DataViLij
Software Design Description

Author: Aditya Taday
April 2018
Version 1.0

Abstract: DataViLiJ (Data Visualization Library in Java) 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 1016TM-2009 document format

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

1. Introduction

This is the Software Design Description (SDD) for the DataViLij application. Note that this document format is based on the IEEE Standard 1016-2009 recommendation for software design.

1.1 Purpose

This document is to serve as the blueprint for the construction of the 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, 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

DataViLij will be using frameworks to avoid duplication of work. Frameworks such as ViLij and XMLUtiles will be designed and used. 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.

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

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

1.4 References

IEEE Std 830TM-1998 (R2009) – IEEE Standard for Information Technology – Systems Design – Software Design Descriptions

DataViLiJ SRS – Software Requirements Specification for the DataViLiJ application

1.5 Overview

This Software Design Description document provides a working design for the DataViLij software application as described in the DataViLij 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 Draw.io editor.

2. Package-Level Design Viewpoint

As mentioned, this design will encompass the DataViLij application, ViLiJ and XMLUtiles Framework to be used in the construction. In building them, 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 DataViLij, ViliJ and XMLUtiles Framework overview

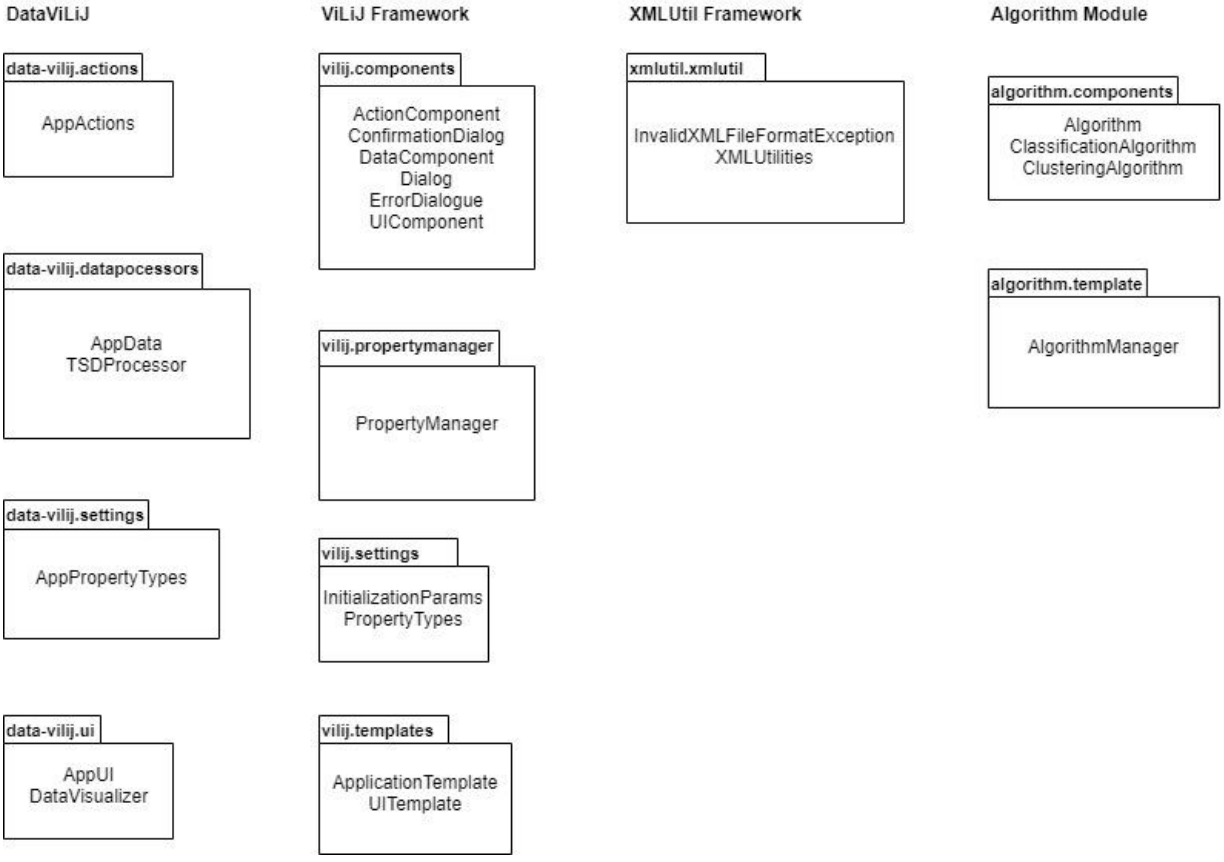

Figure 2.1 specifies all the components to be developed and places all classes in home packages

2.2 Java API Usage

All the frameworks and the DataViLij application will be developed using the Java programming languages. As such, this design will make use of the classes specified in Figure 2.2.

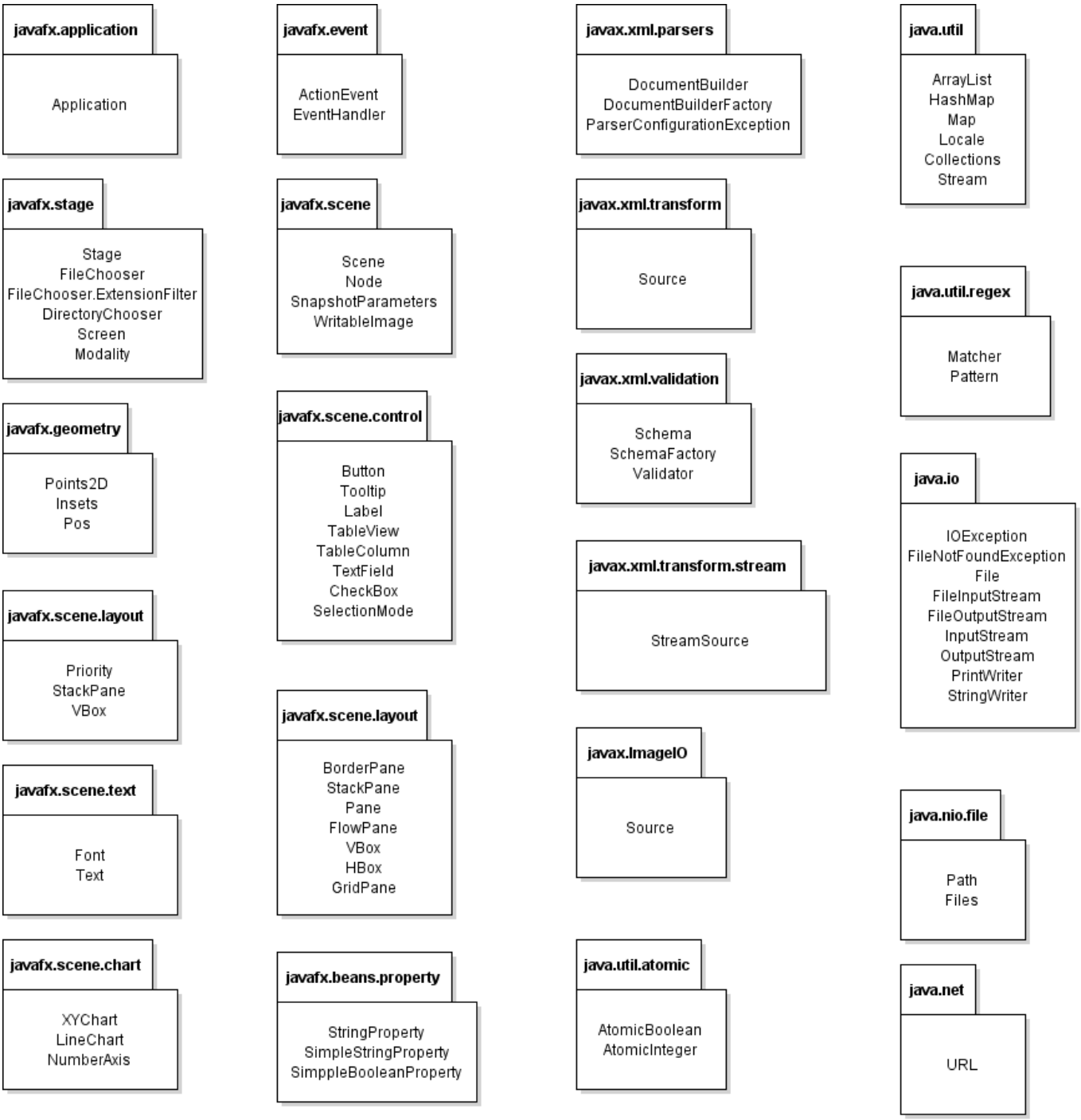

Figure 2.2: Java API Classes and Packages to Be Used

2.3 Java API Usage Descriptions

Tables 2.1-2.24 below summarize how each of these classes will be used.

Class	Use
Application	Used for extending JavaFX applications.

Table 2.1: Uses Used for classes in the Java API's javafx.application package

Class	Use
Stage	Used for the top level of JavaFX container.
FileChooser	Used for support of standard file dialogs.
FileChooser.ExtensionFilter	Used for filtering which files can be chosen in a FileDialog based on the file name extensions.
DirectoryChooser	Used for standard directory chooser dialogs.
Screen	Used for describing the characteristics of a graphics destination such as monitor.
Modality	Used for defining the possible modality types Used for a Stage.

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

Class	Use
BorderPane	Used for laying out children in top, left, right, bottom, and center positions.
Pane	Used for layout panes which need to expose the children list as public so that users of the subclass can freely add/remove children.
FlowPane	Used for laying out children in a flow that wraps at the flowpane's boundary.
VBox	Used for laying out children in a single vertical column.
HBox	Used for laying out children in a single horizontal row.
GridPane	Used for laying out children within a flexible grid of rows and columns.
StackPane	Used for laying out children in a back-to-front stack.

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

Class	Use
Scene	Used for containing all content in a scene graph.
Node	Used for a base class used as scene graph nodes.
SnapshotParameters	Used for specifying the rendering attributes Used for Node snapshot.
WritableImage	Used for representing a custom graphical image that is constructed from pixels supplied by the application.

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

Class	Use
XYChart	Used for drawing the two axes and the plot content
LineChart	Used for creating a line object in a chart.
NumberAxis	Used for creating an axis object that plots a range of numbers.

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

Class	Use
Button	Used for a simple button control.
Tooltip	Used for showing additional inUsed formation about a Control when the Control is hovered over by the mouse.
Label	Used for a non-editable text control.
TableView	Used for designing an unlimited number of rows of data, broken out into columns.
TableColumn	Used for making columns in TableView.
TextField	Used for text input component that allows a user to enter a single line of unUsed formatted text.
CheckBox	Used for an implementation of the CheckBox
SelectionModel	Used for providing a consistent API Used for maintaining selection.
SplitPane	Used for 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.

Table 2.6: Uses for classes in the Java API's javafx.scene.control package

Class	Use
Points2D	Used for describing the bounds of points.
Insets	Used for providing the set of inside offsets Used for an object
Pos	Used for a set of values Used for describing vertical and horizontal positioning and alignment.

Table 2.7: Uses for classes in the Java API's javafx.geometry package

Class	Use
ActionEvent	Used for an Event representing some type of action.
EventHandler	Used for handler Used for events of a specific class / type.

Table 2.8: Uses for classes in the Java API's javafx.event package

Class	Use
StringProperty	Used for a full implementation of a Property wrapping a String value.
SimpleStringProperty	Used for a full implementation of a Property wrapping a String value.
SimpleBooleanProperty	Used for a full implementation of a Property wrapping a Boolean value.

Table 2.9: Uses for classes in the Java API's javafx.beans.property package

Class	Use
ArrayList	Used for a resizable-array implementation of the List interface
HashMap	Used for a hash table-based implementation of the Map interface.
Map	Used for an object that maps keys to values.
Locale	Used for a representation of specific geographical, political, or cultural region.
Collection	Used for static methods that operate on or return collections

Table 2.10: Uses for classes in the Java API's java.util package

Class	Use
Matcher	Used for an engine that perUsed forms match operations on a character sequence by interpreting a Pattern.
Pattern	Used for a compiled representation of a regular expression.

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

Class	Use
AtomicBoolean	Used for a Boolean value which may be updated atomically.
AtomicInteger	Used for an Integer value which may be updated atomically.

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

Class	Use
DocumentBuilder	Used for defining the API to obtain DOM Document instances from an XML document.
DocumentBuilderFactory	Used for defining a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.
ParserConfigurationException	Used for indicating a serious configuration error.

Table 2.13: Uses for classes in the Java API's javax.xml.parsers package

Class	Use
Schema	Used for an immutable in-memory representation of grammar.
SchemaFactory	Used for creating Schema objects. Entry-point to the validation API.
Validator	Used for a processor that checks an XML document against Schema.

Table 2.14: Uses for classes in the Java API's javax.xml.validation package

Class	Use
URL	Used for representing a UniUsed form Resource Locator, a pointer to a "resource" on the World Wide Web.

Table 2.15: Uses for classes in the Java API's java.net package

Class	Use
Source	Used for an object that implements the inUsed formation needed to act as source input (XML source or transUsed formation instructions).

Table 2.16: Uses for classes in the Java API's javax.xml.transUsed form package

Class	Use
StreamSource	Used for holding a transUsed formation Source in the Used form of a stream of XML markup.

Table 2.17: Uses for classes in the Java API's javax.xml.transUsed form.stream package

Class	Use
FileInputStream	Used for obtaining input bytes from a file in a file system.

FileOutputStream	Used for writing data to a File or to a FileDescriptor.
File	Used for representing files and directory pathnames.
InputStream	Used for representing an input stream of bytes.
OutputStream	Used for representing an output stream of bytes.
PrintWriter	Used for printing a Used formatted representation of objects to a text-output stream.
StringWriter	Used for collecting output in a string buffer, which can then be used to construct a string.
IOException	Used for signaling that an I/O exception of some sort has occurred.

Table 2.18: Uses Used for classes in the Java API's java.io package

3. Class-Level Design Viewpoint

As mentioned, this design will encompass the DataViLiJ application, ViLiJ framework and XMLUtiles 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.

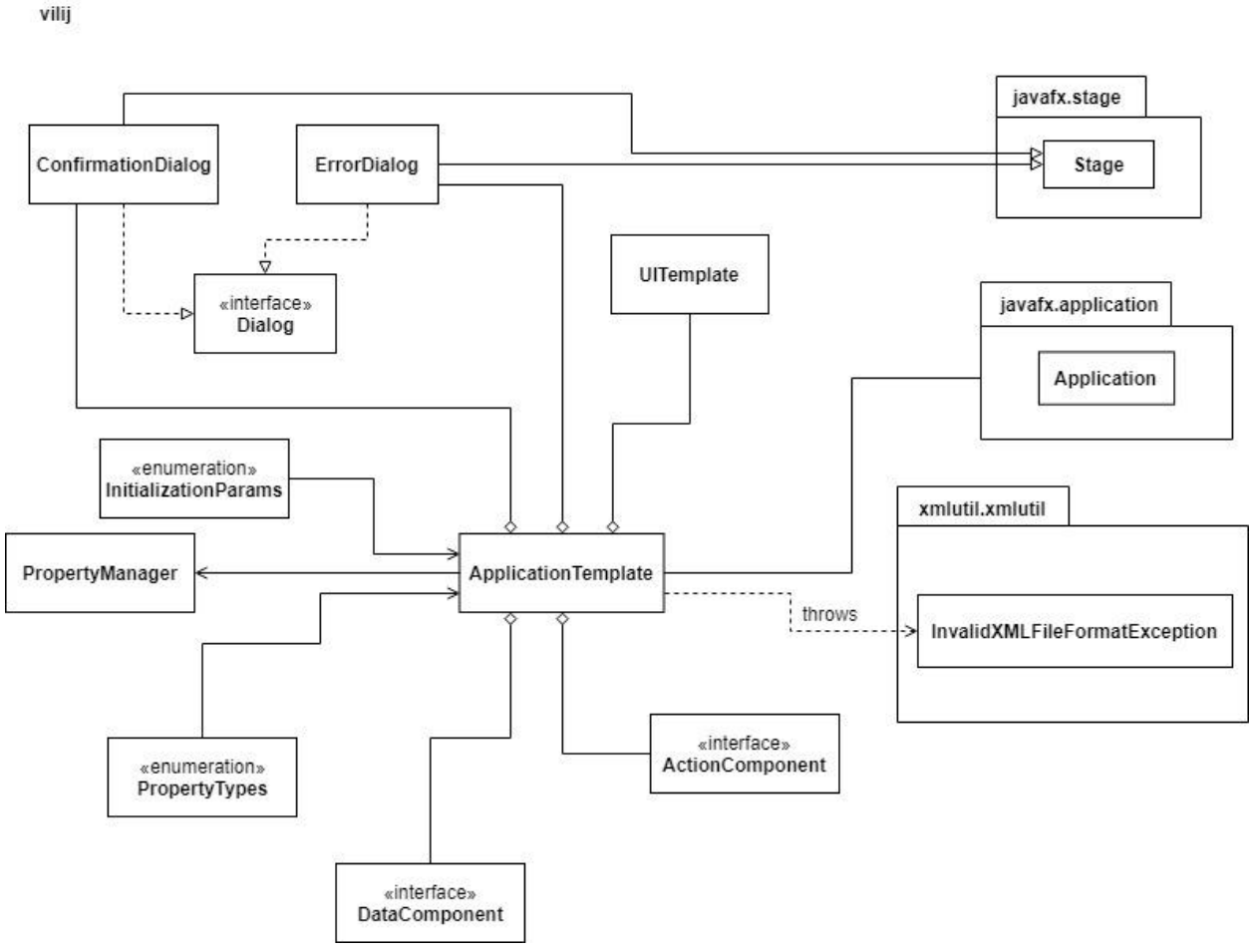

Figure 3.1: ViLiJ Overview UML Class Diagram

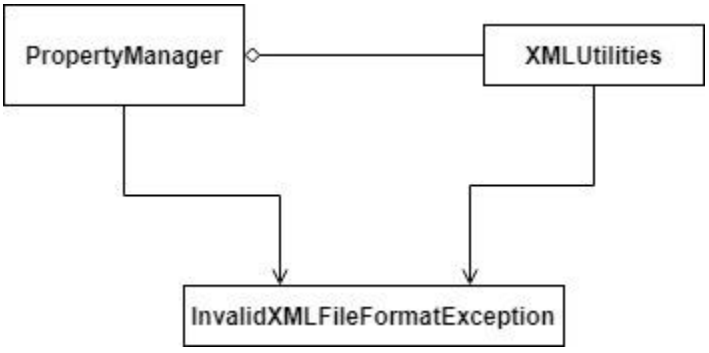

Figure 3.2: PropertiesManager Framework Overview UML Class Diagram

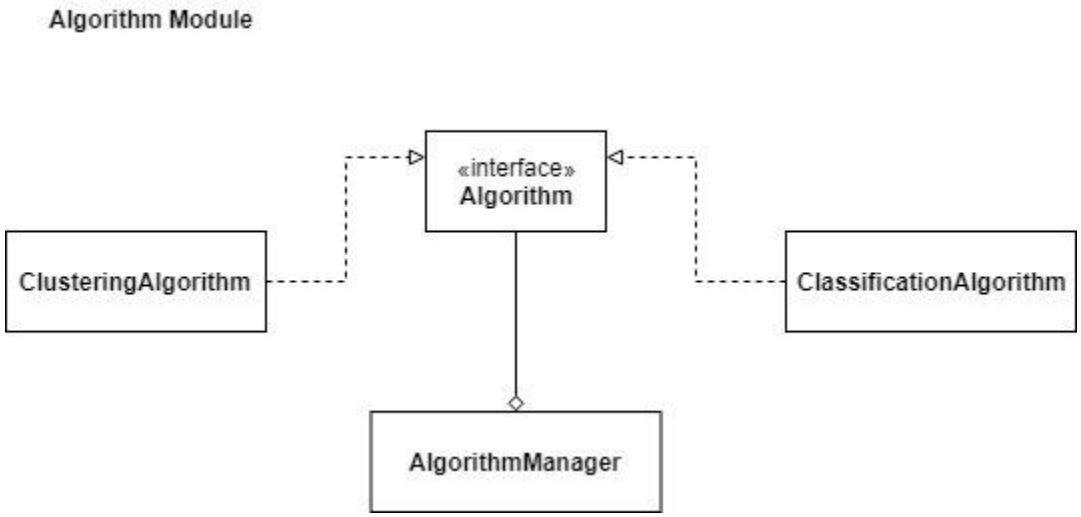

Figure 3.3: Algorithm Module Overview UML Class Diagram

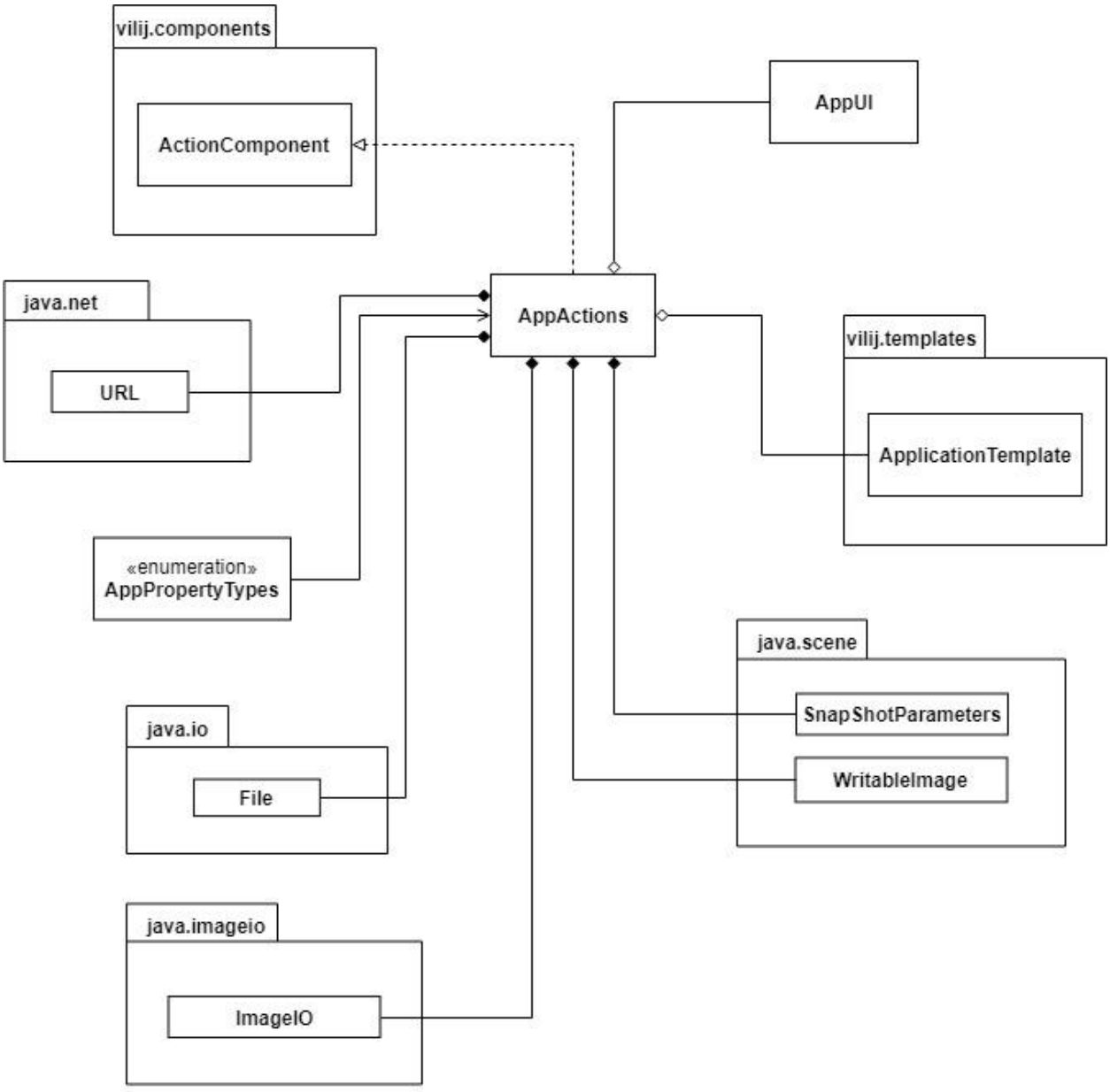

Figure 3.4: AppActions Overview UML Class Diagram

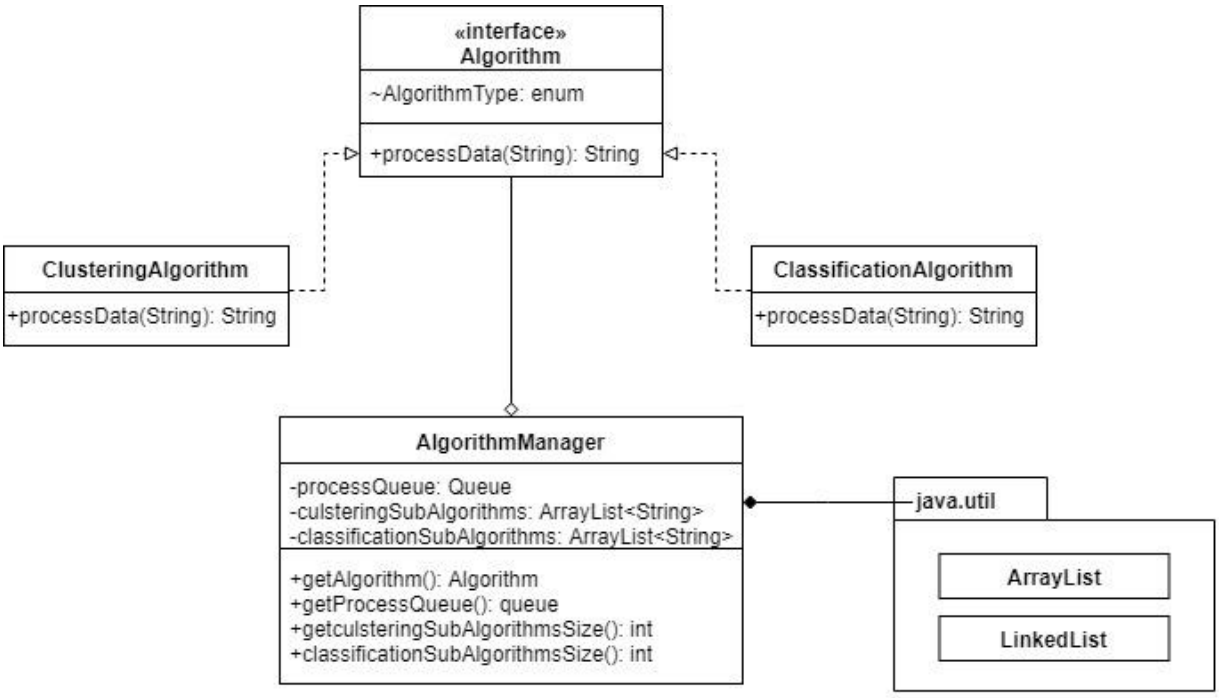

Figure 3.5: Algorithm Module Detailed UML Class Diagram

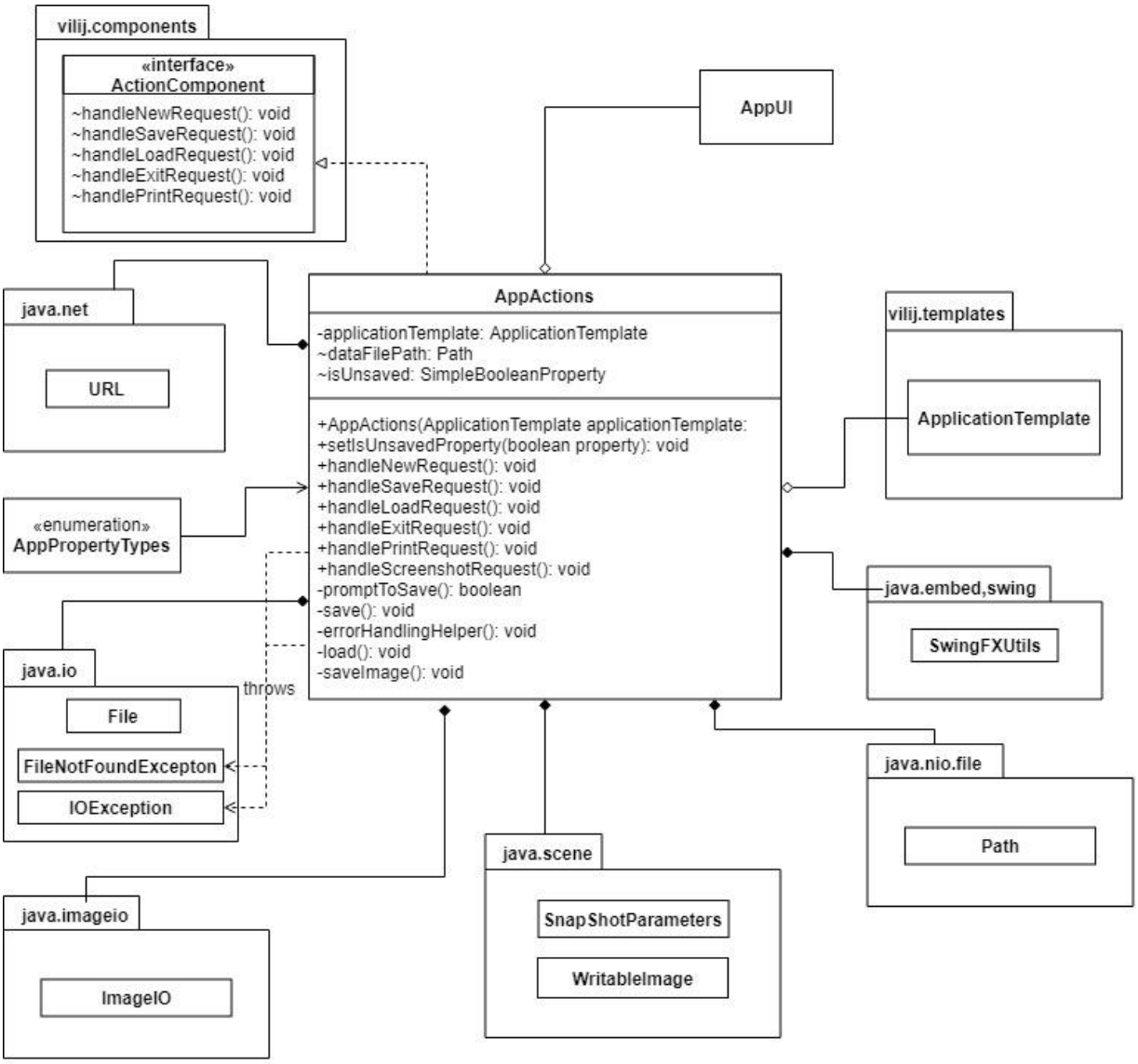

Figure 3.6: AppActions Detailed UML Class Diagram

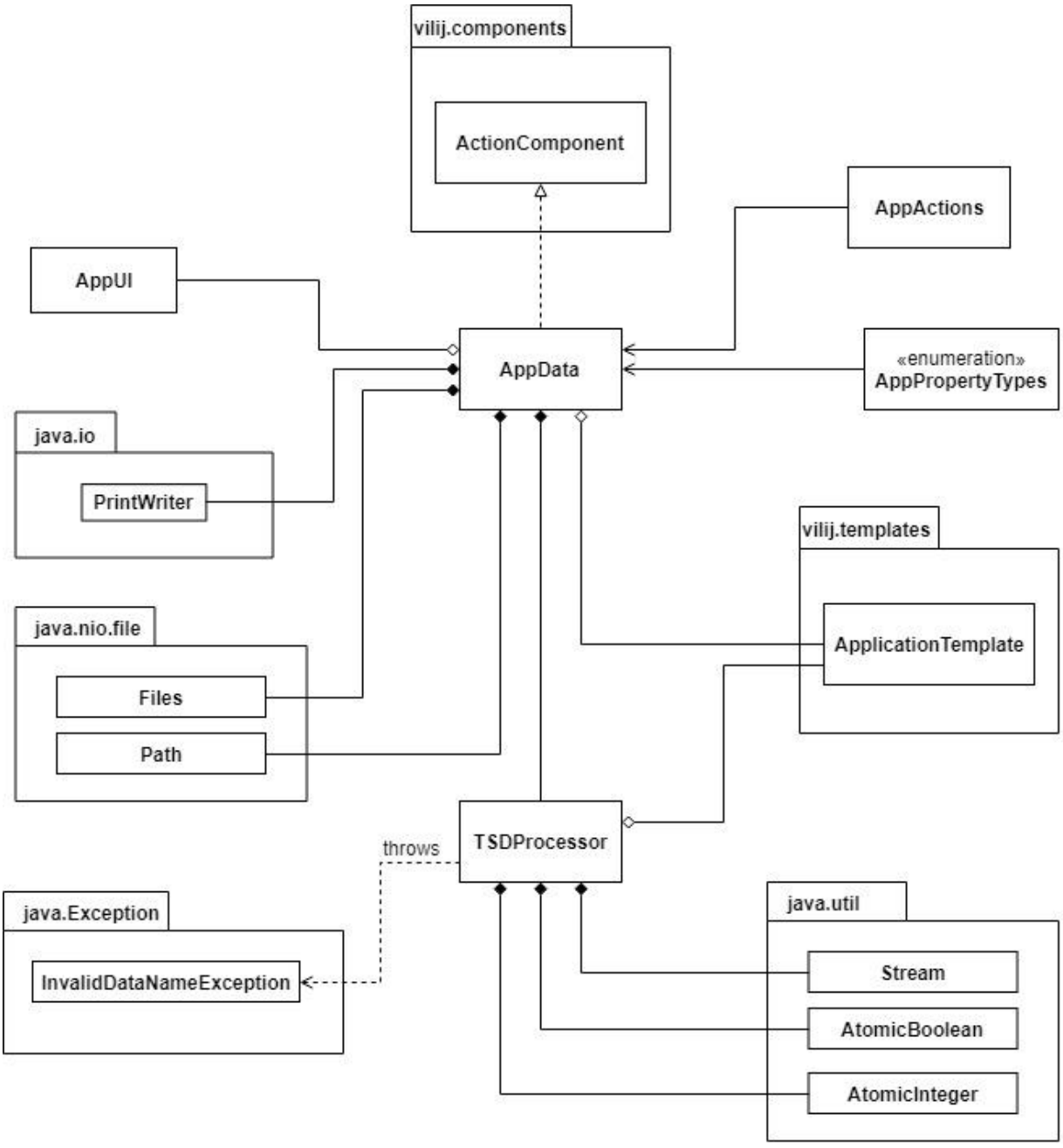

Figure 3.7: AppData and TSDProcessor Overview UML Class Diagram

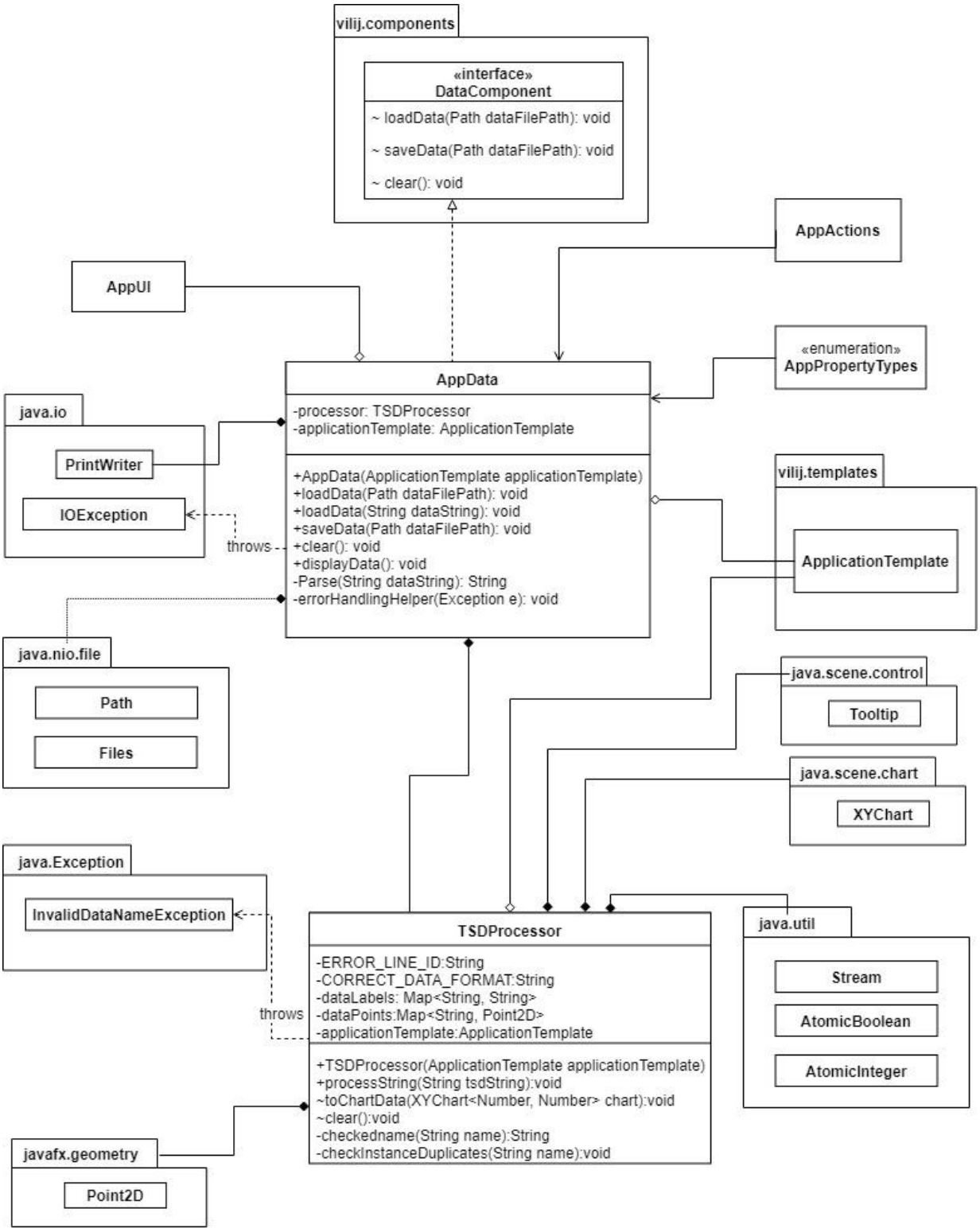

Figure 3.8: AppData and TSDProcessor detailed UML Class Diagram

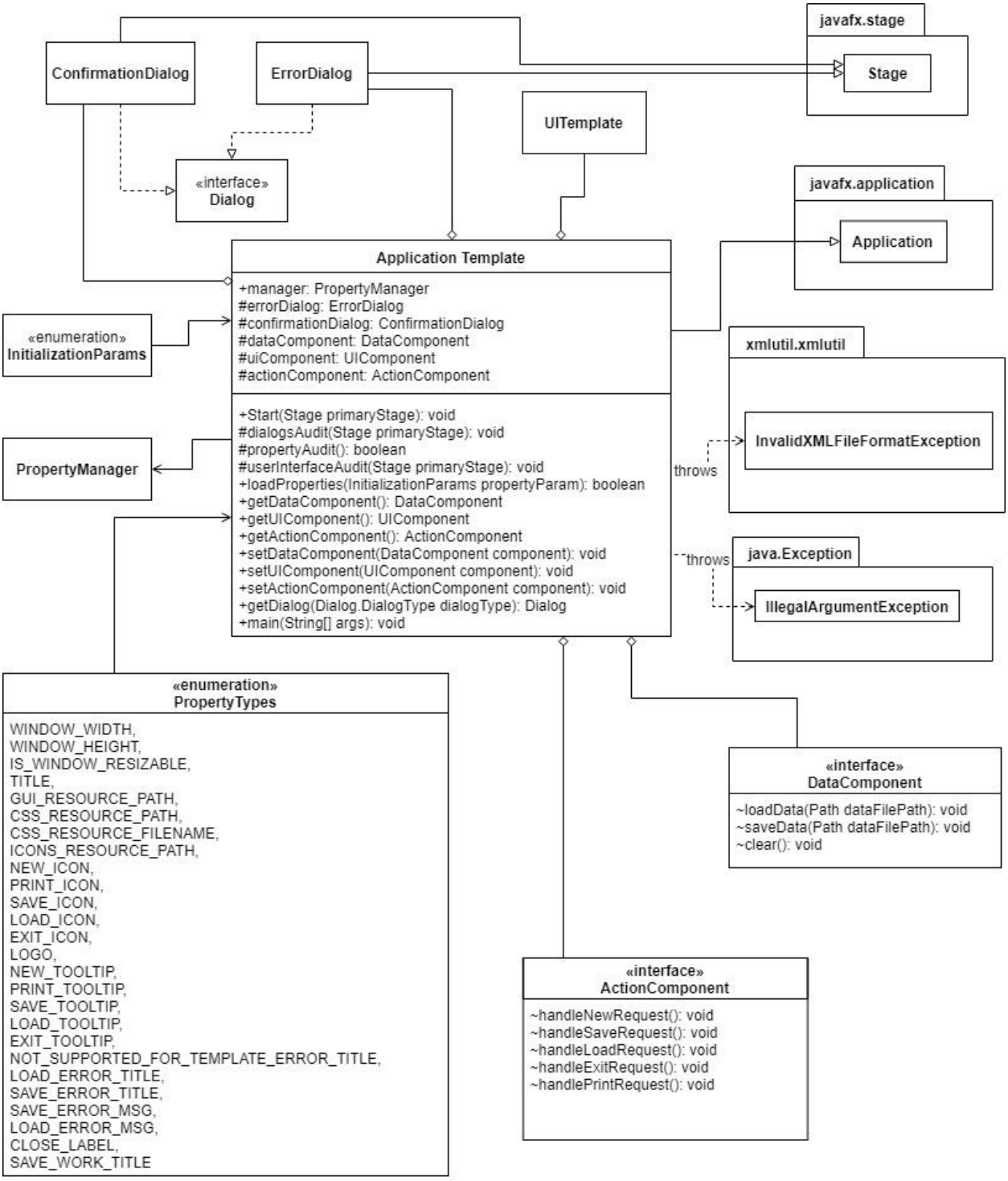

Figure 3.9: ViLiJ Framework Detailed UML Class Diagram

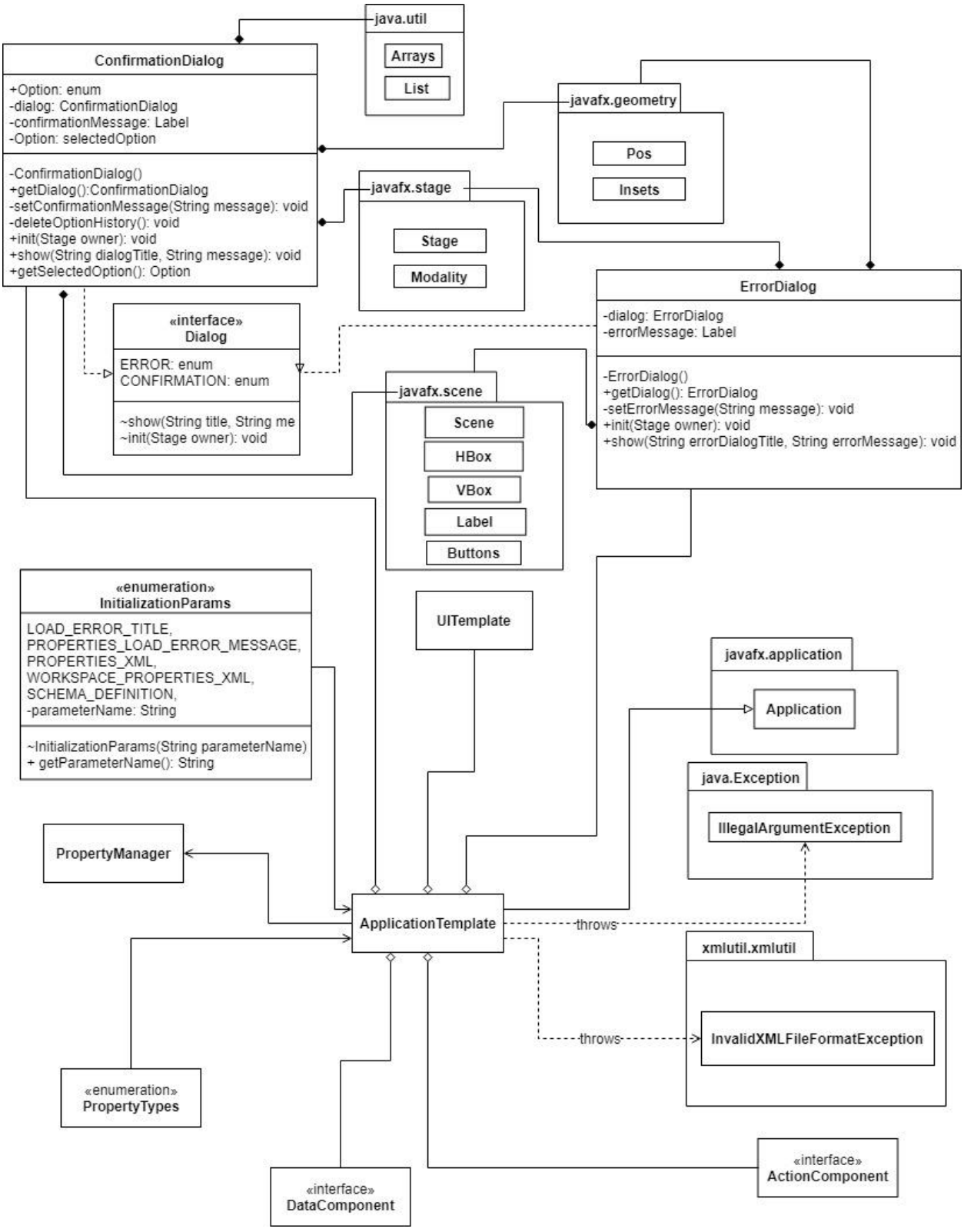

Figure 3.10: ViLiJ Framework Detailed UML Class Diagram

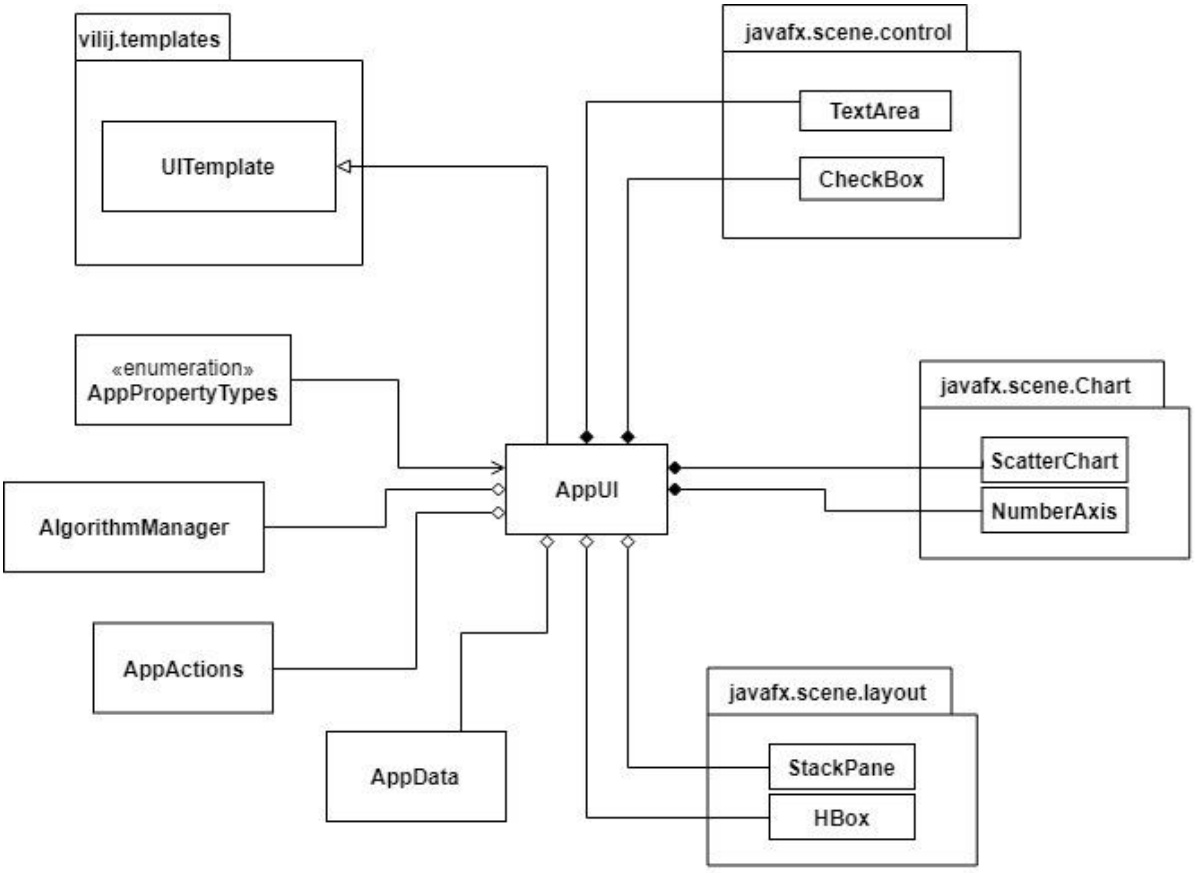

Figure 3.11: AppUI overview UML Class Diagram

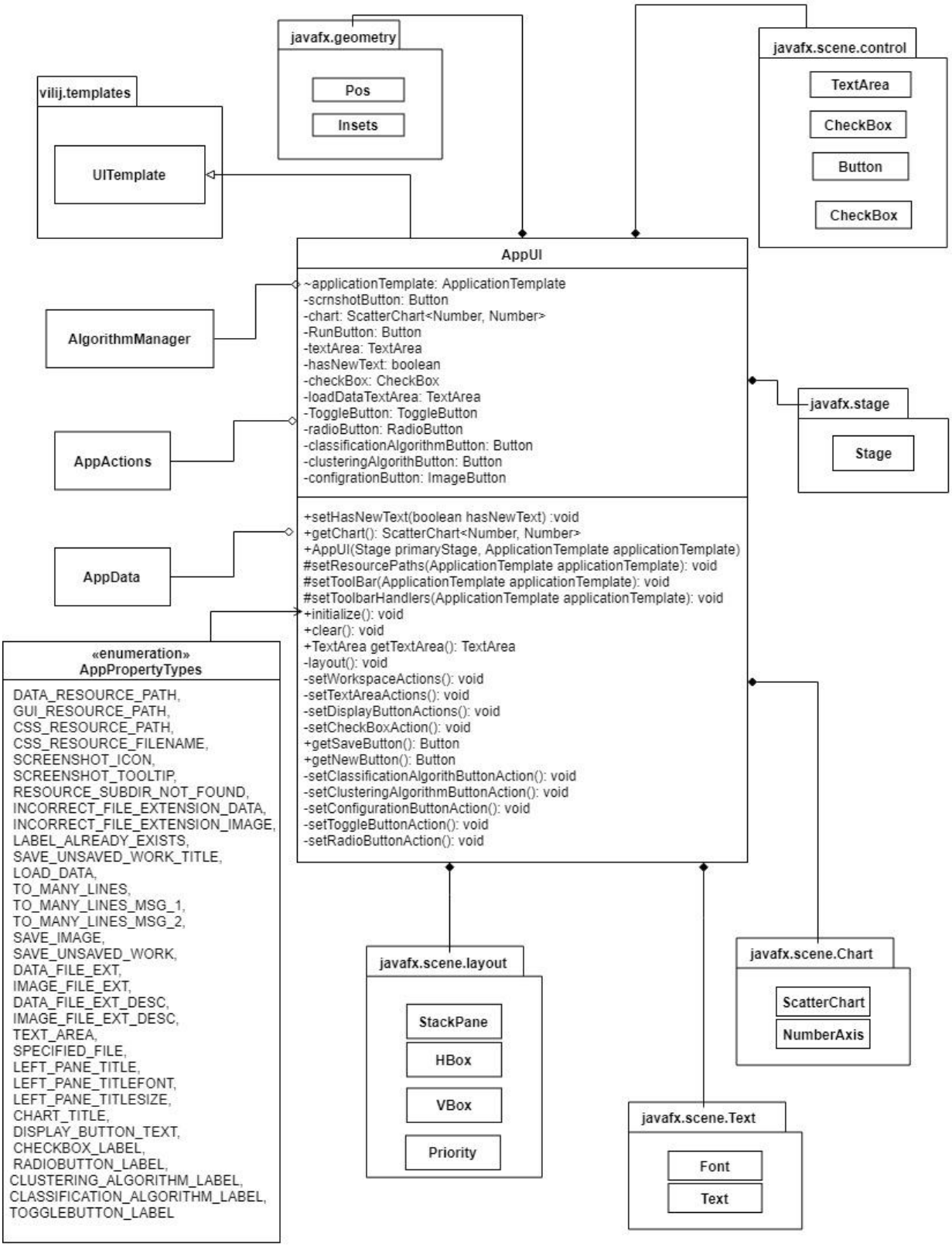

Figure 3.12: AppUI Detailed UML Class Diagram

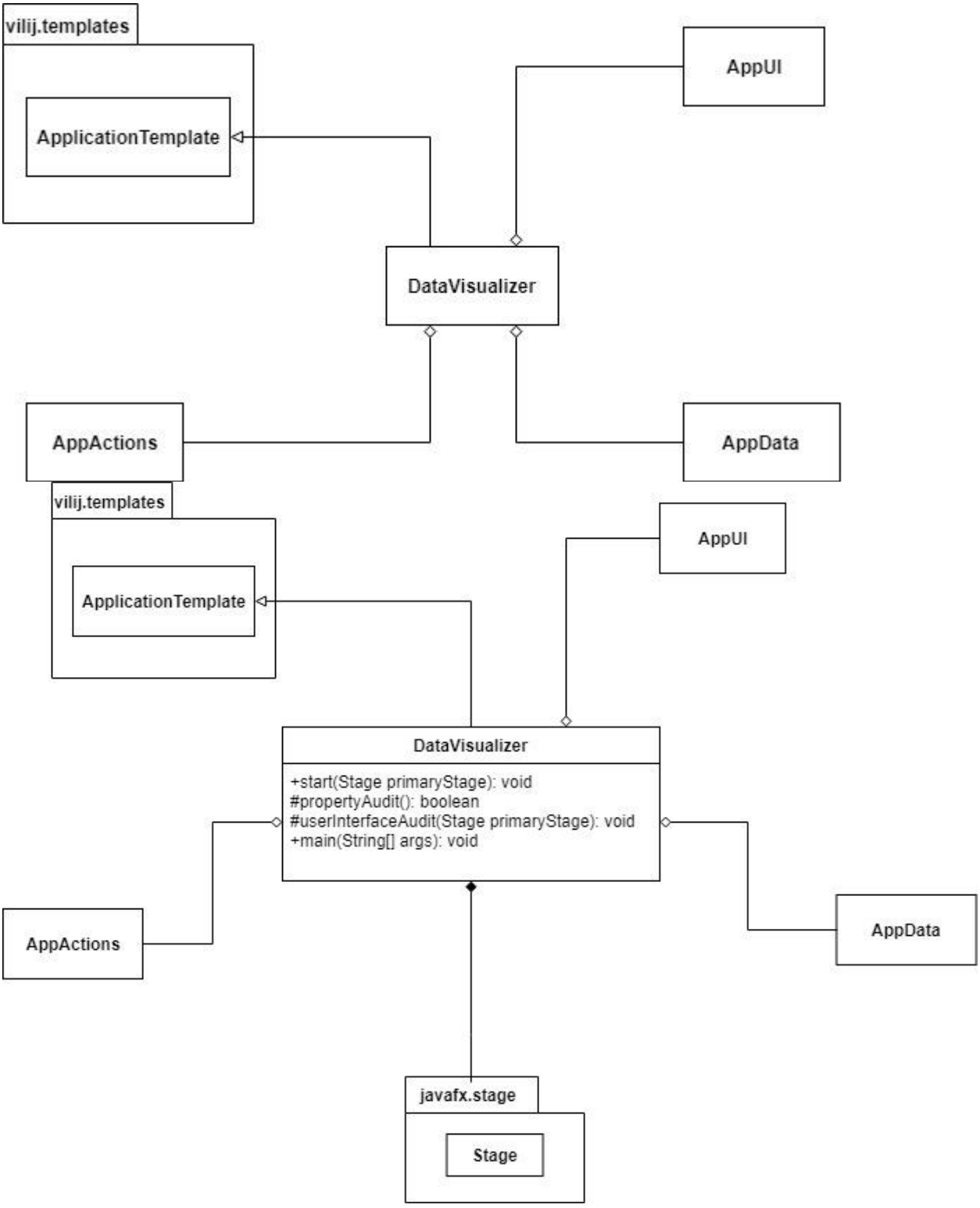

Figure 3.13: DataVizualizer Detailed UML Class Diagram

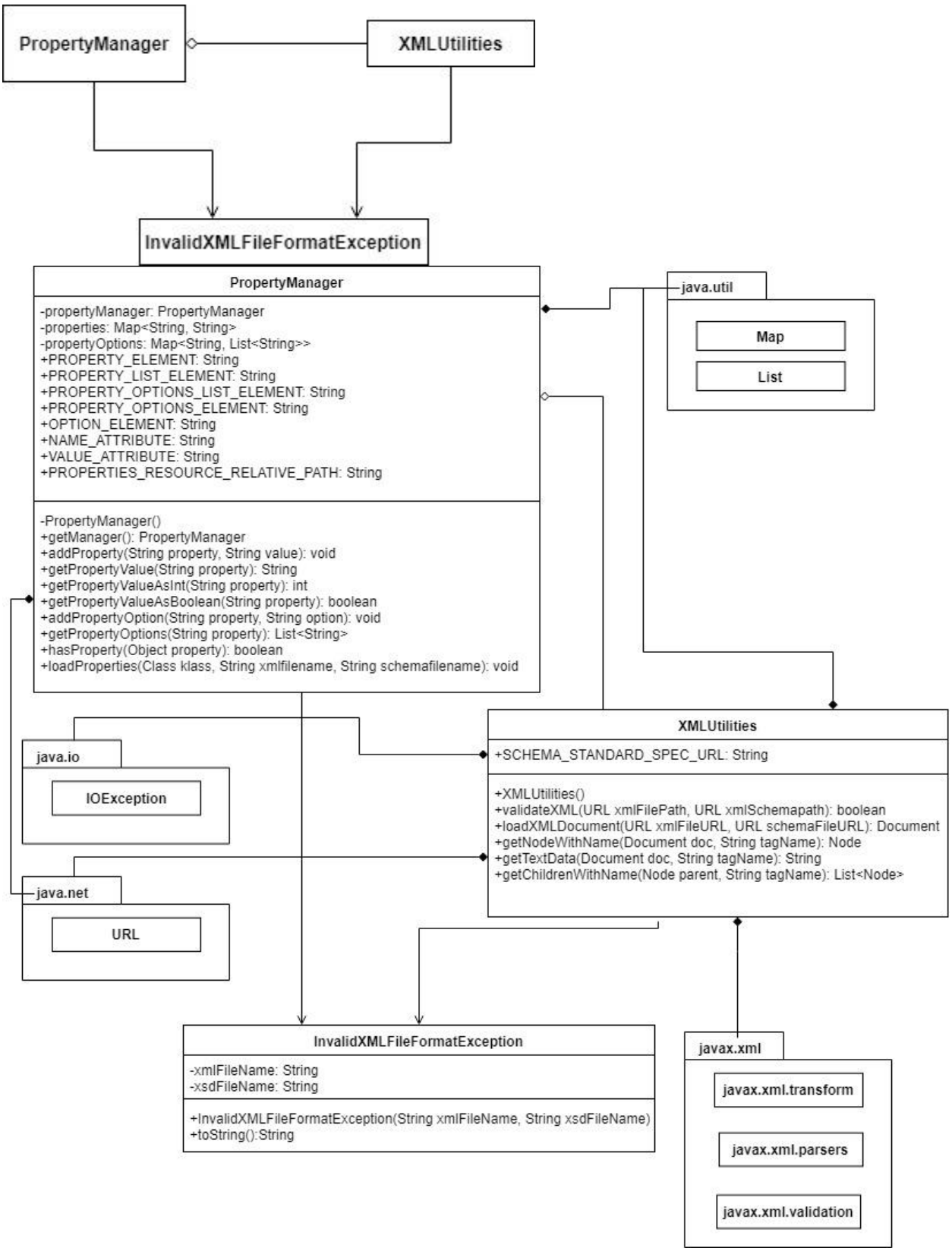

Figure 3.14: Property Manager Framework Detailed UML Class Diagram

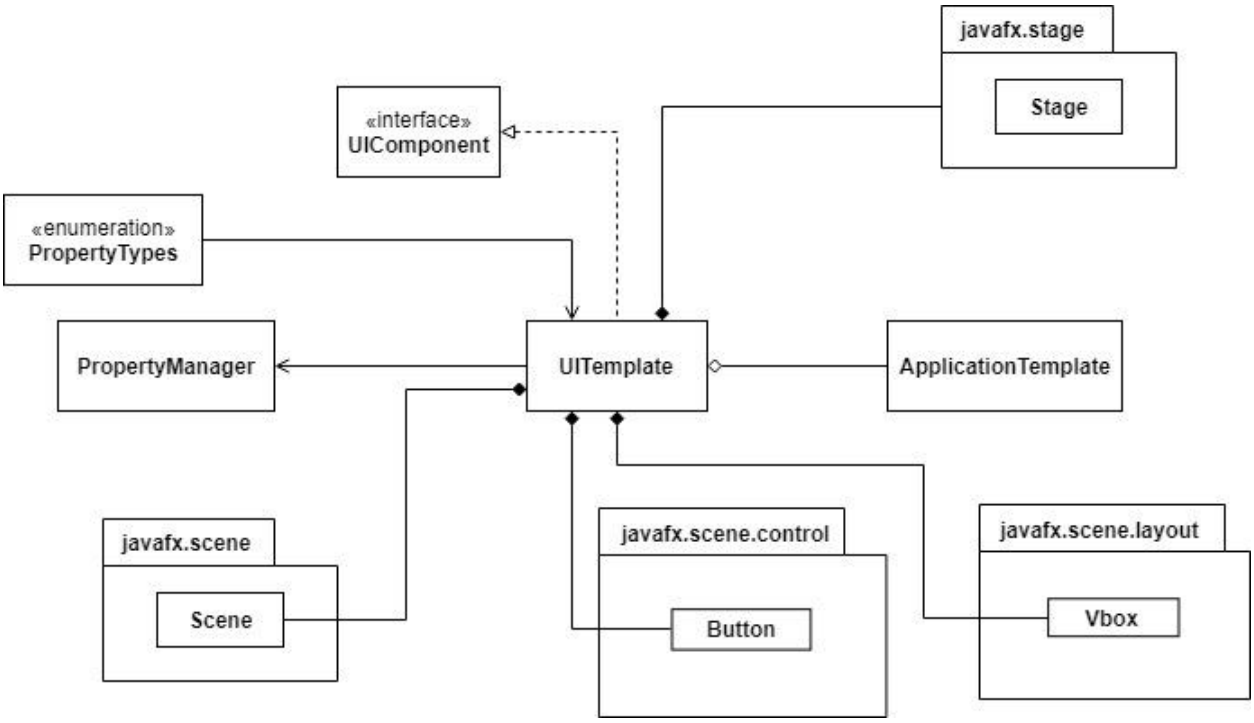

Figure 3.15: UITemplate overview UML Class Diagram

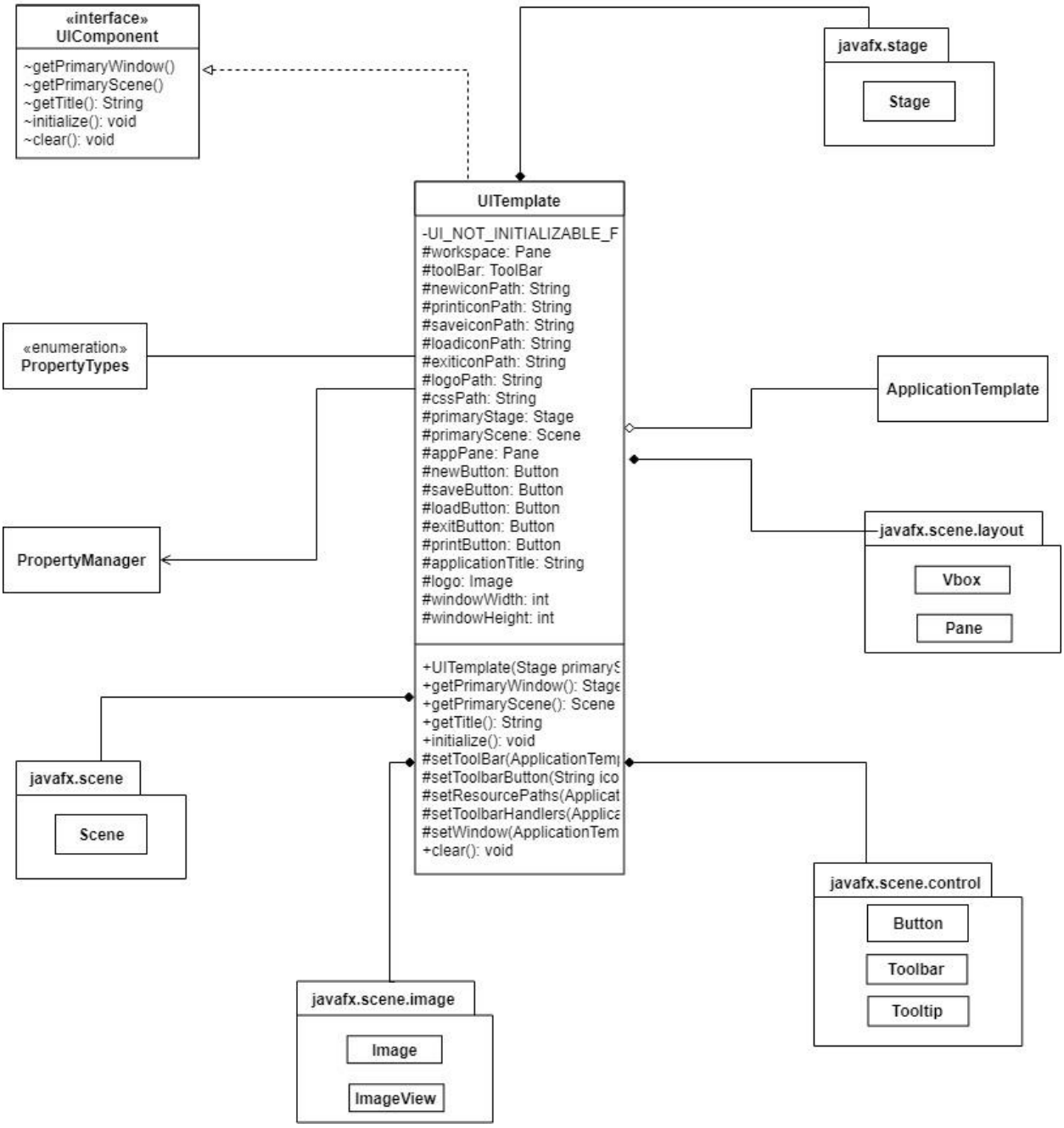

Figure 3.16: UITemplate Detailed UML Class Diagram

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 to provide the appropriate event responses.

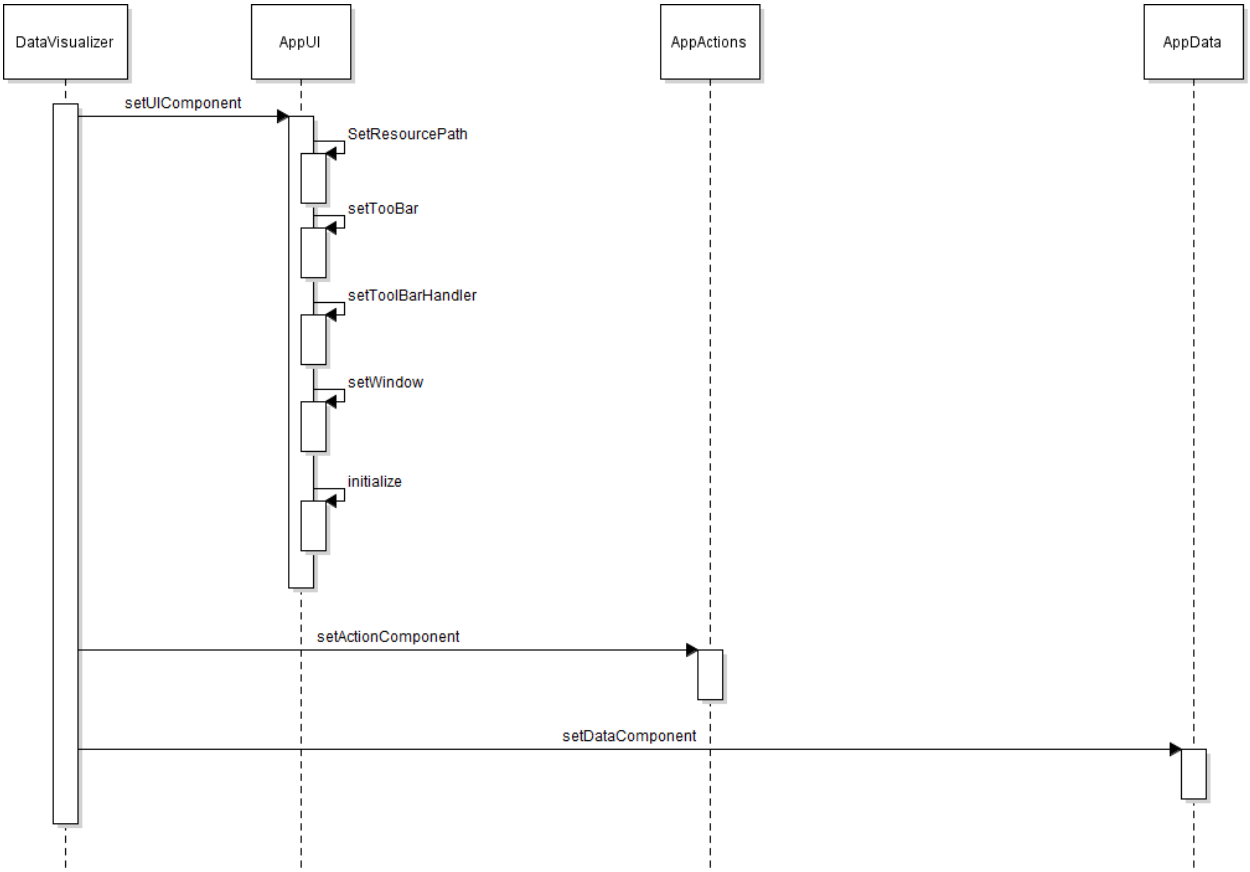

Figure 4.1 Use Case 1 – Start Application Sequence Diagram

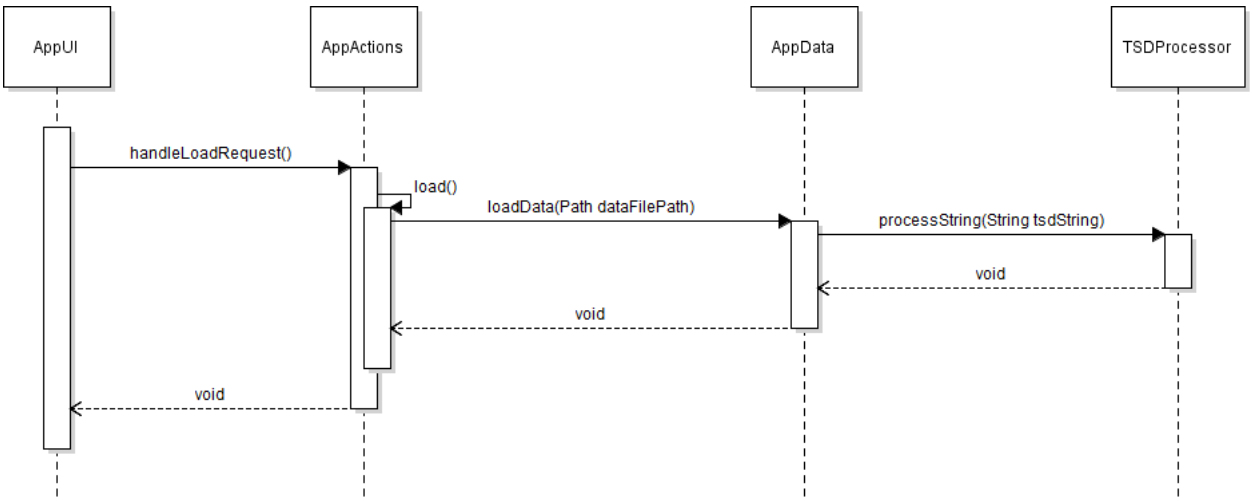

Figure 4.2 Use Case 2 – Load Data Sequence Diagram

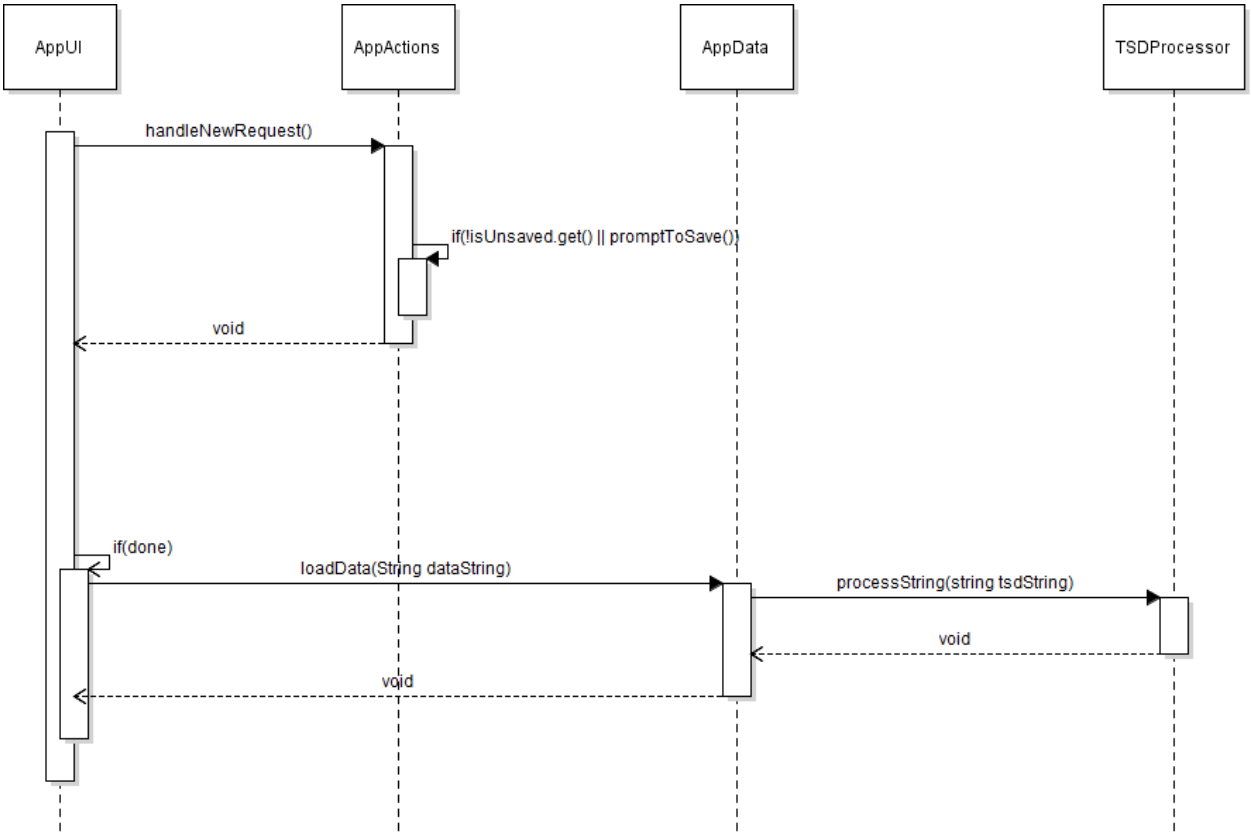

Figure 4.3 Use Case 3 – Create New Data Sequence Diagram

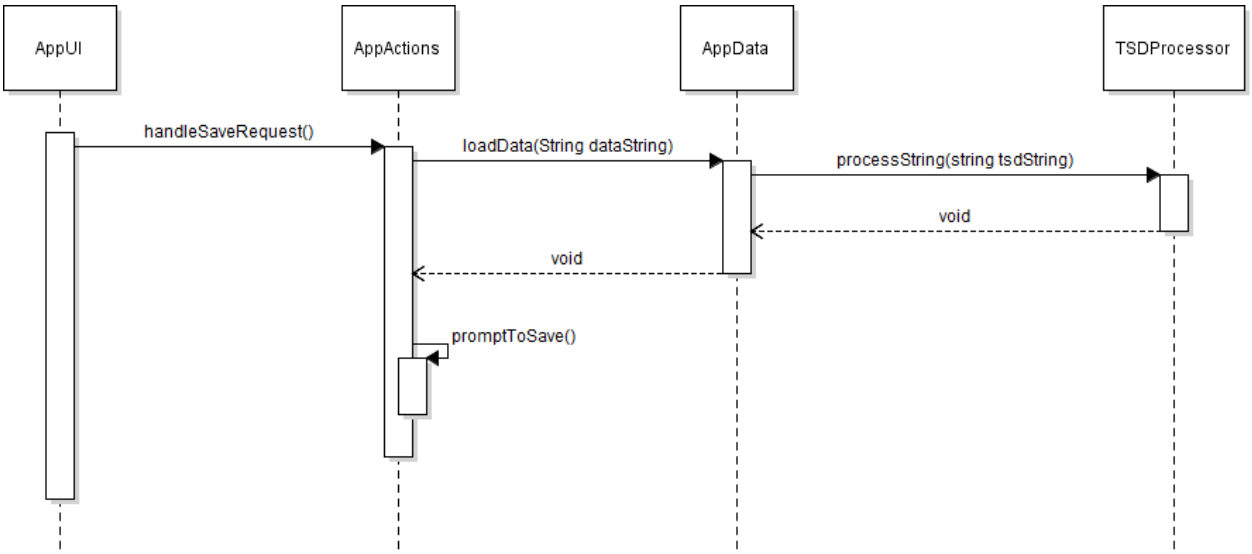

Figure 4.4 Use Case 4 – Save Data Sequence Diagram

5. File Structure and Formats

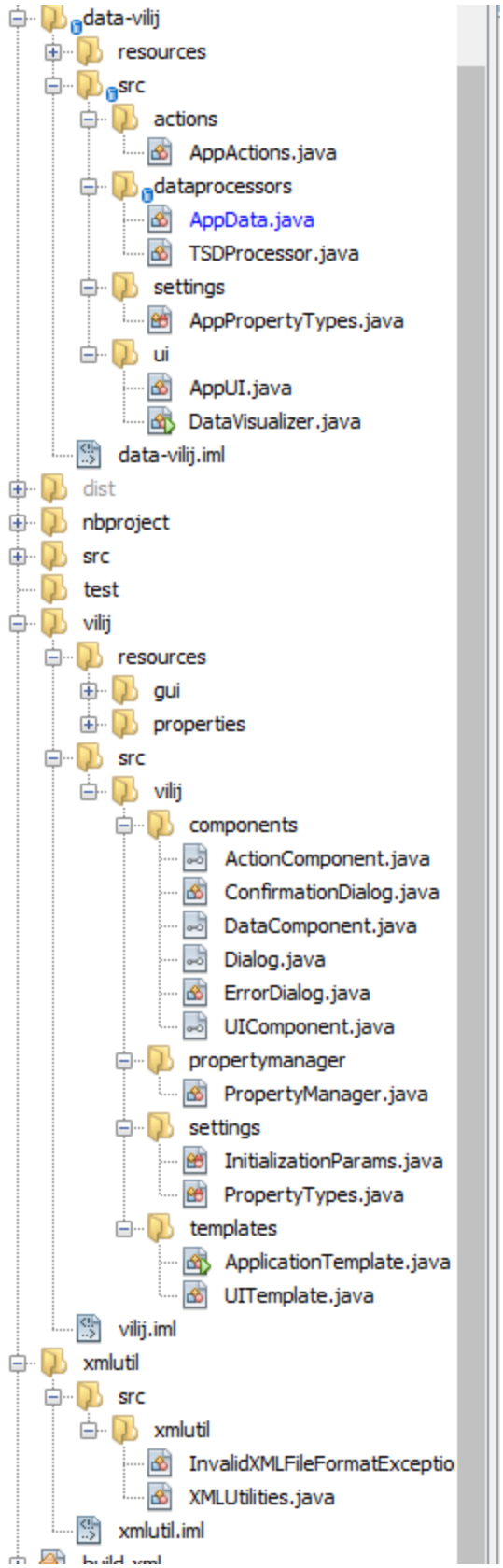
data-vilij
resources
src
actions
AppActions.java
dataprocessors
AppData.java
TSDProcessor.java
settings
AppPropertyTypes.java
ui
AppUI.java
DataVisualizer.java
data-vilij.iml
dist
nbproject
src
test
vilij
resources
gui
properties
src
vilij
components
ActionComponent.java
ConfirmationDialog.java
DataComponent.java
Dialog.java
ErrorDialog.java
UIComponent.java
propertymanager
PropertyManager.java
settings
InitializationParams.java
PropertyTypes.java
templates
ApplicationTemplate.java
UITemplate.java
vilij.iml
xmlutil
src
xmlutil
InvalidXMLFileFormatExceptio
XMLUtilities.java
xmlutil.iml

Figure 5.1: DataViLij 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