUI. Moreover, a classification algorithm will NOT change

- the label of any instance provided as part of its input, Fig. 5. Output of a classification algorithm
- the actual (x, y) position of any instance in its input.

The run configuration of a classification algorithm should comprise the maximum number of iterations (it may, of course, stop before reaching this upper limit), the update interval, and the continuous run flag. These are shown in Fig. 4.

A.2 Clustering Algorithms

Clustering algorithms figure out patterns simply based on how data is distributed in space. These algorithms do not need any labels from the input data. Even if the input data has labeled instances, these algorithms will simply ignore them. They do, however, need to know the total number of labels ahead of time (i.e., before they start running). Therefore, their run configuration will comprise the maximum number of iterations (it may, just like classification algorithms, stop before reaching this upper limit), the update interval, the number of labels, and the continuous run flag. The output of a clustering algorithm is, thus, the input instances, but with its own labels. The instance labels get updated iteratively by the algorithm, and must be shown as part of the dynamically updating visualization in the GUI. This, along with the importance of providing the number of labels in the run configuration, is illustrated in Fig. 6 (a) and (b) below.

Fig. 6 (a). The input data need not have any labels for a clustering algorithm. Shown here as all instances having the same color. The number of labels decides what the output will look like.

Fig. 6 (b). If the number of labels provided to the algorithm as part of its run configuration is 3, then this is what its output might look like. If the run configuration specified 2 labels, the green instances would get split between the blue and purple ones.

Just like a classification algorithm, the actual position of any instance will not change.