Software Design Description

for

AJCON, Applet to JSF Converter

Version 1.0

Prepared by

Anıl Sevim

Berkan KISAOĞLU

Özge TOKGÖZ

26.12.2010

Table of Contents

1. Introduction

This report intends to present initial design and progress of the Applet to Java Server Faces (JSF) Converter (AJCON) project, conducted by Team Teaplet. AJCON is supposed to be a software development tool which helps a software developer to migrate from Applet technology to JSF. This report explains initial descriptions of the proposed software system design. In this design document, general design architecture of the project will be enlightened and current project status will be indicated.

1.1. Problem Definition

Java Applets can provide web applications with interactive features that cannot be provided by HTML. When Java enabled browser is used to view a page that contains an applet, the applet's byte codes are transferred to user's system and executed by browser's Java Virtual Machine (JVM). Nowadays, applet technology has become out of date. Meanwhile, with new Java 2 Enterprise Edition (J2EE) technologies, same functional requirements can be met with less dependency.

JSF is one of these technologies, but switching from Applet to JSF requires both lots of money and manpower. Also, it is really long-lasting to write a JSF based application which does the same work with Applet from scratch. Even though there are some converters that may help employees at intermediate levels, there is no existing service, which does this conversion.

1.2. Purpose

The purpose of this document is to explain initial design details of AJCON project. As IEEE standards document indicates, the Initial Design Report show how the proposed software system will be structured in order to satisfy the requirements identified in the Software Requirements Specifications document. In other words, it is aimed to translate software requirements defined in SRS document into a representation of software components, interfaces and data to be used later in implementation phase of the project. However, since every software design is open to changes and modifications, it is highly possible to make changes during implementation and update SRS and SDD documents accordingly.

1.3. Scope

This initial SDD will contain the general definition and features of the project, design constraints, the overall system architecture and data architecture, a brief explanation about our current progress and schedule of the project. With the help of UML diagrams, design of the system and subsystems/modules will be explained visually in order to help the programmer to understand all information stated in this document correctly and easily.

1.4. Overview

This document encompasses a design model with architectural, interface, component level and deployment representations. Design model will be contained in this document, which will be used as a medium for communicating software design information, assessed for quality, improved before code is generated. Many graphical representations and verbal explanations were added to this document to achieve the goal of AJCON.

This document is divided into subsections to make it more understandable. Those are:

Section 2 contains general description about the system components.

Section 3 contains the assumptions made during the design process, dependencies and other constraints.

Section 4 contains general data structures that AJCON used.

Section 5 contains the most important diagrams of the document. Class diagrams, data flow diagrams and sequence diagrams of components are stated in this section. Also a brief explanation about the classes is mentioned.

Section 6 contains the user interface design and some screenshots.

Section 7 contains the detailed design issues and future works.

Section 8 contains the libraries and tools that we will use.

Section 9 contains the basic timeline of the project.

Those sections and subsections of them are mentioned in the table of contents more precisely.

1.5. Definitions, Acronyms and Abbreviations

SDD	Software Design Document
AJCON	Applet to JSF Converter
JVM	Java Virtual Machine
JSF	Java Server Faces
J2EE	Java 2 Enterprise Edition
JavaML	Java Markup Language

1.6. References

[1] IEEE Recommended Practice for Software Design Descriptions

[2] AJCON Software Requirements Specifications Document, v1.0

[3] JavaML – A Markup Language for Java Sources, www.cs.washington.edu/research/constraints/web/badros-javaml-www9.ps.gz

[4] Apache Log4j, logging.apache.org/log4j/

[5] Jikes, jikes.sourceforge.net

[6] Apache Tomcat Wikipedia Page, Wikipedia.org/Apache_Tomcat

[7] Richfaces Community, jboss.org/richfaces

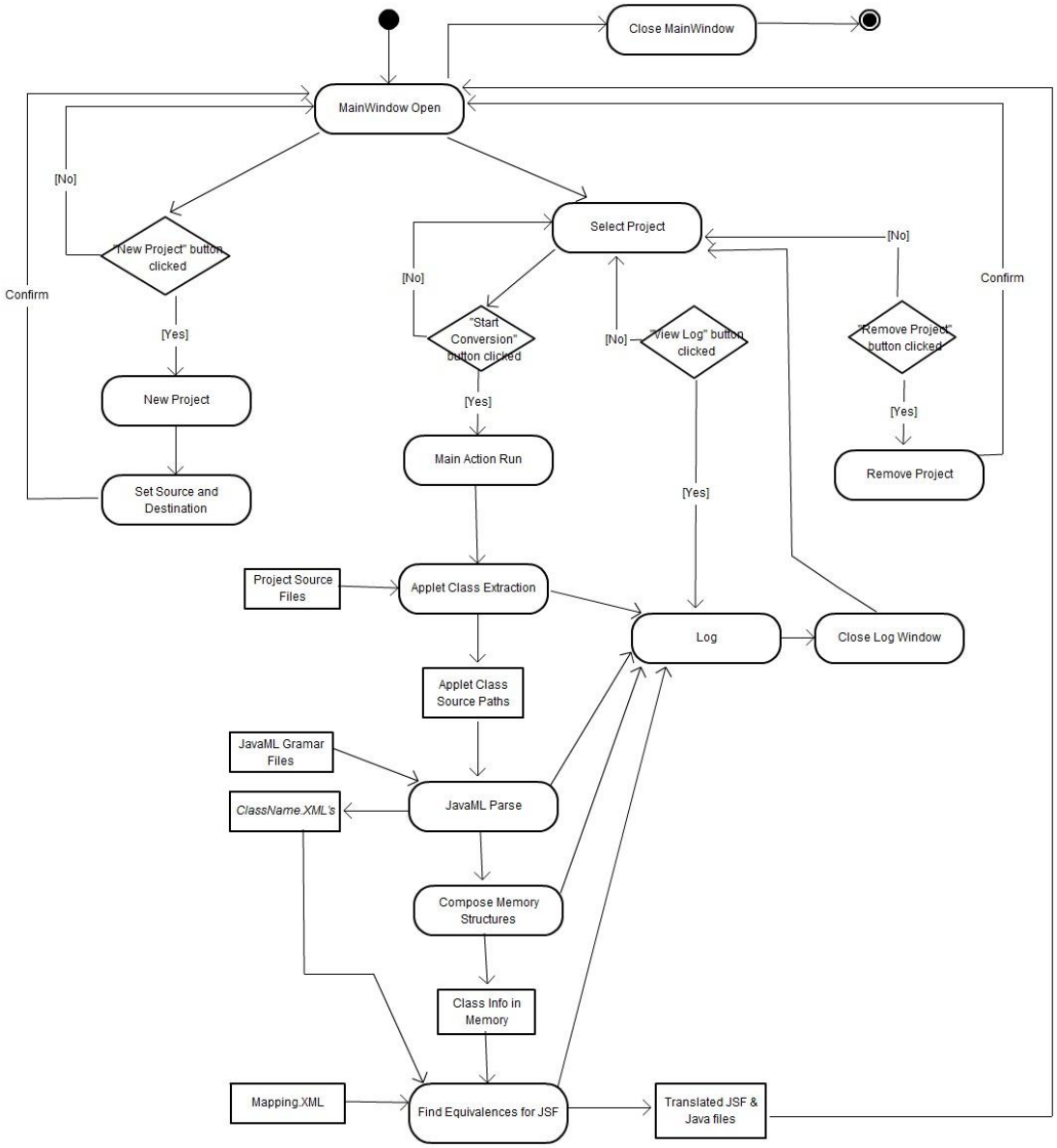
[8] Java Reflection API, http://download.oracle.com/javase/tutorial/reflect/index.html

2. System Overview

Main concern of the AJCON project is to help developers to make their work easy. For an applet project, converting it into a JSF project totally can be costly. With the use of AJCON, cost, man power needs and time needs of converting process can be decreased. It is not possible to convert all the projects with a rate of 100% of correctness but, after the convert operation little changes can raise the output of AJCON up.

In this context, we designed AJCON in a manner stated in section 5.

General description of the system drawn on the activity diagram stated below. Reactions defined on the user interface depends on the users actions, on the other hand, with the start of the conversion operation it is automated. User decides the operation will be done. Those operations can be adding/removing/selecting/deselecting/converting operations. Once converting operation starts, other related things done by AJCON. Finding applets, parsing sources, displaying log information and etc.



3. Design Considerations

3.1. Design Assumptions, Dependencies and Constraints

3.1.1. Design Assumptions

AJCON is a huge project to design and implement. Since we have approximately six months to finish, we are requested by Siemens EC to make some assumptions in order to narrow down project to a certain level.

For initial design, our design assumptions can be stated as:

- This project runs on a Microsoft Windows platform (Vista or later),
- JRE must be installed on running computer,
- Application will be deployed to Apache Tomcat 6.0 or higher server,
- Input Java project should be syntactically correct and runnable.
- Input Java project should include at least one Applet class.
- For final design, inputs will be an Applet embedded html file, but for now, we assume an Applet Desktop Application as input. Later, we will turn to web based ones.
- For start, we will consider converting 8 basic Applet components to JSF. (See Section 4. Data Design)
- We will use JavaML tool [3] for parsing Java source files. Although, this software product is stated to be working for every Java source file, we have to assume that JavaML works properly. It should generate a well-formed and correct XML file, which is a complete self-describing representation of Java source code.

3.1.2. Design Dependencies

For initial design, our design dependencies can be stated as:

- JSF will depend on Java SE 5 (or higher).
- Software should run on a Microsoft Windows platform.

3.1.3. Design Constraints

3.1.3.1. Time

Under the scope of CEng 491-492 courses, we have approximately six months to finish our projects. In order to meet deadlines, we have to obey our schedule strictly. As we mentioned in our SRS document, we will be following agile software development model. Since it is a step-by-step approach, it is a must to update requirements and solutions. According to the feedback we will take, we will improve the general design and process of our project. Thus, we are planning not to fall behind the schedule.

3.1.3.2. Performance

For every software product, performance is an important criteria. Since AJCON project will be run by local clients at Siemens EC and there is no multi-user operation, we expect that conversion from Applet to JSF will end up at most in a few seconds.

3.2. Design Goals and Guidelines

3.2.1. Portability

There will be an installer for AJCON that runs only on Microsoft Windows platform mentioned both in assumptions and dependencies. Although Java ML tool is written in C++, there is only a Microsoft Windows executable publicly available. We are planning to request a Unix platform executable from designer of Java ML tool. If we are able to access that executable, we will make AJCON project portable for every operating system since Java is a machine independent language and works on every platform.

3.2.2. Reliability

Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Responses and the work done by the system should be consistent.

3.2.3. Correctness

AJCON will work correctly if all the requirements and assumptions are met. It will give the same result regardless of time, environment, etc.

4. Data Design

4.1. Data Description

We will keep our data in simple XML files; therefore converting those XML files into data structures in the memory is so simple. Several files are processed during the process of conversion and running of the system. Those are:

4.1.1. Data Objects

4.1.1.1. External Data Objects:

- User defined inputs
- Project input files
- Mapping.xml
- javaml-2.dtd & javaml-2.xsd
- ClassName.xml(Output of the JavaML)
- Output files
- Log files

All the above files except from output files are required to run the system properly. ClassName.xml and output files are constructed during the conversion operation and they are not temporary files. We will keep them to compare the results of the output with the initial sources. Functionalities and structures of those files are described below.

User defined inputs:

User must define source project folder path and destination project folder path from GUI. These data are used to get all project files included in source project folder path and generate output JSF project files in destination project folder path.

Project input files:

Project input files will be specified in run time. User will specify the input files for each project in run time via user interface.

Mapping.xml

```
<MappingElements>
        <MapElement>
                <Object>
                        <Applet> JButton </Applet>
                        <JSF> h:Button </JSF>
                </Object>
```

```
            <Properties>
                <Property type= “message”>
                    <Applet> addActionListener </Applet>
                    <JSF> action </JSF>
                </Property>
                <Property type= “message”>
                    <Applet> setText </Applet>
                    <JSF>   value </JSF>
                </Property>
                <Property type= “message”>
                    <Applet> repaint </Applet>
                    <JSF> rerender </
                </Property>
            </Properties>
        </MapElement>
        <MapElement>
            <Object>
                <Applet> JCheckBox </Applet>
                <JSF> h:selectBooleanCheckbox </JSF>
            </Object>
            <Properties>
                <Property type=“message”>
                    <Applet> addActionListener </Applet>
                    <JSF> value </JSF>
                </Property>
            </Properties>
        </MapElement>
    </MappingElements>
```

This mapping file is an example-mapping file. At initial step, we will not only consider these two components stated in the example mapping file. We will try to convert other components also. All the components that we will try to convert are:

- JButton
- JCheckBox
- JTextField
- JTextArea
- JComboBox
- JLabel
- JRadioButton
- JList

javaml-2.dtd & javaml-2.xsd

javaml-2.dtd and javaml-2.xsd files are reference documents to grammar and lexer rules of JavaML. For further investigations on their hierarchical structures, please see References section for corresponding website link.

ClassName.xml

ClassName.xml is the output file of JavaML. ClassName should be the class name that extends the Applet class. This xml file is automatically generated after JavaML's run on Java source code files of the input Applet project. It conforms to javaml-2.dtd and javaml-2.xsd file structures. This file will be parsed with the help of Mapping.xml by Translator component.

Output Files

Output files are JSF files that have been converted from input Applet project.

4.1.1.2. Internal Data Objects

Internal Data Objects for each component are shown in diagrams in section 5.

4.1.3. Data Models

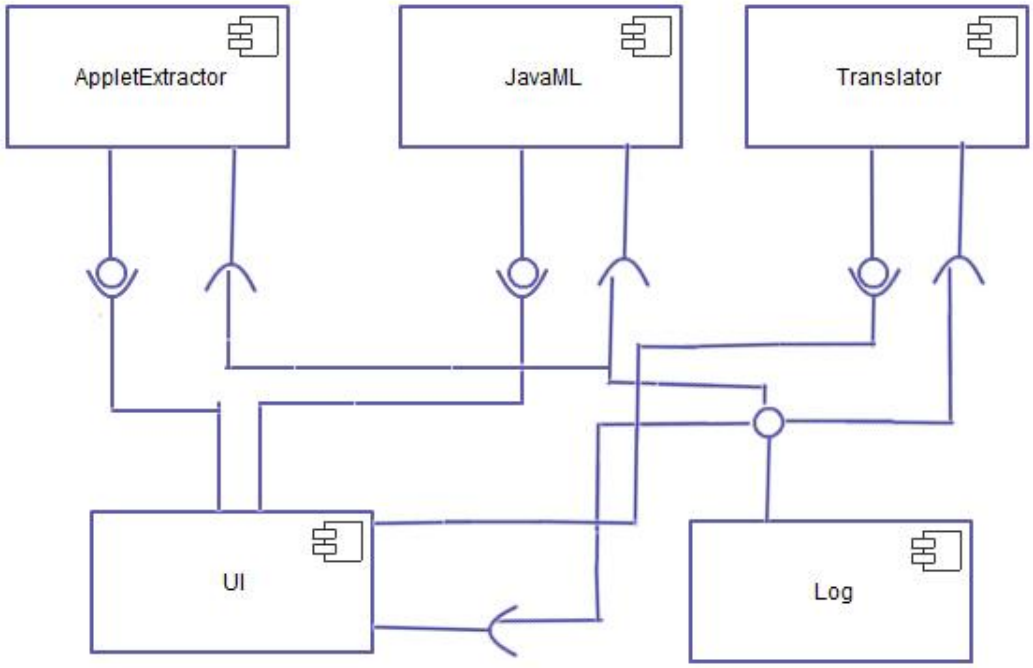
AJCON project does not use database. Therefore, ER Diagram for database modeling is not drawn. For data modeling of the system, data flow diagram is supplied in section 4.1.2.

4.1.4. Data Dictionary

AJCON project does not use database. Therefore, ER Diagram for database modeling is not drawn. For data modeling of the system, data flow diagrams drawn for components are supplied in section 5.2.

5. System Architecture

5.1. Architectural Design



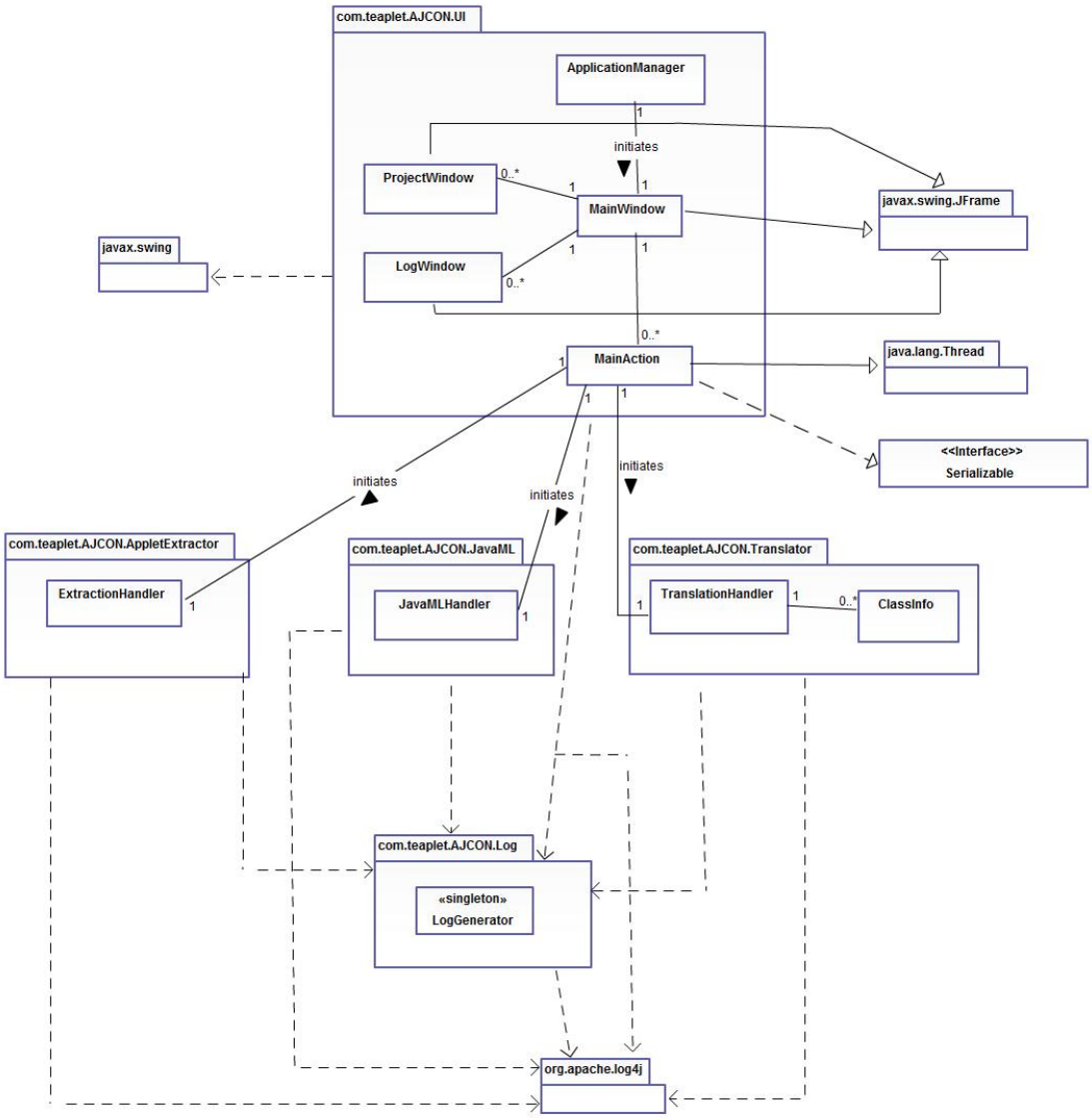
Main concern of AJCON is to convert an Applet project to JSF project. For this purpose AJCON project composed of several components: Log Component, UI Component, Translator Component, JavaML component, Applet Extractor Component.

Those components are interacting with each others. Some of them provide some interfaces to other ones, and some of them use the provided interface. Generally the interfaces provided by the other components are the methods of the classes in it.

Above relations shows that Log component provides an interface to other components and all the other components uses it. By the same way, it is shown that UI Component uses all the interfaces provided in the system. All the existing interfaces and the relations between the components are on the diagram.

5.2. Description of Components

Below there is the package diagram of the overall system. Each package/component will be described in subsections.



5.2.1. UI Component

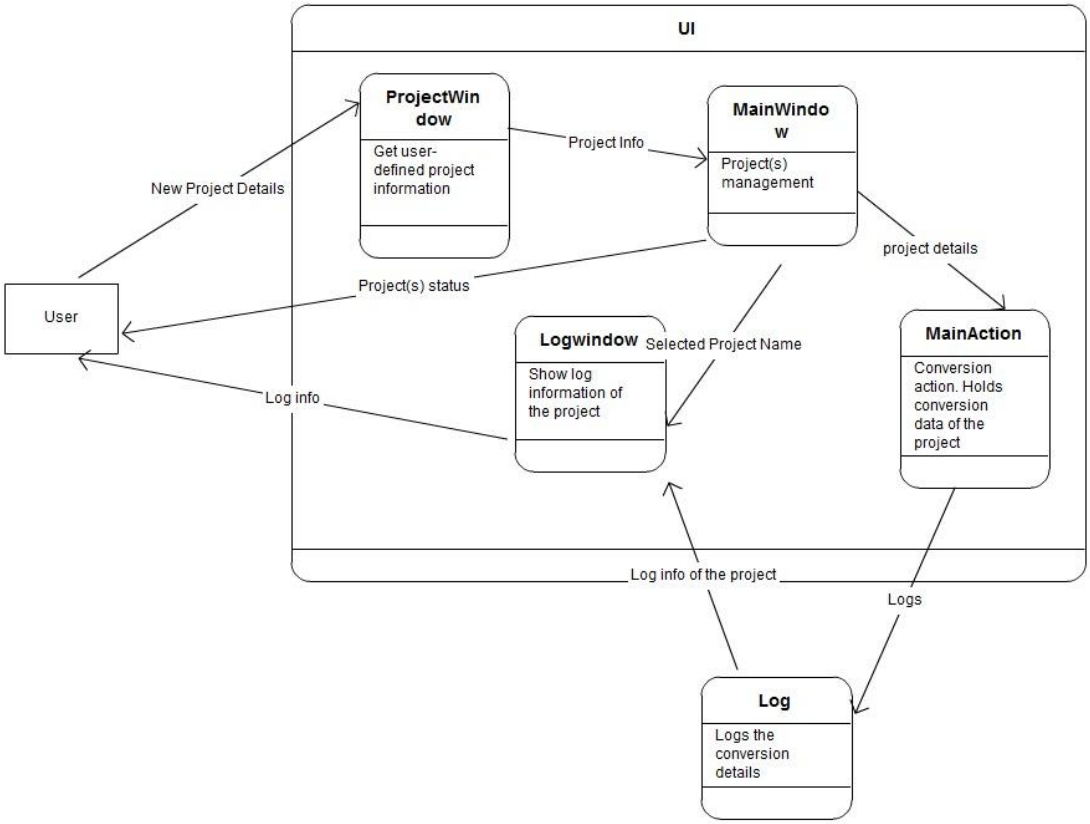
5.2.1.1. Processing Narrative for UI Component

This is the component which interacts with user. Since our project does not require lots of user interactions, this component is not complex structured. It has a simple interface and simple purpose. By this component, user can manage projects to be converted with add/remove/select/startConversion options.

At the beginning, “ApplicationManager” class which has the main function initiates the system and shows the user UI main window. Then, when user clicks the “New Project” button, “Project Window” will pop up. By selecting destination and source folder, user

adds project to list of project to be converted in main window. User can manage the main window by adding or removing projects with this method and start the conversion of any project that he/she selects. After starting a conversion, user can watch the live continuation of conversion process from main window and see logs.

5.2.1.2. Interface Description of UI Component



5.2.1.3. Processing Detail of UI Component

UI component consists of 5 different classes.

5.2.1.3.1. ApplicationManager Class

This class has the main function of the project. It initiates run of the project and sets MainWindow.

5.2.1.3.1.1. Attributes

- public static MainWindow mainWin: This instance variable is set by Application Manager in main function of the project.

5.2.1.3.1.2. Methods

- public static void main (String[] args): This is the main function of the project. When the project runs, this function is called automatically. In this function, main window will be created and system will be initiated.

5.2.1.3.2. MainWindow Class

This is the window that the user can directly manage all conversion operations. This class extends javax.swing.JFrame class and uses javax.swing components for GUI.

5.2.1.3.2.1. Attributes

- private int[] selectedProjects: This keeps id numbers of the projects that user selected.
- private static ArrayList<MainAction> mainActionList: List of main actions for each project thread.
- private static ArrayList<LogWindow> logWindowList: Keeps list of log windows that user wants to see.

- private javax.swing.JSeperator: Seperator between top-level labels and values.
- private javax.swing.JPanel panel: Contains javax.swing GUI components.
- private const javax.swing.JLabel labelProjectName: Constant header label, set as “Project Name” at first.
- private const javax.swing.JLabel labelSourceFolder: Constant header label, set as “Source Folder” at first.
- private const javax.swing.JLabel labelCreateDate: Constant header label, set as “Create Date” at first.
- private const javax.swing.JLabel labelProgressBar: Constant header label, set as “Progress” at first.
- private const javax.swing.JLabel labelSelected: Constant header label, set as “Selected” at first.
- private javax.swing.JButton buttonNewProject: User can create a new project by pressing this button.
- private javax.swing.JButton buttonRemoveProject: User can remove selected project(s) from the list by pressing this button.
- private javax.swing.JButton buttonStartConversion: User can start conversions of the selected project(s) by pressing this button.
- private javax.swing.JButton buttonViewLog: User can see log(s) of the selected project(s) by pressing this button.
- private static ArrayList<javax.swing.JLabel> listProjectNames: This instance variable keeps names of the projects in the main window.
- private static ArrayList<javax.swing.JLabel> listSourceFolders: This instance variable keeps source folder paths of the projects in the main window.
- private static ArrayList<javax.swing.JLabel> listCreateDates This instance variable keeps creation dates of the projects in the main window.:
- private static ArrayList<javax.swing.JProgressBar> listProgressBars: This instance variable keeps progress bar info of the projects in the main window.
- private static ArrayList<javax.swing.JCheckBox> listCheckBoxes: This instance variable keeps checbox’s status for each project in the main window.

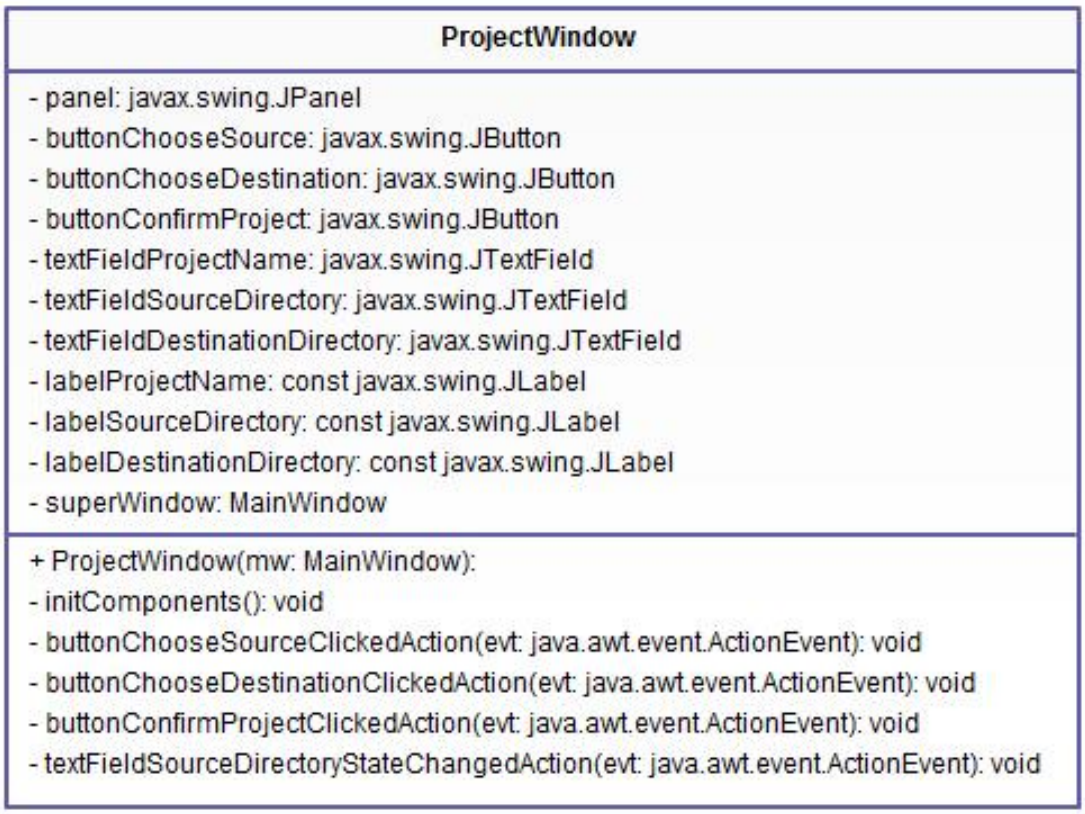
- private org.apache.log4j.Logger logger: This variable is used to log any kind of information inside this class.

5.2.1.3.2.2. Methods

- public MainWindow(): Constructor of the MainWindow class.
- public void initComponents(): Initializes interface components.
- public ArrayList<javax.swing.JLabel> getProjectNames(): Returns list of project names.
- public ArrayList<javax.swing.JLabel> getSourceFolders(): Returns list of source folder paths of the projects.
- public ArrayList<javax.swing.JLabel> getCreateDates(): Returns list of creation dates of the projects.
- public ArrayList<javax.swing.JProgressBar> getProgressBars(): Returns list of progress bar info of the projects.
- public ArrayList<javax.swing.JCheckBox> getCheckBoxes(): Returns list of check box's statuses of the projects.
- public ArrayList<LogWindow> getLogWindows(): Returns list of log windows that user wants to see.
- private void buttonNewProjectClickedAction (java.awt.event.ActionEvent evt, MainWindow mw): When user clicks "New Project", information related to project taken from project window is used as parameter and this function is called.
- private void buttonRemoveProjectClickedAction (java.awt.event.ActionEvent evt): When user clicks "Remove Project", this function is called.
- private void buttonStartConversionClickedAction (java.awt.event.ActionEvent evt): When user clicks "Start Conversion", this function is called.
- private void buttonViewLogClickedAction (java.awt.event.ActionEvent evt): When user clicks, log window(s) open and shows log info to user.

5.2.1.3.3. ProjectWindow Class

This is the class that lets user add a new project with a new window. This class extends javax.swing.JFrame class and uses javax.swing components for GUI.



5.2.1.3.3.1. Attributes

- private javax.swing.JPanel panel: The panel that keeps objects in project window together.
- private javax.swing.JButton buttonChooseSource: Button that is used for choosing source folder.
- private javax.swing.JButton buttonChooseDestination: Button that is used for choosing destination folder.
- private javax.swing.JButton buttonConfirmProject: Button that is used for confirming project conversion.
- private javax.swing.JTextField textFieldProjectName: Text field object that is used for entering project name .

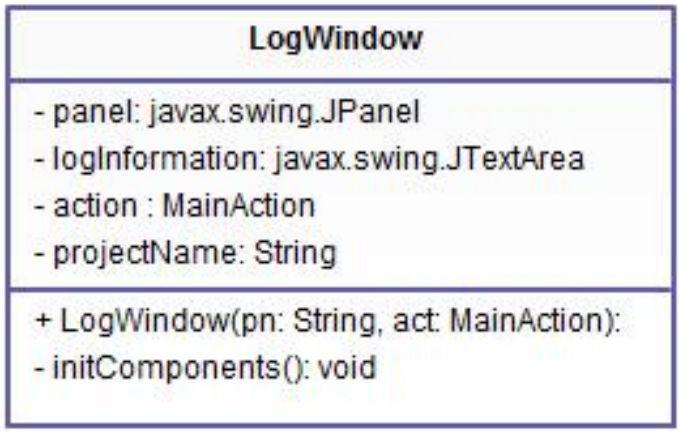
- private javax.swing.JTextField textFieldSourceDirectory: Text field object that is used for entering source directory.
- private javax.swing.JTextFiled textFiledDestinationDirectory: Text field object that is used for entering destination directory.
- private javax.swing.JLabel labelProjectName: Label of project name, that is “Project Name”.
- private javax.swing.JLabel labelSourceDirectory: Label of source directory, that is “Source Directory”.
- private javax.swing.JLabel destinationDirectory: Label of destination directory, “Destination Directory”.
- private MainWindow superWindow: Reference for main window object instance.

5.2.1.3.3.2. Methods

- public ProjectWindow (MainWindow mw): Constructor of ProjectWindow class. Sets mw:MainWindow as its super class object.
- private void initComponents (): Initiates object’s project window components.
- private void buttonChooseSourceClickedAction (java.awt.event.ActionEvent evt): Event handler for clicking “Choose Source” button.
- private void buttonChooseDestinationClickedAction (java.awt.event.ActionEvent evt): Event handler for clicking “Choose Destination” button.
- private void buttonConfirmProjectClickedAction (java.awt.event.ActionEvent evt): Event handler for clicking “Confirm Project” button.
- private void textFieldSourceDirectoryStateChangedAction (java.awt.event.ActionEvent evt): Event handler for text field source directory.

5.2.1.3.4. LogWindow Class

This class shows log information that it takes from Logger object and shows it to user. This class extends javax.swing.JFrame class and uses javax.swing components for GUI.



5.2.1.3.4.1. Attributes

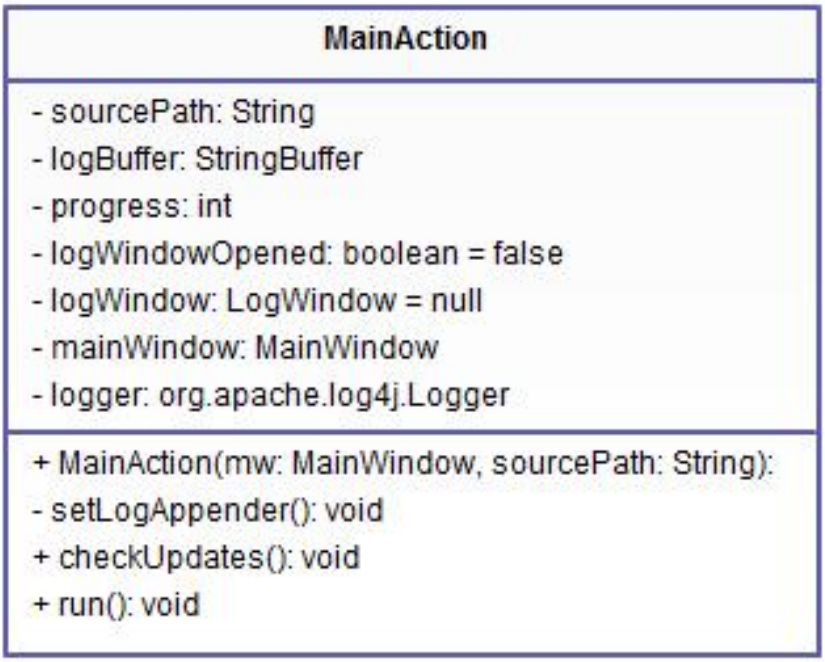
- private javax.swing.JPanel panel: The panel that keeps objects in log window together.
- private javax.swing.JTextArea logInformation: Text area field for log information.
- private MainAction action: Reference for main action object instance.
- private String projectName: Shows name of the project that are being logged.

5.2.1.3.4.2. Methods

- public LogWindow (String pn, MainAction act): Constructor of LogWindow.
- private void initComponents(): Initiates log window components.

5.2.1.3.5. MainAction Class

When user clicks “Start Conversion”, one instance of this class is instantiated for every project and it starts to run. This class also extends Thread and implements Serializable because it uses a multi-threaded approach for every single project run. It lets user to convert several projects at a time.



5.2.1.3.5.1. Attributes

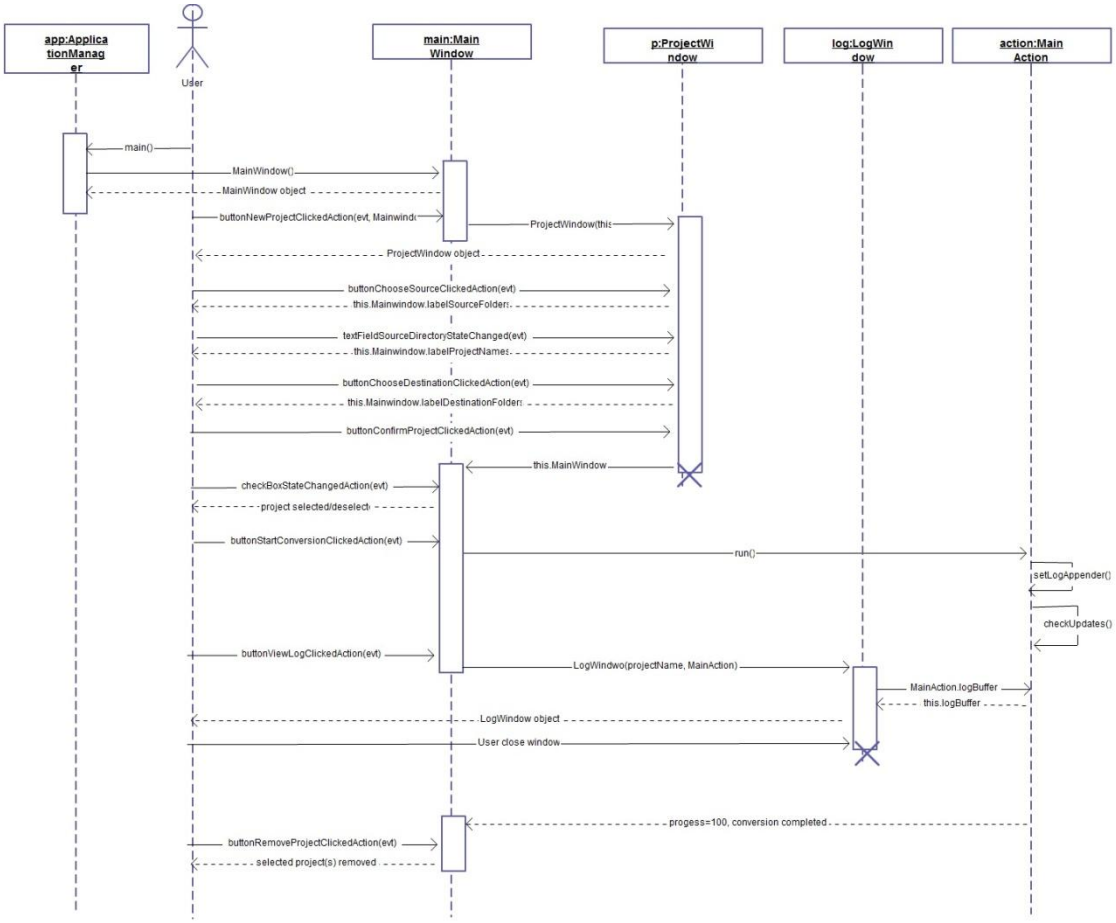
- private String sourcePath: Keeps project source path.
- private StringBuffer logBuffer: The StringBuffer object for logging continuously.
- private int progress: Keeps percentage of the project. Between 0-100.
- private boolean logWindowOpened=false: Boolean value for log window. If open, it is updated in real-time.
- private LogWindow logWindow=null: Reference for LogWindow object instance.
- private MainWindow mainWindow: Reference for MainWindow object instance.
- private org.apache.log4j.Logger logger: Singleton object reference for only one Logger object instance.

5.2.1.3.5.2. Methods

- public MainAction (MainWindow mw, String sourcePath): Constructor of MainAction class.

- private void setLogAppender(): Initiates format of the logger and type of buffer for project.
- public void checkUpdates(): Refreshes the screens.
- public void run(): Function that is needed to be called for thread's start.

5.2.1.4. Dynamic Behavior of UI Component

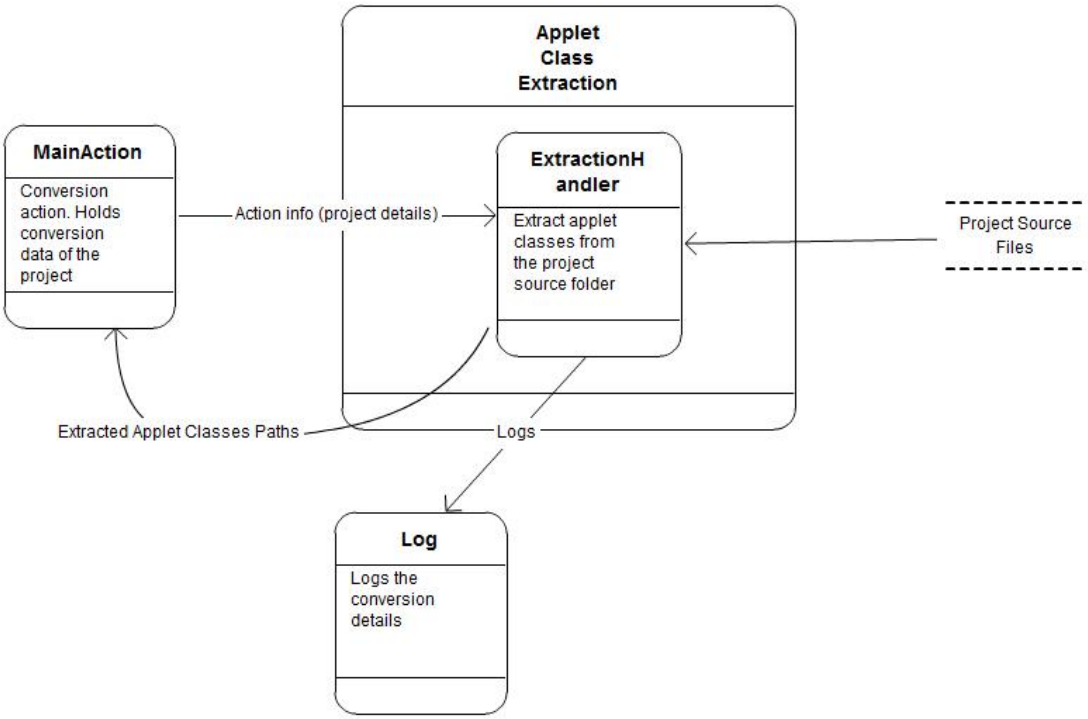


5.2.2. AppletExtractor Component

5.2.2.1. Processing Narrative for AppletExtractor Component

AppletExtractor Component is responsible from finding java sources that extends JApplet class. When the MainAction class is invoked from the user interface, MainAction class constructs an ExtractionHandler in Applet Extractor Component. This component searches the project folder into the deep, and looks all the files in the folders. Component notes down the source files that extend JApplet.

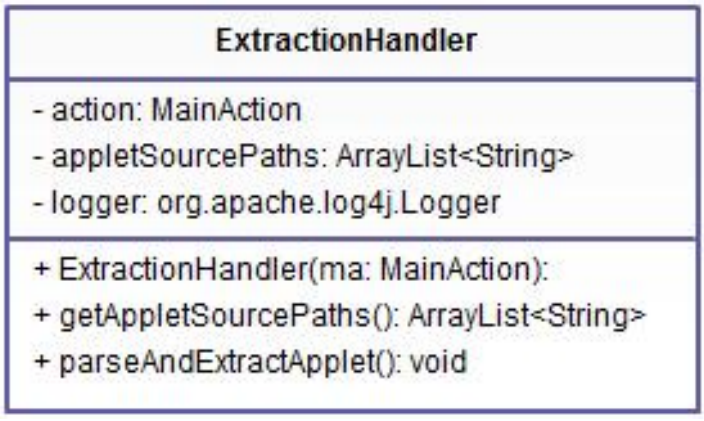
5.2.2.2. Interface Description of AppletExtractor Component



5.2.2.3. Processing Detail of AppletExtractor Component

AppletExtractor component has only one class: ExtractionHandler.

5.2.2.3.1. ExtractionHandler Class



5.2.2.3.1.1. Attributes

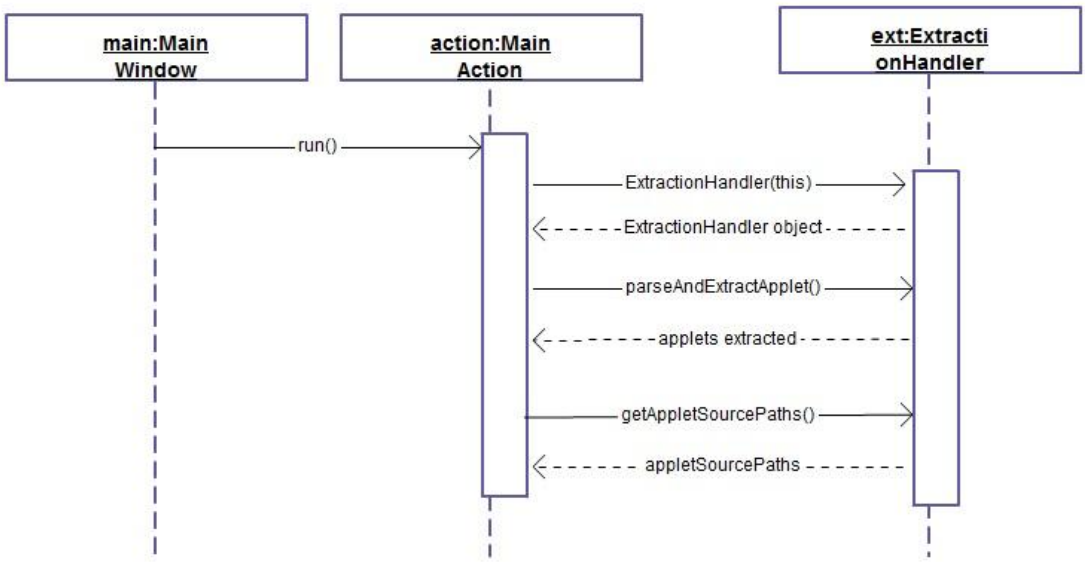
- private MainAction action: Reference to an instance of MainAction class.

- private ArrayList<String> appletSourcePaths: When the class finds a source pushes the file path to list.
- private org.apache.log4j.Logger logger: Singleton object reference for only one Logger object instance.

5.2.2.3.1.2. Methods

- public ExtractionHandler (MainAction ma): Constructor of ExtractionHandler.
- public ArrayList<String> getAppletSourcePaths(): Getter method for field appletSourcePaths.
- public void parseAndExtractApplet():Looks into to deeps of project folder to find source files, which extends JApplet.

5.2.2.4. Dynamic Behavior of AppletExtractor Component



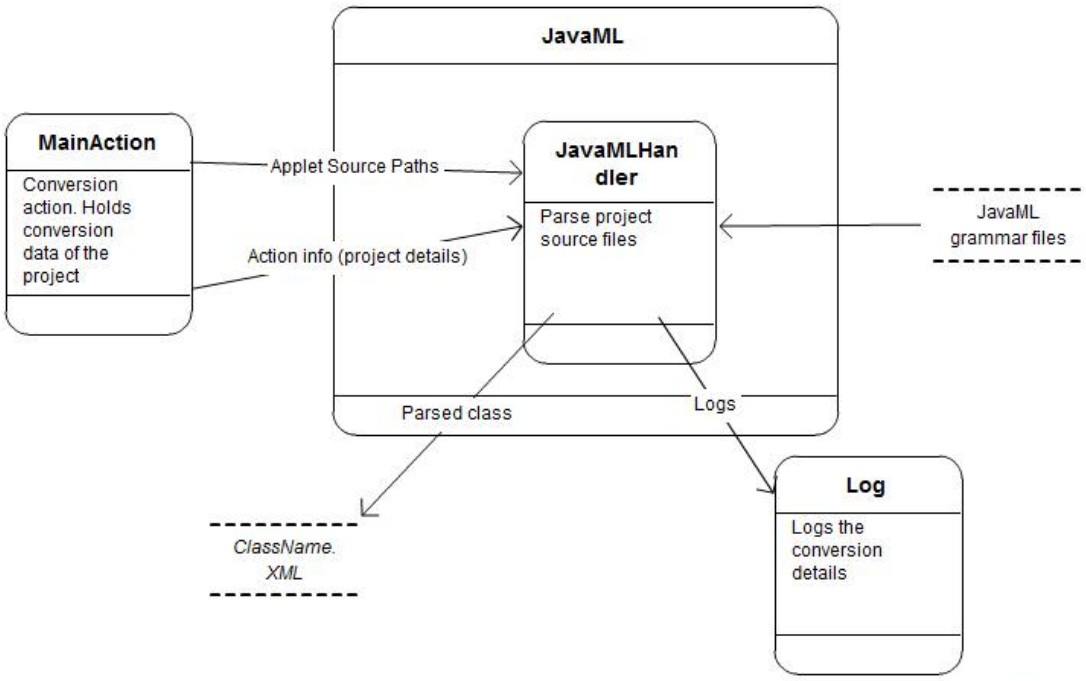
5.2.3. JavaML Component

5.2.3.1. Processing Narrative for JavaML Component

JavaML Component is responsible from lexical analysis and tokenizing the source files. After the process of Applet Extractor Component finishes, MainAction class inititates a JavaMLHandler object. JavaMLHandler object gathers the paths of the source files,

which extends JApplet, from the ExtractionHandler object. After gathering those paths runs Jikes over them.

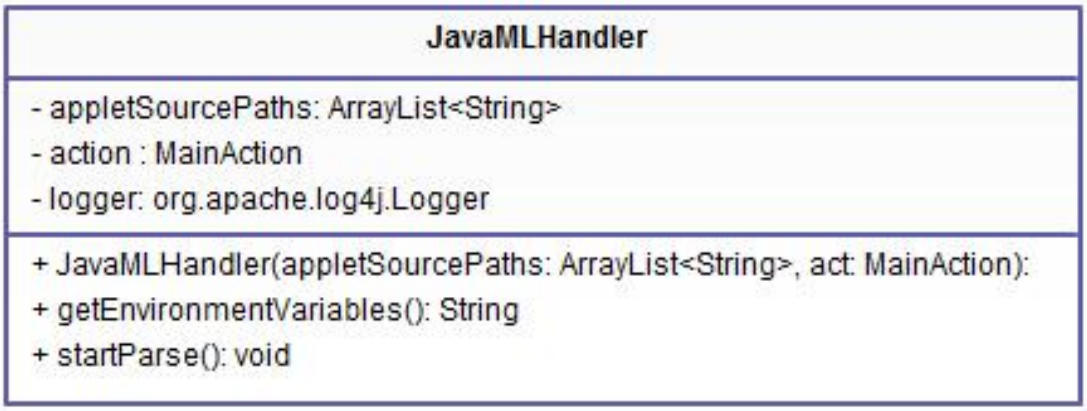
5.2.3.2. Interface Description of JavaML Component



5.2.3.3. Processing Detail of JavaML Component

JavaML component consists of only one class: JavaMLHandler.

5.2.3.3.1. JavaMLHandler Class



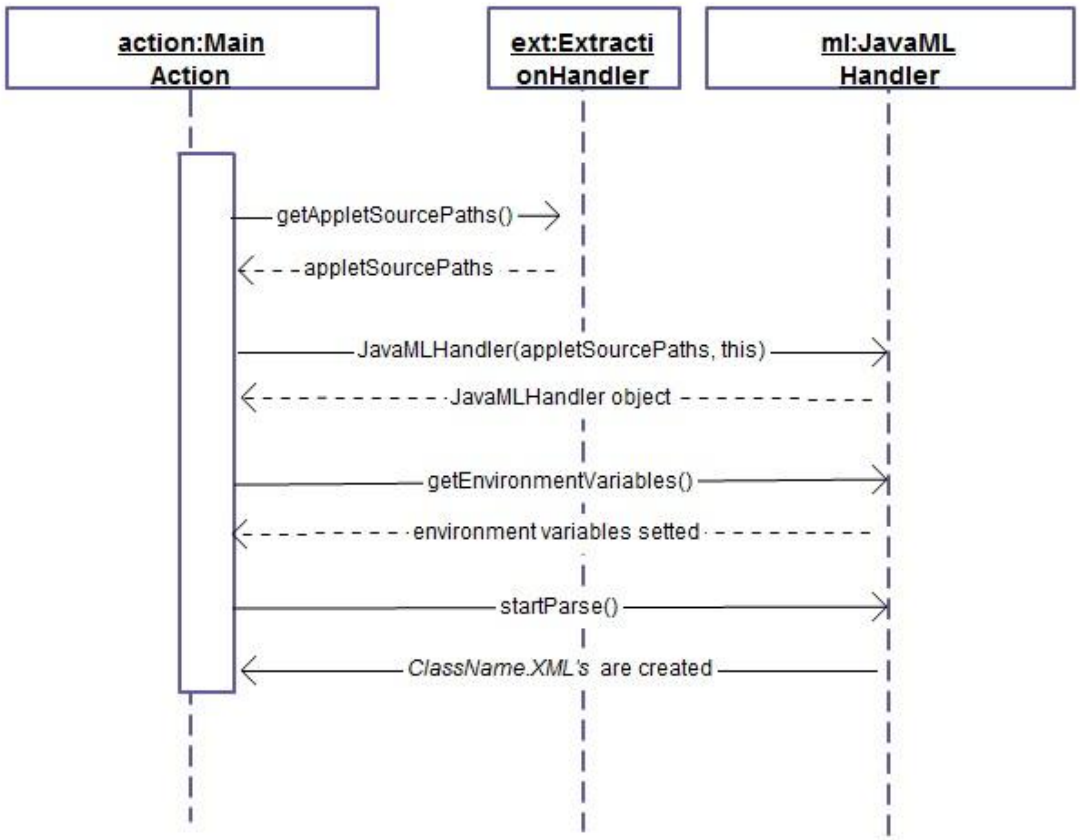
5.2.3.3.1.1. Attributes

- private ArrayList<String> appletSourcePaths: Gathered path information from the ExtractionHandler object.
- private MainAction action: Reference to MainAction instance.
- private org.apache.log4j.Logger logger: Singleton object reference for only one Logger object instance.

5.2.3.3.1.2. Methods

- public JavaMLHandler (ArrayList<String> appletSourcePaths, MainAction act): Constructor for JavaMLHandler class.
- public String getEnvironmentVariables (): Gets the environment variables defined on the system to look for JDK path.
- public void startParse(): Runs JavaML/Jikes over the files.

5.2.3.4. Dynamic Behavior of JavaML Component



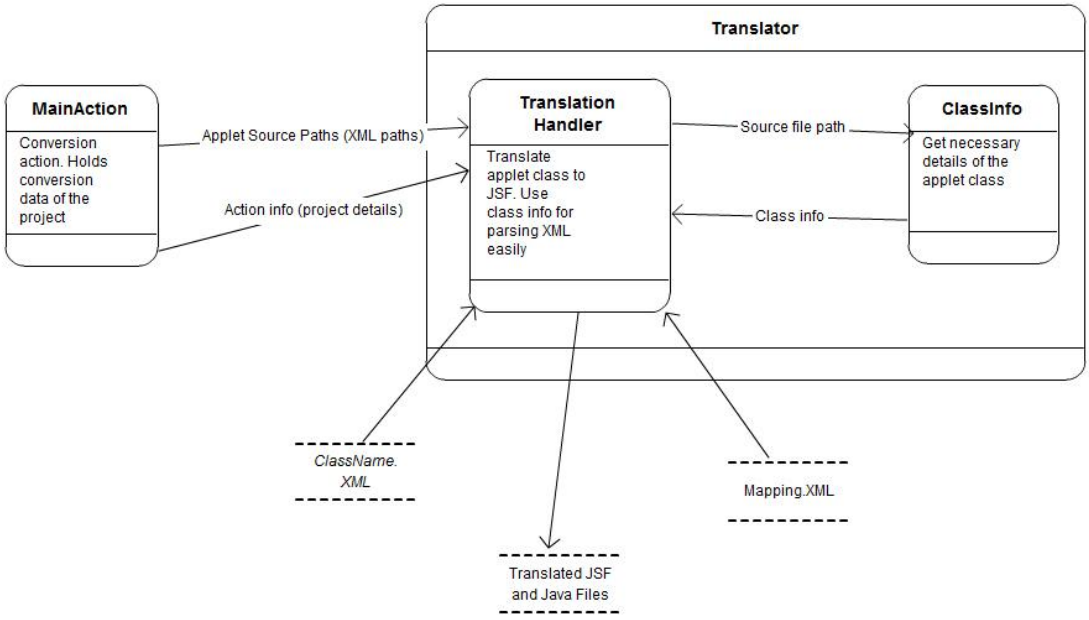
5.2.4. Translator Component

5.2.4.1. Processing Narrative for Translator Component

Translator component uses output of JavaML component – that is ClassName.xml, related ClassInfo object instances and Mapping.xml file in order to generate output files. In this design, we may use Java Reflection API instead ClassInfo objects in detailed design. More information can be found at Section 8 6. Java Reflection API.

This component is going to be instantiated at MainAction class and be triggered from there.

5.2.4.2. Interface Description of Translator Component



5.2.4.3. Processing Detail of Translator Component

Translator component consists of only one class: TranslationHandler.

5.2.4.3.1. TranslationHandler Class

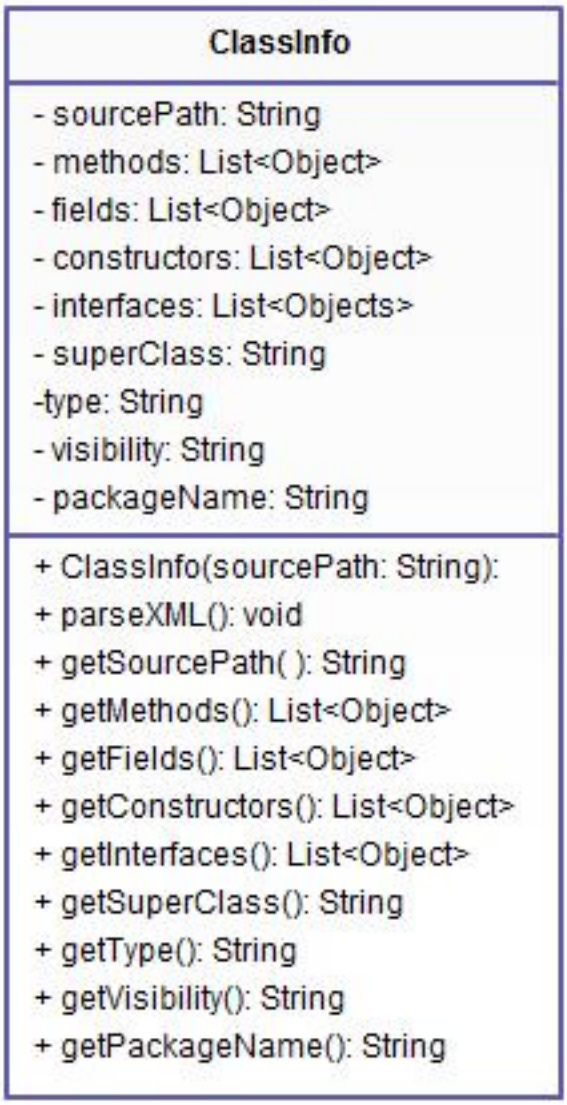
5.2.4.3.1.1. Attributes

- private ArrayList<ClassInfo> listClassInfo: Keeps ClassInfo object instances.
- private MainAction action: Reference for MainAction object instance.
- private ArrayList<String> appletSourcePaths: Keeps paths of java class files which extends JApplet class.
- private org.apache.log4j.Logger logger: Singleton object reference for only one Logger object instance.

5.2.4.3.1.2. Methods

- public TranslationHandler (ArrayList<String> appletSourcePaths, MainAction act): Constructor for TranslationHandler class.
- public void composeMemoryStructure (): Generates ClassInfo objects in memory.
- private void findEquivalentJSF (String filename): Uses Mapping.xml to compare and generate output JSF tags.
- private void write2JSF (String filename): Output stream writer for output JSF files.
- public void findEquivalences(): Interface for MainAction class. Calls findEquivalentJSF and write2JSF.

5.2.4.3.2. ClassInfo Class



5.2.4.3.2.1. Attributes

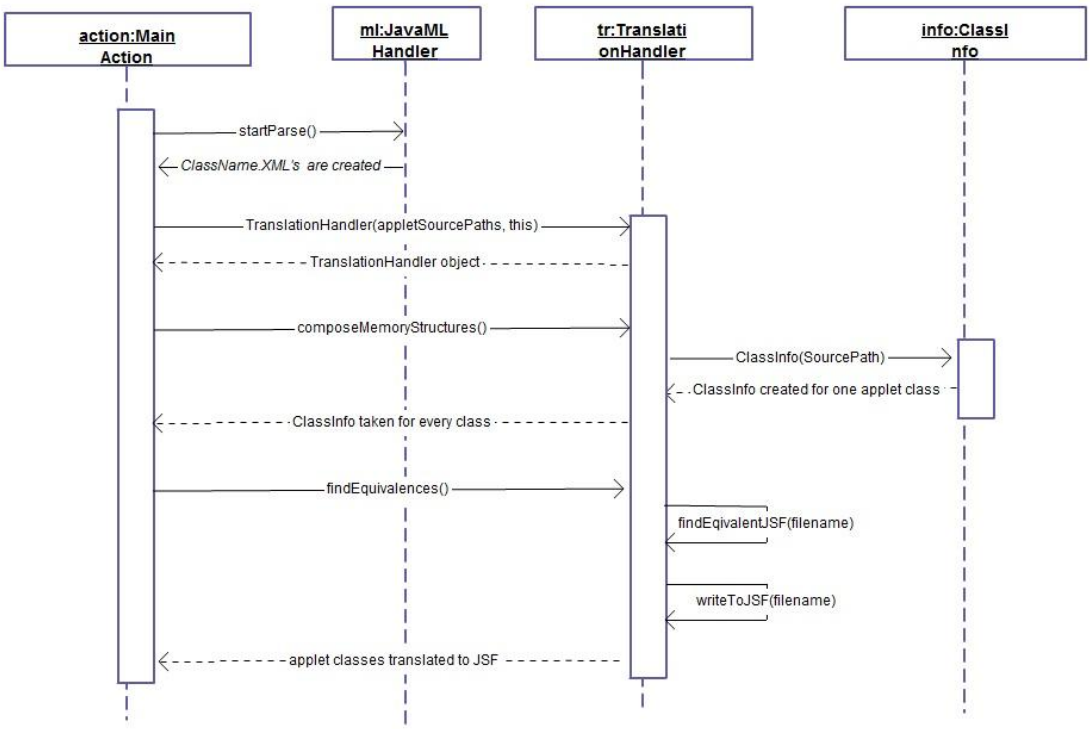
- private String sourcePath: Path of the source file which extends JApplet.
- private List<Object> methods: Method list of the source file which extends JApplet.
- private List<Object> fields: Field list of the source file which extends JApplet.
- private List<Object> constructors: Defined constructors on the source file which extends JApplet.
- private List<Object> interfaces: List of the interfaces that class implements.
- private String superClass: Name of the super class.
- private String type: Type of the class: Abstract…

- private String visibility: Accessibility of the class: public, private
- private String packageName: Package of the class.

5.2.4.3.2.2. Methods

- public ClassInfo (String sourcePath): Constructor for the class ClassInfo. It may be constructed with Java Reflection API in future.
- public void parseXML(): Parses the output of the JavaML.
- public String getSourcePath(): Getter method for the field “sourcePath”.
- public Object getMethods(): Getter method for the field “methods”.
- public Object getFields(): Getter method for the field “fields”.
- public Object getConstructors(): Getter method for the field “constructor”.
- public Object getInterfaces(): Getter method for the field “interfaces”.
- public String getSuperClass(): Getter method for the field “superClass”.
- public String getType(): Getter method for the field “type”.
- public String getVisibility(): Getter method for the field “visibility”.
- public String getPackageName(): Getter method for the field “packageName”.

5.2.4.4. Dynamic Behavior of Translator Component



5.2.5. Log Component

5.2.5.1. Processing Narrative for Log Component

Log component is responsible from only logging. There will be only one logger while the system is running. Logger Component will be accessible from all the other components to log appropriate information. Logger will be configured to log different places for each project. It will log into a file named projectName.log and also, it will produce logs on the screen.

Apache log4j library will be used while logging.

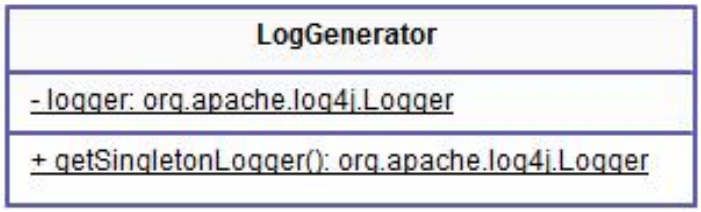
5.2.5.2. Interface Description of Log Component

Log component is not a complex component and there is no complex data flow over the component. Data flow of the Log component described in other components data flow diagrams.

5.2.5.3. Processing Detail of Log Component

Log component consists of only one class: LogGenerator.

5.2.5.3.1. LogGenerator Class



5.2.5.3.1.1. Attributes

- private static org.apache.log4j.Logger logger: Singleton logger object.

5.2.5.3.1.2. Methods

- public org.apache.log4j.Logger getSingletonLogger(): Getter method for the “logger” field.

5.2.5.4. Dynamic Behavior of Log Component

All the other components send log information after all the operations by done the component. So there is no need to show the sequence of the flow in this section. Any component can log any time.

6. User Interface Design

6.1. Overview of User Interface

In this project, there will be no complex user interfaces. This tool will be a single developer tool; in fact there will be no user interface requirements. Running it on command line will be enough.

Our designed user interfaces provide some facilities to users. When the user starts to use the system, main window stated in part 6.2 welcomes the user.

Capabilities of the main window are to:

- Operate over the existing projects
    - Remove an existing project
    - Select an existing project
    - Deselect an existing project
    - Start conversion of selected projects
    - View log information of selected projects
- Add new project

All those operations mentioned above are the directly user related operations. Actions of the user will be converted to system functions related to that action.

This project does not contain a main window only. According to user actions, some other pre-defined user interfaces will appear on the window. When the main window is opened and the user wants to add a new project, another user interface will appear which is stated in 6.2.

Capabilities of the “Project” window are to:

- Select a project folder
- Select a destination folder
- Confirm project details

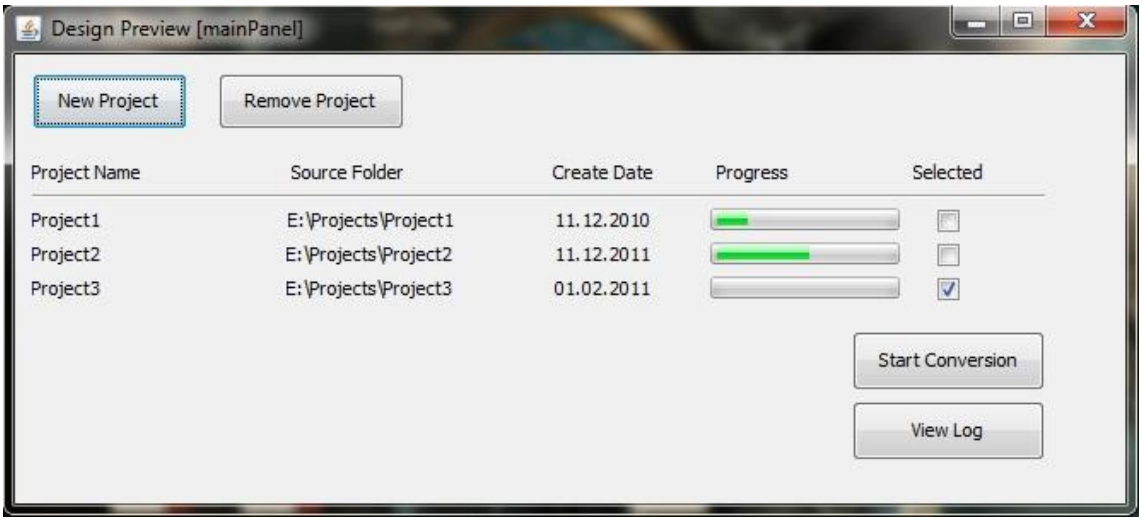
Another window that can be seen via main window is log information window.

Capabilities of the “Log” window are;

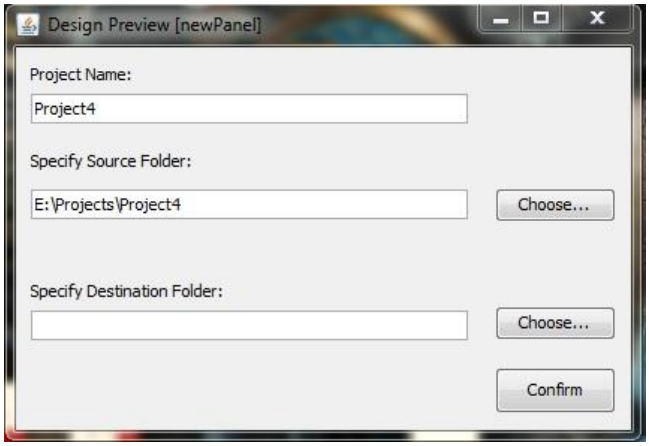
- Display real time information about the project being converted.

All the information stated above is directly from the users perspective. In addition to those, there are some other internal operations that invoke the user interface. According to the conversion process user interface shows the percentage of the conversion.

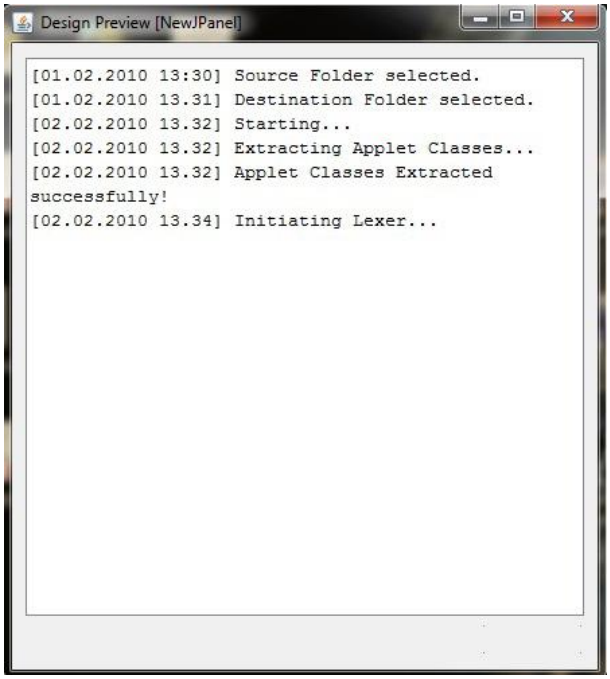
6.2. Interface Screens



Main Window



Project Window



Log Window

6.3. Screen Objects and Actions

This part includes objects on the screen interfaces and the actions linked to that objects.

6.3.1 Screen Objects

For the main window:

- Panel: Panel is to group other objects in the window. There will be only one panel to group objects.
- Buttons
    - buttonNewProject: This button is to add a new project to convert.
    - buttonRemoveProject: This button is to remove an existing project.
    - buttonStartConversion: This button is to start conversion operation of selected projects.
    - buttonViewLog: This button is to view log information.

There are actions linked to those buttons. All the actions are stated below in section 6.3.2.

- Labels
    - labelProjectName: Label for the project name.
    - labelSourceFolder: Label for the source project folder.
    - labelCreateDate: Label for the creation date of the project.
    - labelProgressBar: Header label for the progress bars.
    - labelSelected: Header labels for the checkboxes defined below.

Those labels are the headers. According to the existing projects, there will be some other labels related with each project under above header labels.

- Progress Bars: Progress bars are to show the status of the conversion operation.
- Check Boxes: Checkboxes are to select or deselect a project to operate on it.
- Separator: Separates the headers from the project information.

Progress bars and Check boxes can be more than one according to existing projects. Also there are some actions linked to those checkboxes.

For the project window:

- Panel: Panel is to group other operations on the window.
- Buttons
    - buttonChooseSource: This button is to opens a standard dialog window to select the source folder.
    - buttonChooseDestination: This button is to opens a standard dialog window to select the destination folder.
    - buttonConfirmProject: This button is to confirm project details stated.
- Text Fields
    - textFieldProjectName: This text field is for to specify project name. It is a disabled field and automatically generated with the selected source directory.
    - textFieldSourceDirectory: This text field is to specify source directory. It is an enabled component and also automatically generated with the selection of source directory.
    - textFieldDestinationDirectory: This text field is to specify destination directory. It is an enabled component and also automatically generated with the selection of destination directory.

There are actions defined on the objects. Those actions are described in section 6.3.2.

- Labels
    - labelProjectName: Label for the textFieldProjectName .
    - labelSourceDirectory: Label for the textFieldSourceDirectory.
    - labelDestinationDirectory: Label for the textFieldDestinationDirectory.

For the log :

- Panel: Panel is to group another objects together.
- TextArea: TextArea component is to show log information about the process.

6.3.2 Screen Actions and Relations

Defined actions for the interfaces stated below.

For the “Main” window:

- Actions of Buttons
    - buttonNewProjectClickedAction: Action performed when the buttonNewProject button clicked on the main window. Opens “Project” window stated in section 6.2.
    - buttonRemoveProjectClickedAction: Action performed when the buttonRemoveProject button clicked on the main window. Removes the selected project from the list of existing projects.
    - buttonStartConversionClickedAction: Action performed when the buttonStartConversion button clicked on the main window. Starts the main operation conversion of the selected projects.
    - buttonViewLogClickedAction: Action performed when the buttonViewLog button clicked. Opens “log” window, which is stated in section 6.2.
    - checkboxStateChangedAction: Selects or deselects a project from the existing projects.

For the “Project” Window:

- Actions of Buttons
    - buttonChooseSourceClickedAction: Action performed when the buttonChooseSource button clicked in the “Project” window. Action opens a dialog window that contains the system directories to choose source folder.
    - buttonChooseDestinationClickedAction: Action performed when the buttonChooseDestination button clicked in the “Project” widow. Action opens a dialog window which contains the system directories to choose destination folder.
    - buttonConfirmProjectClickedAction: Action performed when the buttonConfirmProject button clicked in the “Project” window. Action closes the current “Project Window” and adds the new project to list of existing projects in the main window.

- textFieldSourceDirectoryStateChangedAction: Action performed when the state of the textFieldSourceDirectory changed. State of the textFieldSourceDirectory object changes with if any user enters a text. With the change of the state of textFieldSourceDirectory textFieldProjectName field will be automatically generated.

7. Detailed Design

Necessary details of each design entity/component (classifications, definitions, responsibilities, constraints, user-interactions, interfaces, data flows, interconnections etc.) for Initial Design Report level were given in above sections. More detailed explanations will be made in Detailed Design Report. These detailed information contains exceptions, constants, more detailed data-design, lower-level components, detailed-algorithms etc.

It should be noted that ClassInfo class ,which will be used for holding class general structure, is written roughly. In Detailed Design Report, data types for Object's will be determined, maybe by using Java Reflection API (see section 8.6).

8. Libraries and Tools

8.1. JavaML [3]

The Java Markup Language (JavaML) [4] builds a bridge between Java and XML. It generates a self-describing representation of Java source code. Its nested representation in XML-based syntax directly reflects the structure of software artifact. It has many advantages because since XML is a text-based representation, it still keeps the classical source representation. XML files are also very easy to parse with external Java parsers (Apache Xerxes DOM, SAX etc.)

JavaML is defined by document type definition (DTD) in [4]. In JavaML, concepts such as methods, superclasses, message sends and literal numbers are all directly represented in the elements and attributes of the document contents. The representation reflects the structure of the programming language in the nesting of the elements.

In our project, we will use JavaML in order to parse Java source code and generate corresponding XML file. It will enable us to see hierarchical structure of Java classes and create mapping file.

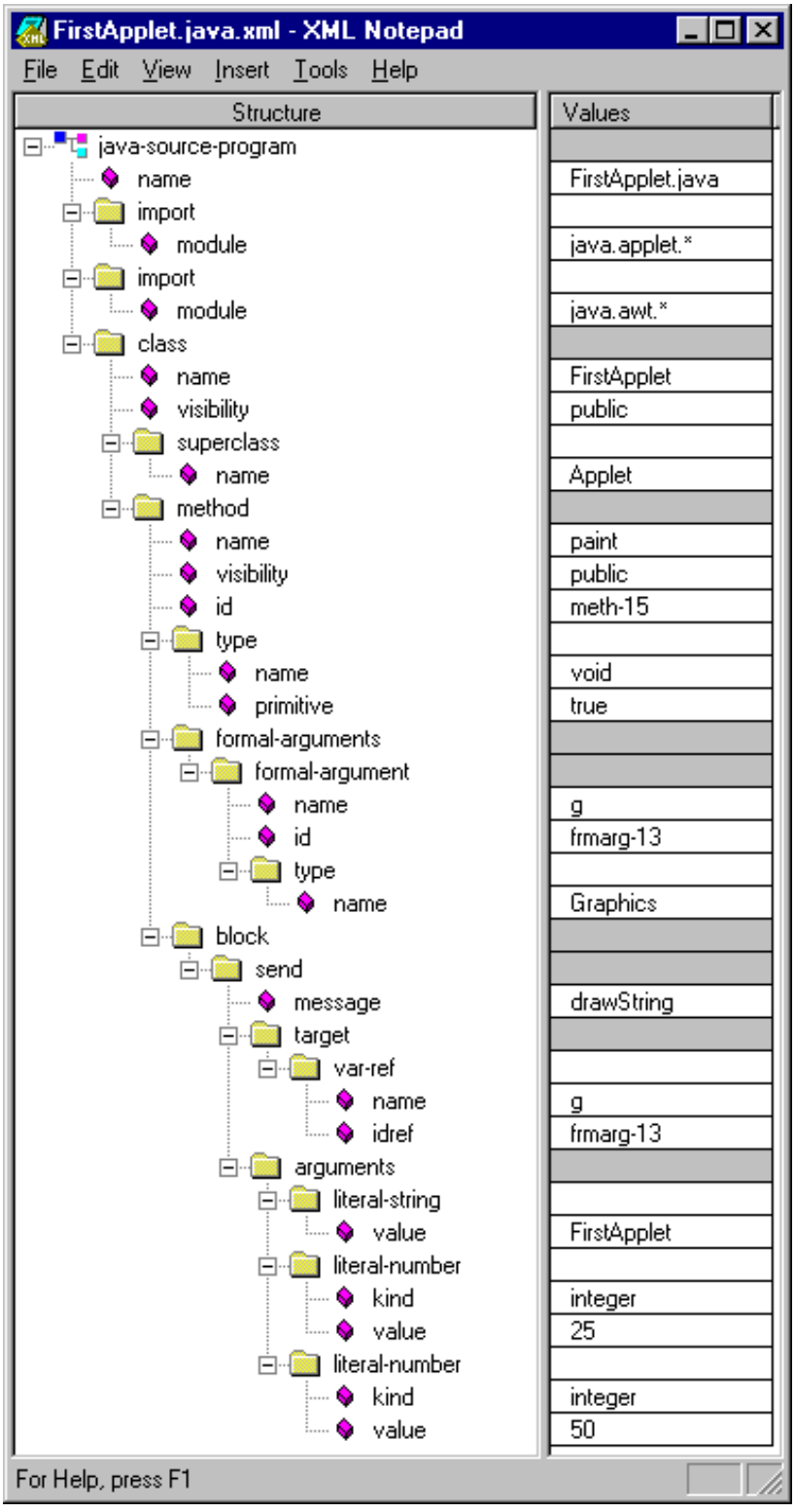
In order to understand the concept, lets look at the sample Java code.

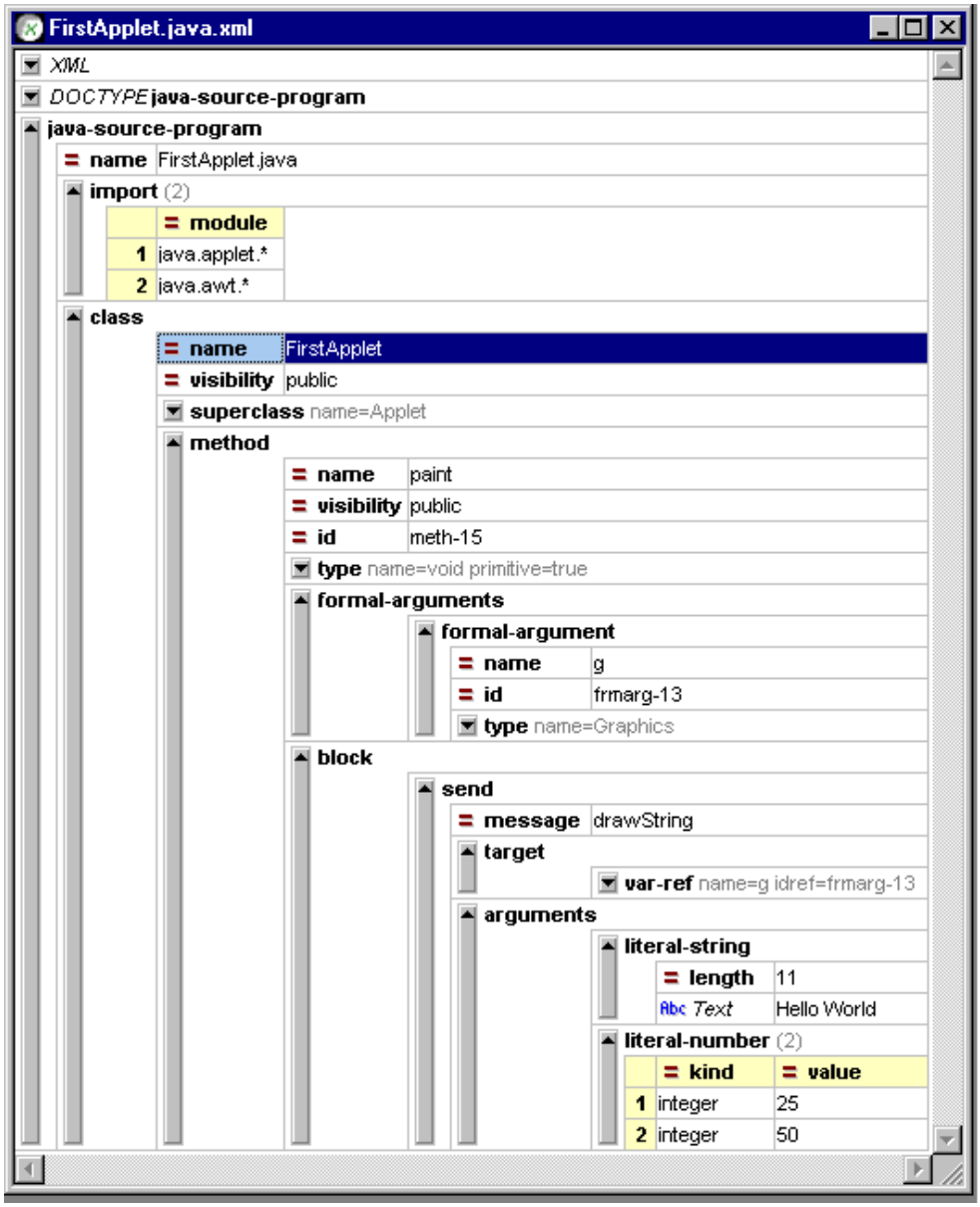
```
import java.applet.*; import java.awt.*;
public class FirstApplet extends Applet
{
     public void paint(Graphics g)
     {
          g.drawString("HelloWorld!",25,50);
     }

}
```

```
</block> </method>
</class> </java-source-program>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE java-source-program SYSTEM "java-ml.dtd">

<java-source-program name="FirstApplet.java">
  <import module="java.applet.*"/>
  <import module="java.awt.*"/>
  <class name="FirstApplet" visibility="public">
    <superclass class="Applet"/>
    <method name="paint" visibility="public" id="meth-15">
      <type name="void" primitive="true"/>
      <formal-arguments>
        <formal-argument name="g" id="frmarg-13">
          <type name="Graphics"/>
        </formal-argument>
      </formal-arguments>
      <block>
        <send message="drawString">
          <target>
            <var-ref name="g" idref="frmarg-13"/>
          </target>
          <arguments>
```

```
            <literal-string value="HelloWorld!"/>
            <literal-number kind="integer" value="25"/>
            <literal-number kind="integer" value="50"/>
          </arguments>
        </send>
      </block>
    </method>
  </class>
</java-source-program>
```

Hierarchical structure of the corresponding XML file can be seen in the figures below:


FirstApplet.java.xml - XML Notepad
File Edit View Insert Tools Help
Structure
Values
java-source-program
name
FirstApplet.java
import
module
java.applet.*
import
module
java.awt.*
class
name
FirstApplet
visibility
public
superclass
name
Applet
method
name
paint
visibility
public
id
meth-15
type
name
void
primitive
true
formal-arguments
formal-argument
name
g
id
frmarg-13
type
name
Graphics
block
send
message
drawString
target
var-ref
name
g
idref
frmarg-13
arguments
literal-string
value
FirstApplet
literal-number
kind
integer
value
25
literal-number
kind
integer
value
50
For Help, press F1



8.2. Log4J[4]

In order to decrease the size of the code in the project, we have decided to use Apache Log4J[4] for Logger component. With log4j it is possible to enable logging at runtime without modifying the application binary. The log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behavior can be controlled by editing a configuration file, without touching the application binary.

Logging equips the developer with detailed context for application failures. One of the distinctive features of log4j is the notion of inheritance in loggers. Using a logger

hierarchy it is possible to control which log statements are output at arbitrarily fine granularity but also great ease. This helps to reduce the volume of logged output and the cost of logging.

The target of the log output can be a file, an OutputStream, a java.io.Writer, a remote log4j server, a remote Unix Syslog daemon, or many other output targets.

8.3. Jikes[5]

Jikes is a compiler that translates Java source files into the byte coded instruction set and binary format. We know that java is also a Java compiler that Sun provides free with its SDK. However, Jikes has some advantages that make it a valuable contribution to the Java community. It is open source and strictly Java compatible. Its performance is high and also its dependency analysis concept provides two very useful features: incremental builds and makefile generation. In order to use JavaML, it is a must to use Jikes compiler because JavaML library is integrated to Jikes compiler and comes with it.

8.4. Apache Tomcat[6]

Apache Tomcat is an open source servlet container developed by the Apache Software Foundation. Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a pure java HTTP web server environment for Java code to run. We will use Apache Tomcat in order to test the output of our conversion operation. It is needed for testing JSF outputs to ensure their correctness.

8.5. Richfaces[7]

RichFaces is an open source Ajax enabled component library for JavaServer Faces (JSF), hosted by JBoss.org. It allows easy integration of Ajax capabilities into enterprise application development. We will use Richfaces components for mapping Applet components to JSF ones.

8. 6 Java Reflection API[8]

Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine. This is a relatively advanced feature and should be used only by developers who have a strong grasp of the fundamentals of the language. With that caveat in mind, reflection is a powerful technique and can enable applications to perform operations which would otherwise be impossible.

- **Extensibility Features:** An application may make use of external, user-defined classes by creating instances of extensibility objects using their fully-qualified names.
- **Class Browsers and Visual Development Environments:** A class browser needs to be able to enumerate the members of classes. Visual development environments can benefit from making use of type information available in reflection to aid the developer in writing correct code.
- **Debuggers and Test Tools:** Debuggers need to be able to examine private members on classes. Test harnesses can make use of reflection to systematically call a discoverable set APIs defined on a class, to insure a high level of code coverage in a test suite.

Drawbacks of Reflection

- Performance Overhead:
- Security Restrictions:
- Exposure of Internals

9. Change LOG

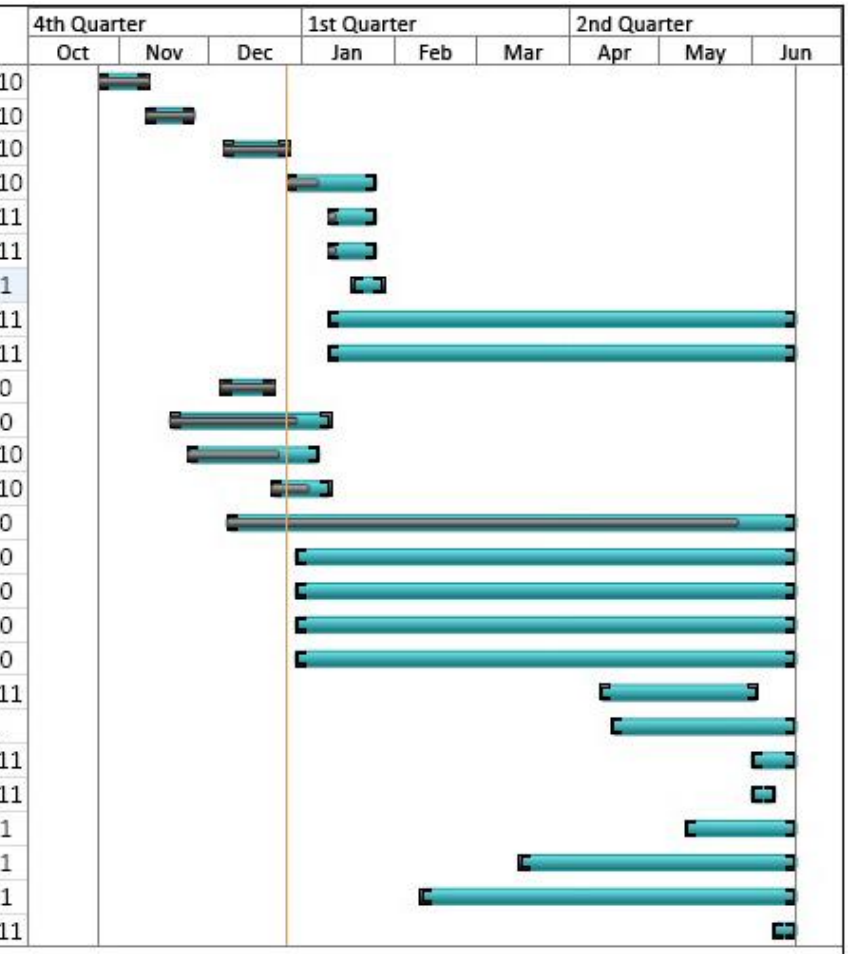
There are some changes so far with respect to Software Requirement Specifications.

SDD version 1.0 (this document)	SRS version 1.0

User can understand that conversion is completed by looking at the progress bar's %100 value	Reference sections are 2.2 "Product Functions" and 3.2.1.6 for conversionCompleted() product function
Our system will run only on Microsoft Windows platform (Vista or later)	Reference sections are 2.3 "Constraints, Assumptions and Dependencies" and 3.3.4.2.1 "Adaptability" for working platforms (OS)
Parser and Lexer component are combined into JavaML component	Reference sections are 2.1 "Product Perspective", 2.2. "Product Functions", 3.2.3. "Lexer Component Functions", 3.2.4. "Parser Component Functions" related to Lexer and Parser component

10. Time Planning (Gantt Chart)

ID	Task Mode	Task Name	Duration	Start
1		Proposal Report	13 days	Mon 25.10.10
2		Requirement Analysis Report	12 days	Wed 10.11.10
3		Initial Design Report	16 days	Mon 06.12.10
4		Detailed Design Report	22 days	Mon 27.12.10
5		Prototype	12 days	Mon 10.01.11
6		Demo	12 days	Mon 10.01.11
7		Presentation	9 days	Tue 18.01.11
8		Blog Management	113 days	Mon 10.01.11
9		Web Page	113 days	Mon 10.01.11
10		IDE Analysis	14 days	Sun 05.12.10
11		Lexer Tool Analysis	38 days	Thu 18.11.10
12		Parser Tool Analysis	32 days	Wed 24.11.10
13		Logger Tool Analysis	14 days	Wed 22.12.10
14		User Interface	137 days	Tue 07.12.10
15		Applet Class Extraction	120 days	Thu 30.12.10
16		Lexer Implementation	120 days	Thu 30.12.10
17		Translator Implementation	120 days	Thu 30.12.10
18		Parser Implementation	120 days	Thu 30.12.10
19		Integration Text	39 days	Mon 11.04.11
20		Debug	44 days	Fri 15.04.11
21		Extra Features	11 days	Wed 01.06.11
22		Finalize	6 days	Wed 01.06.11
23		Document Completion	27 days	Tue 10.05.11
24		Code Review	67 days	Tue 15.03.11
25		Test	90 days	Thu 10.02.11
26		Presentation	6 days	Wed 08.06.11



Project: Project1
Date: Mon 27.12.10

Task	External Milestone	Manual Summary Rollup	
Split	Inactive Task	Manual Summary	
Milestone	Inactive Milestone	Start-only	
Summary	Inactive Summary	Finish-only	
Project Summary	Manual Task	Deadline	
External Tasks	Duration-only	Progress	

Page 1

11. Conclusion

In this document, design considerations for project AJCON were dealt with. How our system work, how our system was decomposed, how these components work, their design architecture and connections, data design and flows were stated both by UML diagrams and by explanations. Moreover, user interactions were determined through user interfaces design. Libraries and tools which will be used during system development and operation were presented.

This document is an initial document and will be reference for Detailed Design Document. All detailed information hasn't been taken, but the general design of the system has been outlined. More details, algorithms and improvements will be given in Detailed Design Document.