NTNU
Norwegian University of
Science and Technology

Designing and Implementing Support for Web Browser-Based UIs by Using Ajax Technology

Asim Cihan Erdemli
Onur Hazar

Master in Information Systems
Submission date: June 2011
Supervisor: Hallvard Trætteberg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

One of the most important ability of Eclipse Modeling Framework (EMF) with its tools is to view the runtime state of the application based on model-driven runtime architecture. EMF supports wide range of technologies for managing, navigating, querying, storing and sharing EMF object structures.

These technologies can enable and simplify the implementation of many interesting functionalities. Regarding to its functionalities, Wazaabi represents a concrete user interface rendering engine for model-based applications and it is an EMF-based user interface model, with editor and runtime support based on the Eclipse platform. The runtime allows the developer to run user interfaces within Eclipse, however it does not have support for web browser-based user interfaces currently.

Our aim is to design and implement support for web browser-based user interfaces by using Ajax technology, while focusing on the Wazaabi framework intensively.

NTNU

Norwegian University of Science and Technology

TDT4900 Master Thesis

Designing and Implementing Support for Web Browser-Based UIs by Using Ajax Technology

Asim Cihan Erdemli
erdemli@stud.ntnu.no

Onur Hazar
hazar@stud.ntnu.no

Master of Science in Information Systems	
Submission date:	**13th June, 2011**
Supervisor:	**Hallvard Trætteberg, IDI**

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Due to the advancements in graphical user interface design and modeling technology, model-based user interfaces are becoming more dynamic and modeling frameworks allow developers to focus more on abstract modeling which means they can spend more time on user interface requirements rather than focusing model interpretation of executable user interfaces and how their codes are generated. Additionally, it can be noticed that the user interfaces of desktop based applications are fairly faster, more responsive, and more ubiquitous as they are compared with the user iterfaces of their web based counterparts, even though web based applications are evolving gradually in the last decades. With the introduction of Ajax technology, user interfaces of web based applications has become as dynamic as the ones in the desktop based counterpars. By using this advantage of Ajax technology, the main objective is to implement support for modeling web browser-based user interfaces to the existing work of "Wazaabi" project which is currently lack of handling them in an adequate way. To sum up, this master thesis describes a contribution to a new framework which is called "Wazaabi" by implementing a web browser modeling support to its already defined architecture that does not support modeling for web browser-based user interfaces as today.

Keywords: User Interfaces, Modeling, Eclipse Modeling Framework, Declarative Live User Interface Models, Ajax Technology, Ajax Frameworks.

Preface

This report was written as a master thesis by Asim Cihan Erdemli and Onur Hazar in the course TDT4900 Master Thesis at Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology in Spring 2011. The project was selected by the course supervisor and was composed of two international master students from the Department of Computer and Information Sciences, NTNU.

The project task was given by the course supervisor "Hallvard Trætteberg". This work is a contribution to Wazaabi framework, which contains the design and implementation details of a support for web browser-based user interfaces.

The group wants to thank our supervisor Mr. Hallvard Trætteberg for his important guidance and ongoing support as well as for giving useful advices and feedbacks throughout the thesis.

Norwegian University of Science and Technology, Trondheim, Norway,

June 1, 2011

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In our everyday life, computers are becoming more involved in our daily routine and all of us as computer users expect more simplicity during interaction with computers. Computer users want more user friendly interfaces when they are working. However, there are lots of information context, data flowing into the computer monitor and these screens are the only output computers have. Thus, user interfaces are playing a crucial role by becoming a bridge between the computer applications and users to inform them about what they are doing. Most of the applications have a graphical user interface (GUI) and GUI's visual ability affects the information flow in a positive way. Using this ability not only the application can receive the input from the user easily but can also provide the relevant information in the timely manner and in a format user can understand. Users interact with modern applications using graphical components such as windows, textboxes, buttons and menus. Since, it would be complex to write a GUI application by programming with code. Model-based user interface design is an exciting field because it avoids the developer from intensive coding and lets him think more about the concepts and models of components needed for user interfaces.

There are well established modeling frameworks which have already defined architecture for model-based user interface design. Furthermore, some of them has strong data models of behaviour. For example, Eclipse Modeling

Framework supports modeling classes with multiple inheritance data types and with its tools it provides runtime support for representing instance data.

1.1 Motivation

The world of web based applications development grows and grows. We can read many articles and books that explain how to develop web based applications. Nearly every week a new framework emerges in the sphere of web based applications development. However, each new technology and framework means that we have to learn it. Without learning, it is not possible to leverage the complete power of the chosen technique. There are some reasons for the emergence of such a wide range of possibilities for developing web based applications. The evolution of technology is one reason, and another is the demand for a faster and more efficient way to develop web based applications.

There are different of methods to support the web based applications that we already use today. When we consider about the traditional desktop based applications, we can easily realize the fact that the desktop based applications are typically faster and more efficient than web based applications. Even though, the web based applications have continued to become more advanced over the last years, this fact still remains that they are much less responsive than the traditional desktop based applications.

With the introduction of using model based techniques for making web based applications dynamic, the main distinction between the traditional desktop based applications and the web based applications is started to close. That is to say, the importance of building the web based applications as robust as their desktop counterparts came into question in the last years. Because, most of the users who use web based applications are only interested in gathering information as quickly as possible, as well as avoiding the waiting time which occurs in the traditional web based applications. Whereas, the desktop based counterparts are performing their tasks even faster and in a more ubiquitous way while the web servers in the traditional web based applications are still performing theirs tasks in the background.

The key motivation for this master thesis[1] is to implement support for the web browser based user interfaces by using the model based techniques of the existing work of Wazaabi project[2] which is currently lack of supporting them properly. Put in another way, our main motivation is to make sure that the existing work of Wazaabi project will be able to handle with the web browser based user interfaces faster, and more interactive and more dynamic by using Ajax technology.

1.2 Objectives

This project as a master thesis is aimed to be developed as a support for model-based user interface (UI) design in existing frameworks. Our main objective is to design and implement web browser-based user interfaces by using Ajax technology in order to facilitate and extend the usage of the existing work of Wazaabi project which has no support for web browser-based user interfaces currently.

Modeling has an ability to put two concepts (user interface design and data binding) together. According to the Ajax technology in the field of web application development as well as the improvements in GUI design and modeling technology, the model-based user interfaces are becoming better, faster, and more interactive and more dynamic. This results the development of Wazaabi framework whose aim is to support model-based user interface design for developers. Wazaabi framework supports different user interface models which are typically based on Eclipse Modeling Framework (EMF) models [3].

The tools of EMF has an ability of displaying the runtime status of the application which is based on model-driven architecture at runtime. According to the functionalities of EMF, the existing work of Wazaabi project provides a concrete user interface rendering engine for model-based applications as well as it has an EMF based user interface model, with editor and runtime support based on the Eclipse platform. The runtime allows the developer to run user interfaces within Eclipse, however it does not have support for web browser-based user interfaces currently.

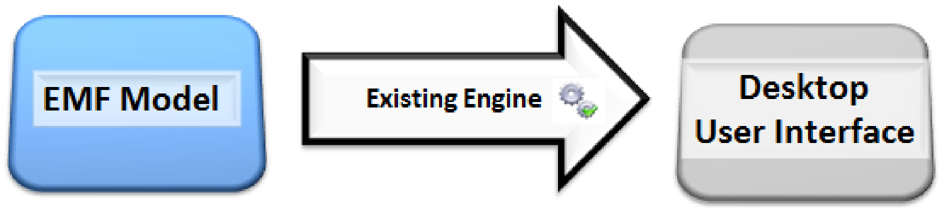


Figure 1.1: Existing Approach

This thesis is connected to earlier works about model-based user interface design techniques such as Wazaabi project developed by Olivier Moises who is the chief architect of Wazaabi. Related to this thesis, the main contribution of Moises' project was designing and implementing a declarative user interface framework based on live EMF models [4].

Since Wazaabi being an open source framework for supporting model-based user interface design, it may also be extended with integrating certain tools of other existing technologies. This is where Ajax technology comes in as a support for possible integration that comprises different tools.

Ajax is a technology which enables web based applications or rich internet applications to call the web server without leaving the current page. It is possible to do this in the background without any notice of the user asynchronously. This avoids loading the same form or page including the HTML codes multiple times, reduces the network traffic and increases the user acceptance. As a consequence, web based applications which use the Ajax technology tend to be more responsive and have more interaction with the user[5].

Wazaabi project has asserted that user interface design could be supported by live user interface models as well as modding their containing data dynamically could be possible. But, the lack of supported components for the user interface models affects developers negatively while building user interfaces. Therefore, our first objective is formed that a seamless integration between Ajax toolkits and Wazaabi framework could be possible and these tools must answer the developers' requests. With the help of the Ajax technology, second objective of this thesis is to develop a generic model for the target technology (Ajax) and to develop specific support for web browser-based user interfaces by using a new engine.

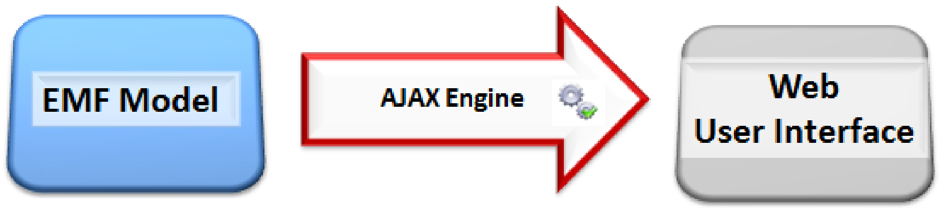


Figure 1.2: Ajax Approach

The ultimate goal of this master thesis is to implement our own Ajax engine (render) which is targeting a server-driven Ajax framework instead of using SWT (the current engine which is being used by the existing work of Wazaabi project), with an initial aim of providing support for web based user interface and model based toolkit by making use of Ajax technology and its relevant frameworks. Clearly, the figure above indicates the new approach which we plan to implement as a support for the existing work.

1.3 Structure of Thesis

This section of the introduction chapter is reserved to give an overview of the remaining chapters in the entire project report. Besides, it also provides short summaries for quick references.

- **Chapter 2 – State of the Art:** This chapter is where we introduce general information about some of the architectures and frameworks of the existing applications which are necessary for the project task. In other words, we also elaborate our knowledge about the problem situation of the project task according to existing technologies and frameworks. Therefore, the main objective of this chapter is to get an understanding of general information about Ajax technology, its main architecture and frameworks as well as the study of the existing work of Wazaabi project, its main architecture, the Wazaabi framework and including its core concepts. Finally, this chapter also covers some possible constraints of existing frameworks that are relevant to our project.

- **Chapter 3 – Problem Analysis:** The aim of the problem analysis chapter is to describe briefly the main task of this master thesis and its relevant concepts in more detail by giving a comprehensive analysis for the given problem. In addition, this chapter is where we highlight the project task by relating it with different modeling frameworks of the existing work of Wazaabi project as well as explaining the relation of the main task of this master thesis with the existing work of Wazaabi project. In other words, this chapter is reserved to study the main task of this master thesis in order to find out a possible solution which can be found as a result of some theoretical discussions where we try to fulfill the objectives of the this master thesis. Finally, this chapter also includes a section called requirements specification where we elicit the possible functional and non-functional requirements by elaborating scenarios about the desired situation at the end.

- **Chapter 4 – Proposed Solution:** This chapter is reserved to elaborate the solution that we propose for the project task which is already described in chapter 3 in more detail. Therefore, the proposed solution chapter is where we describe everything that constitutes our contribution for the project task in more detail. In addition, this chapter expresses the desired scenario in order to give as realistic as possible situation where the existing work of Wazaabi project could have been used. In order to illustrate the way of use, some high level diagrams of the important cases are also given in this chapter. Besides these, the overall system architecture of our proposed solution and where all the important and relevant components of the existing work of Wazaabi project are situated in our proposed solution are also presented in this chapter. Finally, there is also a part where we elaborate the overall goals for the actual implementation of our proposed solution before we closing this chapter.

- **Chapter 5 – Implementation:** The implementation chapter is dedicated to take into account the elaboration of how we actually implement the proposed solution which is already given in chapter 4 in more detail. In addition, this chapter also emphasizes the inner functionalities of the most important and relevant components of the ZK Ajax framework in more detail by giving various diagrams. Furthermore, the implementation chapter is where we also give an overview of the programming standards that we use, including the source code layouts and the usage of existing libraries from the existing work of Wazaabi project. In other words, this section of the report describes how the implementation phase is carried out, how well the work is integrated and above all, first and foremost how well our proposed solution is reflected in the actual Wazaabi project. Finally, this chapter also includes the discussion of different software applications, programming technologies and computer programs that are being used during the implementation of our proposed solution for the main project task.

- **Chapter 6 – Evaluation:** This chapter contains information about the evaluation of the project by giving the description of how we evaluate the implementation of the proposed solution. Additionally, the aim of this chapter is to verify how well the main objectives of the project task are solved as well as to what extent the requirements are met throughout the development stage of the entire project. The evaluation chapter also gives two sample scenarios as well as explaining what the different aspects of the project are by showing them how they are accomplished according to the fulfillment criterias of the requirements specification.

- **Chapter 7 – Conclusion:** This chapter is reserved to conclude the project report by summarizing the contribution we made for the project task as well as it contains the reflections about all aspects of the project (both the positive and negative experiences) by explaining the main project outcome. In addition to them, there is a section called future work which contains suggestions of the ideas for possible improvements by giving examples of how we would accomplish the project task in different ways in a possible future project. To sum up, the conclusion and future work chapter marks an end to this report.

Chapter 2

State of the Art

This chapter of the report introduces a general information about some of the existing application frameworks. The purpose of this section is to get an understanding of the general Ajax architecture and possible constraints of existing frameworks. First, we are going to explain Ajax technology by describing how its architecture is developed. Then, we will give a brief information about an open source framework called Wazaabi and its core concepts that are relevant to our project.

2.1 Ajax Technology

With the new emerging technologies in the field of web application development part of software engineering which is progressing day by day, there is a new coding technique which is becoming quite popular in this area, it is called Ajax technology. The word Ajax stands for Asynchronous JavaScript and XML which has been around since 1998. This new coding technique has a main objective to make traditional client-server communication more efficient. In other words, Ajax is a new coding technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS and JavaScript languages. Ajax technology uses XHTML for content and CSS for presentation, and JavaScript for dynamic content display.

The motivation of why Ajax was created was the idea of making the client part communicate with the server part, without any need of a different page or refreshing. Therefore, the Ajax technique is mainly used on the client part in order to create interactive web applications. By using the Ajax technology, the client part of a web application can retrieve data asynchronously from the server part which works in the background without making any change of display and behavior of the existing pages [6].

In the traditional approach of web applications, the information is transmitted between client and server by using synchronous requests. As an illustration to this, when a user fills out a form and then clicks on submit button, the user is directed to a new screen with new information that came from the server part. Whereas, with the usage of Ajax technique, when the user clicks on submit button, the JavaScript makes a request to the server part, interprets the outcomes and updates the current screen without even directing the user to a new screen. Simply, the user does not know that anything was even transmitted to the server part. Therefore, we can say that, Ajax is a web application technology which is independent of web server software. That is to say, the user can still continue using the web application while the client part requests the information from the server part working in the background. Moreover, in some web applications which use Ajax technique, only a mouse movement is sufficient to trigger an event without any necessity to click a button. Therefore, Ajax also provides not only an interactive web application but also an intuitive and natural user interaction in it.

To make the long story short, the client parts of the web applications which use Ajax technique can interact with the server part without making a typical refreshment of the current screen or redirecting a new screen which is done in the traditional web applications[24]. Because, Ajax technique allows the content on web pages to update themselves immediately when the user performs an action, unlike an HTTP request, during which users must wait for a whole new page to load. Additionally, Ajax can also gather information from other sources. The web applications can have Ajax code in order to validate if the correct information has been entered into a given text field by the user. Furthermore, the web applications that use Ajax can dynamically warn the user if the username that the user has entered is already in use, the e-mail that the user has entered is not valid, the passwords that the user has entered do not match with each other, or the web page itself can give some hints about the security level of passwords that the user has entered[7].

We would like to give some well known examples about the Ajax technique which can help any type of web pages and web applications. Google has implemented the Ajax technique to their maps, e-mail and other tools that they offer. Additionally, some e-commerce web sites are frequently using the Ajax technique for credit card submissions. On the client side of your page is your visitor that has more interaction and quicker results.

- **Google Search Engine:** Google Web Search which is owned by Google, is the most famous and the most used web search engine that uses the Ajax technique on the internet world. It currently receives several hundred million queries everyday through its various services. The following figure shows the real time search suggestions of Google, which provides the Ajax technique. The Google search engine can be accessed via [11].

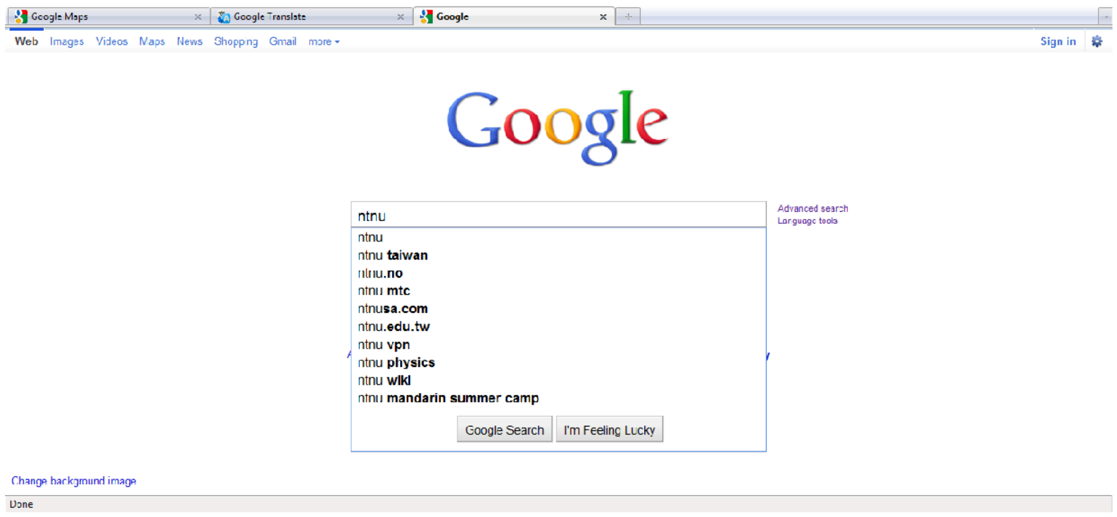


Figure 2.1: Google Search Engine

- **Google Maps:** Google Maps is one of the well known web applications which uses Ajax technique offered by Google. The user can drag the entire map by using the mouse rather than clicking on a button. So, the interface of the application allows the user to change views and manipulate the map in real time. The application can be accessed via [12].

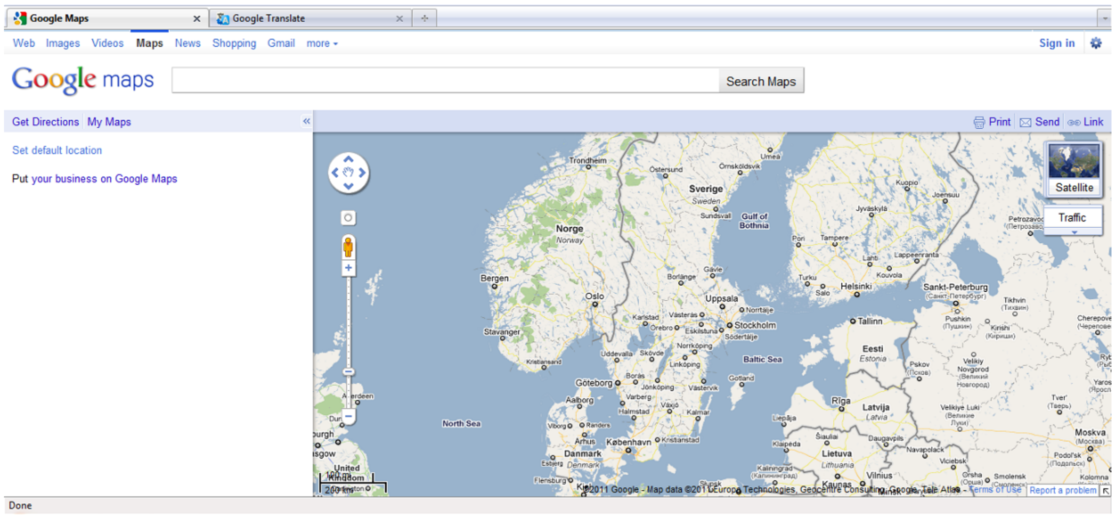
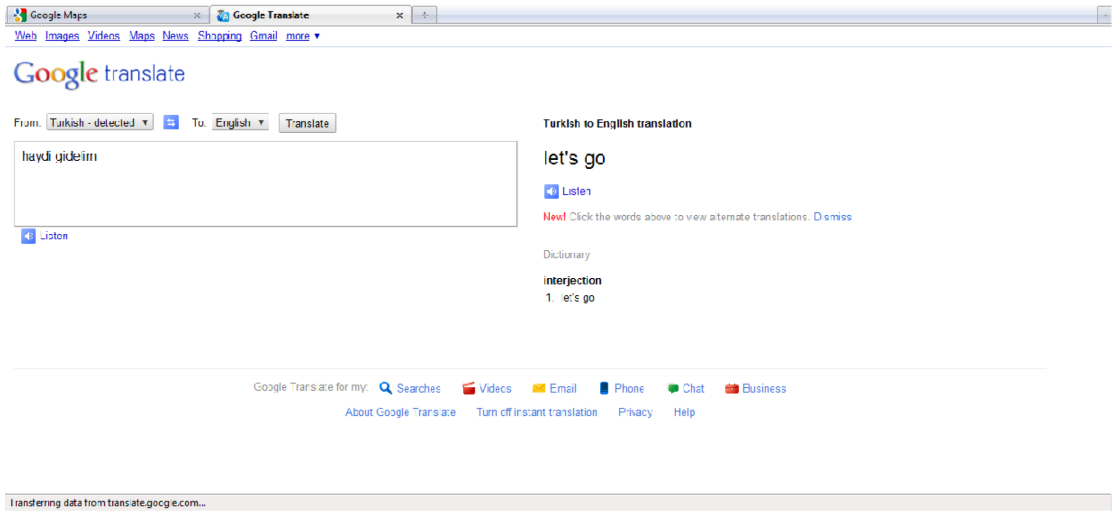


Figure 2.2: Google Maps

- **Google Translate:** Google Translate can be an another example for the well known Ajax based web applications. It is a free statistical machine translation service which uses Ajax technique offered by Google. It has a main functionality of translating a section of text, document or website into another language. The application can be accessed via [13].


Google Maps
Google Translate
Web Images Videos Maps News Shopping Gmail more ▾
Google translate
Translate
Listen
Listen
Google Translate for my: Searches Videos Email Phone Chat Business
About Google Translate Turn off instant translation Privacy Help

From: Turkish - detected To: English

haydi gidelim

Turkish to English translation

let's go

New! Click the words above to view alternate translations. Dismiss

Dictionary

interjection

1. let's go

Transferring data from translate.google.com...

Figure 2.3: Google Translate

The web applications which are created by using Ajax technique use an engine that has a role like a mediator between the client part (in general, the user's web browser) and the server part from which it is requesting the relevant information. Rather than loading a traditional web page, the client part loads the Ajax engine which displays the web page that the user sees. The engine keeps running in the background by using JavaScript to communicate with the client part. When the user performs an action by clicking on the web page, a JavaScript call is sent to the Ajax engine which is responsible for responding back to the client part instantly. If the Ajax engine requires an additional data to perform its action, then it requests the relevant data from the server part by using XML, while it is simultaneously updating the website or the current screen of web application. The following figure illustrates the standart Ajax interaction between the client part (the web browser) and the server part in a general manner.

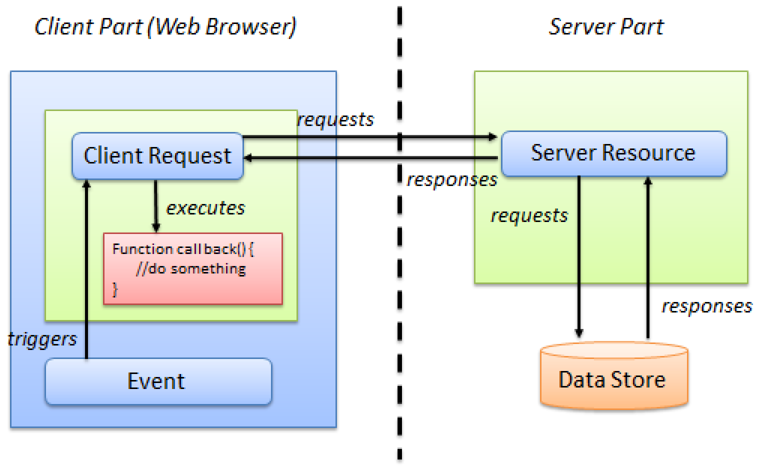


Figure 2.4: Standart Ajax Interaction

Before we dive into the archtitecture of Ajax and the features of its framework, we would like to sum up everything again. As we mentioned before, the rising Ajax technique is becoming one of the best internet technology for creating dynamic web pages[8]. It is done by embedding JavaScript code into the HTML based web pages in order to send requests to the server part. In the server part, some processing is required to handle the requests which are coming from the client part. To handle the requests of finding out the information or just storing the data, we need for a specialized Ajax framework in the server part. The Ajax frameworks have always a JavaScript part in themselves, and they sometimes also have a server part in another scripting language like XML. A lot of them exist in various programming languages working on all programming environments around but we would like to keep in mind only the most widely used ones in our report. Therefore, in the following section, we will continue our discussion by elaborating the architecture of Ajax itself as well as covering the Ajax frameworks and their significant aspects.

2.2 Ajax Architecture & Framework

In this section, we will discuss what the architecture of Ajax is, the main components of it and what makes Ajax based web applications different than traditional web applications firstly. Then, we will continue our discussion with Ajax frameworks that are used for developing Ajax based web applications as well as giving brief explanations of what the different types of Ajax frameworks and their main features are. Finally, we will mention about the various existing Ajax frameworks as a quick review for being a part of the background knowledge in this master thesis.

2.2.1 Ajax Architecture

Ajax has a browser-side technology stack and architectural style which interacts with the server part to update the user interface without refreshing the entire web page. Even though, it is more complicated to code, it can offer a faster and smoother user experience. When we consider ajax technology, we should also mention the differences between a traditional web application and an Ajax application. The following figure shows that differences as well as the architectural implications of Ajax technique.

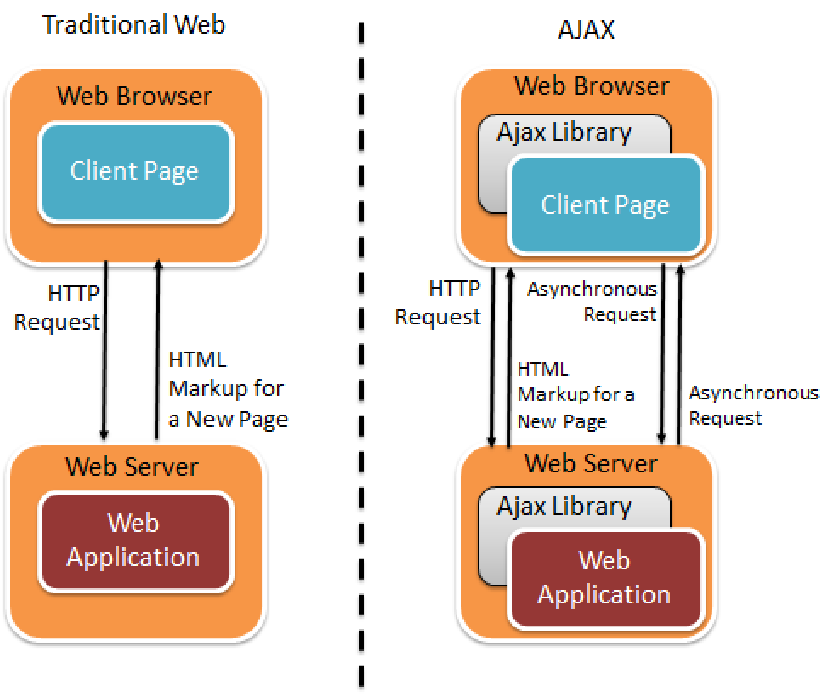


Figure 2.5: Traditional Web Application vs Ajax Web Application

The traditional web application approach simply does everything on the server part. Occasionally, the application emits script code to run a task on the client part for only special circumstances, for example to handle data validation. Whereas, an Ajax application approach uses a client part framework which takes care of issuing calls to the web server. An Ajax server part framework then takes care of the request and returns data feed to the client part. This will often be a JavaScript Object Notation (JSON) data stream, but other formats such as XML, RSS, and CSV can be also used. The client part receives data feeds and updates the user interface by using JavaScript. The server part responds to requests by returning raw data which are encoded in a given format. With the usage of Ajax technique, the bandwidth consumption is minimized, the speed of the web application increases since the requests take less time to complete, and user interface updates are able to take effect with no visible postback. Even though, the usage of Ajax technique solves many problems, the increasement in the actions on the client part also brings about new critical issues, such as new coding practices, new security hazards, accessibility concerns, and so on[9].

2.2.2 Ajax Framework

The Ajax framework is a web application framework which assists to develop web applications that use Ajax technique. As we already metioned in the previous section, the Ajax technique is a collection of different technologies which are used to build dynamic web pages on the client part. The data is read from the server part or sent to the server by JavaScript requests dynamically. Nevertheless, some additional processing at the server part may be required to handle some kind of JavaScript requests, such as finding and storing the data. This is accomplished more easily when a framework is dedicated to process those kind of JavaScript requests. Therefore, the main objective of the Ajax framework is to provide the Ajax engine and the associated server part and the client part functions. In other words, the Ajax framework is an application framework that takes the difficulties out of building web applications that use Ajax technique, both at the client part and server parts. The following figure shows how the architecture of an Ajax framework is structured in a general manner.

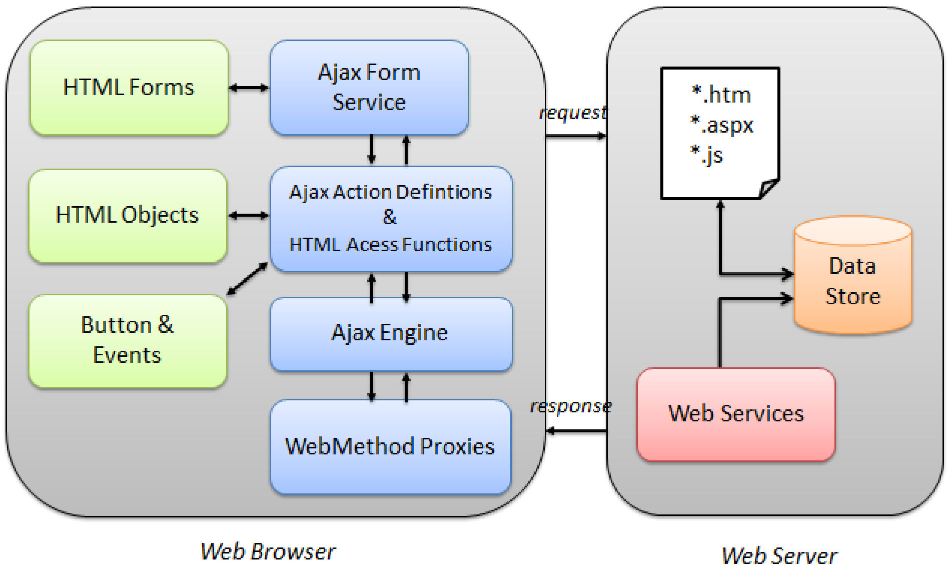


Figure 2.6: General Architecture of Ajax Framework

Why Do We Need an Ajax Framework?

An Ajax framework is essentially an Ajax engine which is intended to suppress the time elapsed and the processes done in the waiting time for the user while client part is accessing the server part[10]. Even though, the Ajax frameworks generally provide classical, cross-browser functions to use the XMLHttpRequest objects, they can go beyond that, and allow us to build rich web based applications, the applications with a graphical user interface and the other features of desktop software which run through a web browser while exchanging data with a remote server part.

Features of an Ajax Framework

Every Ajax frameworks are able to communicate with the server part, and therefore, they can read data or send their data or requests. In the last case, a script code which is running on the server part is required. The frameworks often add components that make use of the asynchronous communication with the server part. The traditional examples can be buttons, tabbed panels, grids, listboxes and other such widgets[21]. A more innovative example, the boxes are more and more often implemented, and lightbox and slimbox are two of them. There are image galleries that place them side by side on the screen and that are making use of the Ajax technique to display them instantaneously. Frameworks can also be server-driven, and in this case, components are created on the server part with a scripting language such as PHP, and sent to the web browser (the client part). The Ajax is used to transmit user actions to the server part, and to handle the results[22].

2.2.3 Types of Ajax frameworks

There are four different types of Ajax frameworks which differ in the amount of coding expertise required, as some necessity of extensive knowledge, while others frameworks use pre-built components that require less experience. The Ajax frameworks can be grouped into four different categories according to the features they offer and the skills required of the user. Therefore, in this section, we will discuss the four different types of Ajax frameworks which are available for developing Ajax based web applications. According to the different features of the Ajax framework, there are four different types which are[20]:

1. Direct Ajax frameworks
2. Indirect Ajax frameworks
3. Ajax component frameworks
4. Server-driven Ajax frameworks

Now, we will be briefly elaborating about these four different type of Ajax frameworks one by one in the rest of this section. Later on, we will explain why we select the server-driven type of Ajax frameworks out of these four different types of Ajax frameworks for making a contribution to the aim of our master thesis.

Direct Ajax Frameworks

This type of Ajax frameworks require some skills and knowledge of HTML, CSS, JavaScript as well as expertise in Ajax technique. In this type of Ajax framework, the developer writes web pages in the HTML and the framework Application Programming Interfaces (APIs) directly interacts with HTML elements. This type of Ajax frameworks provides the APIs which have a variety of different purposes, generally including communications, DOM manipulation, event handling, and animation support in HTML elements. Additionally, this type of Ajax frameworks are commonly smaller so that they are mostly used for shopping websites. Nevertheless, this type of Ajax frameworks are not suitable enough to be used for the complex web applications such as Ajax web based email client as long as there does not exist further Ajax frameworks that are layered on top.

Indirect Ajax Frameworks

This type of Ajax frameworks rely on compiler technology that actually converts the high level language into Ajax, HTML and JavaScript rather than writing directly in Ajax, HTML and JavaScript. In other words, this type of Ajax frameworks includes a high level language is used together with a corresponding compiler which generates the JavaScript from the high level language. Therefore, in this type of Ajax frameworks, some skill and knowledge of the high level language such as CSS and HTML are necessary for the developer. However there is no requirement of knowing a great deal of Ajax and expertise in JavaScript, since the framework itself generates the Ajax and JavaScript code. This type of frameworks are mostly used with the appropriate libraries, modules and classes which are written in the high level language in order to handle communications, DOM manipulation, HTML element manipulation, and event handling.

Ajax Component Frameworks

This type of Ajax frameworks provide lots of pre-made components, such as grids, tabbed panels, trees, comboboxes, date pickers, HTML editors which automatically generate and manage their own HTML codes. All of these pre-made components can be used while the user interfaces of the web applications are being developed via only adding a few lines of JavaScript, XML tags or via adding some special attributes to regular HTML elements. This type of Ajax frameworks are usually larger, and used for web based applications instead of websites. Furthermore, some of the Ajax component frameworks require expretise in Ajax, HTML, CSS, and cross-browser testing. As an illustration to this, grids, tabs, and buttons can be provided by the framework itself but the user input forms are expected to be created directly in HTML or CSS and manipulated by Ajax technique. Whereas, the other Ajax component frameworks can provide a complete component palette so that it makes only some general skills in XML or JavaScript required. As an advantage, the Ajax component framework allows the developers to develop web based applications rapidly. This kind of Ajax frameworks support tailorable APIs and event handling functions, new facilities for making web pages eye-catching, programmatic control of the pre-made components, the extensibility facility which makes the Ajax framework practical as well as providing the possibility of creating new components by extending the existing ones.

Server-Driven Ajax Frameworks

This type of Ajax frameworks rely on the server part components for the Ajax functionality as well as providing more efficiency and performance for server part. In this type of framework aims developing and managing the server part components where the server part itself generates necessary HTML and JavaScript for Ajax support. Therefore, the server-driven Ajax framework renders the components and handles the client-server requests & responses by data handling. The Ajax frameworks which handle presentations completely within the web browser can provide greater ability to response when they are able to manage more user interactions without server intervention. In the server-driven Ajax frameworks, some user interface interactions may react slowly such as when an input field is automatically enabled according to the server part requests. This approach is still quite popular for the cases where the usability of a complete Ajax architecture can not be covered or where the server part interaction is required anyway. Extending the framework requires a well understanding of which parts of the interactions are managed on the client part and which parts of the interactions are managed on the server part, plus some coding skills in JavaScript and Ajax.

2.2.4 Comparison of Existing Ajax Frameworks

In this section, we will continue our discussion with the comparsion of the most widely used existing Ajax frameworks that are used for developing Ajax based web applications as well as giving brief explanations and the main features of them one by one. Additionally, we will provide a table as a quick reivew of what kind of Ajax frameworks they are, for being a part of the background knowledge.

Google Web Toolkit (GWT)

Google Web Toolkit (GWT) is an open source set of tools that allows web developers to create and maintain complex JavaScript front end applications in Java. Other than a few native libraries, everything is Java source that can be built on any supported platform with the included GWT Ant build files. It is licensed under the Apache License version 2.0. GWT emphasizes reusable, efficient solutions to recurring Ajax challenges, namely asynchronous remote procedure calls, history management, bookmarking, internationalization and cross-browser portability[14].

Rich Ajax Platform (RAP)

Rich Ajax Platform (RAP) is an open source software project under the Eclipse technology project which aims to enable software developers to build Ajax enabled rich internet applications by using the Eclipse development model, plug-ins and a Java only APIs. It can be considered as a counterpart for web development to the Rich Client Platform (RCP). The API is very similar to RCP, so developers who are familiar with RCP can reuse their existing knowledge. RAP encourages the sharing of source code between RCP and RAP applications in order to reduce the development effort for business applications that require both a desktop based and a web based front end[15].

Dojo Toolkit

Dojo Toolkit is an open source modular JavaScript library (or more specifically JavaScript toolkit) designed to ease the rapid development of cross platform, JavaScript, Ajax applications and web pages. It uses packages with a mechanism to load them together along with the page. It can build reusable components and widgets, a lot of them is provided on the site. It allows to manipulate the DOM more easily, and to make graphical effects[16].

Vaadin

Vaadin is an open source web application framework for rich internet applications. In contrast to JavaScript libraries and browser plug-in based solutions, it features a server part architecture which means that the majority of the logic runs on the server part. Ajax technology is used at the browser part to ensure a rich and interactive user experience. On the client part, Vaadin is built on top of and can be extended with Google Web Toolkit[17].

ZK

ZK is an open source Ajax web application framework, written in Java that enables creation of rich graphical user interfaces for web applications without JavaScript and programming knowledge. The core of ZK consists of an Ajax based event-driven mechanism, XML User Interface Language (XUL) and eXtensible HyperText Markup Language (XHTML) based components, and a markup language for designing user interfaces. Programmers design their application pages in feature rich XUL & XHTML components, and manipulate them upon events triggered by end user's activity. It is similar to the programming model found in desktop GUI based applications. ZK takes the server-driven approach that the content synchronization of components and the event pipelining between client part and server part are automatically done by the engine and Ajax plumbing codes are completely transparent to web application developers. Therefore, the end users get the similar engaged interactivity and responsiveness as a desktop application, while programmers' development retains a similar simplicity to that of desktop applications[18].

Framework	**Direct**	**Indirect**	**Component**	**Server-Driven**
GWT	✓	✗	✓	✗
RAP	✓	✗	✓	✓
Dojo	✓	✗	✓	✓
Vaadin	✗	✓	✓	✓
ZK	✗	✓	✓	✓

Table 2.1: Comparison of Existing Ajax Frameworks[20]

So far we have mentioned about the different types of Ajax frameworks and the existing Ajax frameworks which are being widely used today. In order to relate the different types of Ajax frameworks with the existing Ajax frameworks that we examined as well as making a comparison among them in a better way, the table above was given.

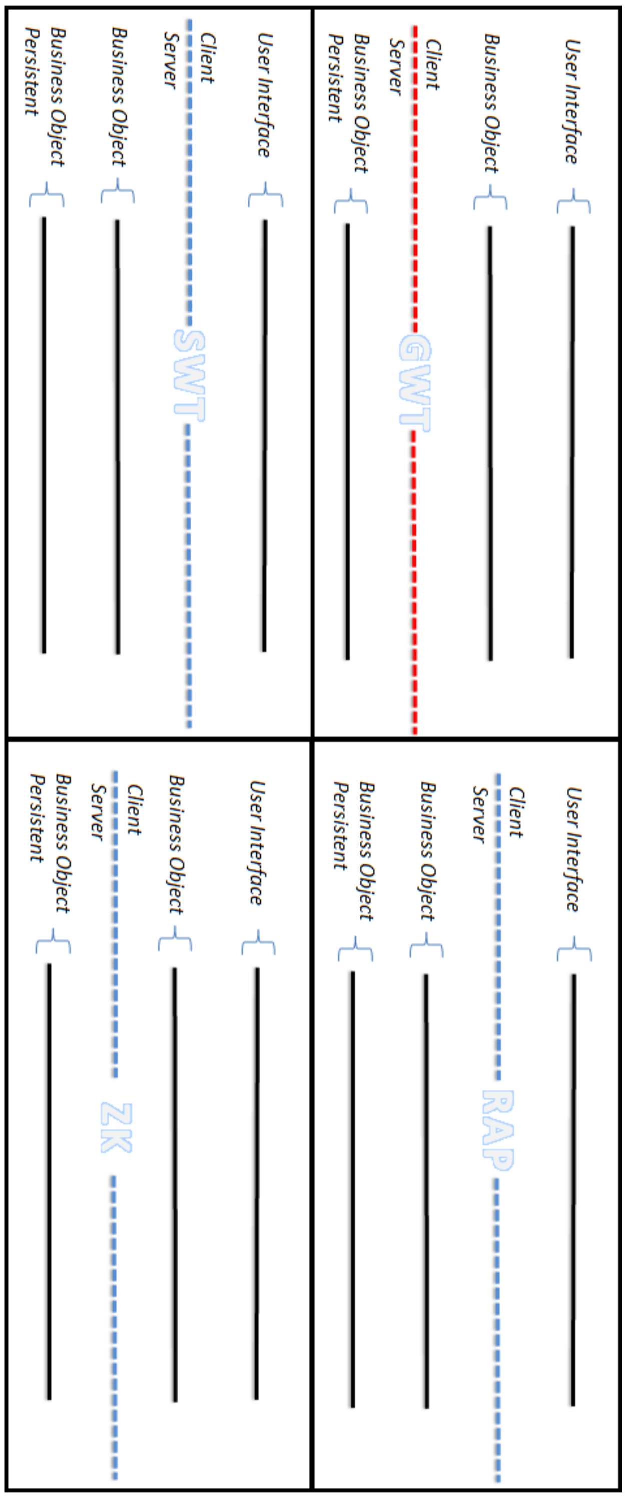


Figure 2.7: Diversity of Existing Ajax Frameworks

In the figure above, the existing Ajax frameworks are compared with regard to their relationship with user interface, business object and business object persistent by taking into consideration of client-server architecture.

2.2.5 Ajax Java Frameworks

Before we start our discussion about the existing work of Wazaabi project and its features, we would like to mention about Ajax Java frameworks in general and the relation with Ajax and XML User Interface Language (XUL) briefly.

Java Based Frameworks

Java is the programming language that is the more often used to build web services. Therefore, a Java framework permits to use Java web services interactively within a web page. They are more commonly used as in Direct Web Remoting (DWR)[19] which allows coding in a web browser to use Java functions running on a web server as if those functions were within the browser. Therefore, the JavaScript part is used to update the web page and to gather data with servlets which are Java applications that are different from applets, so that they run on the server part and generate HTML pages that are sent to the client part servlets which can run on web browsers that are not Java enabled. The technique in generating real time Java codes from JavaScript codes sends it to the server part and runs it.

Google Web Toolkit includes a toolbox to develop Java applications, which are then compiled into JavaScript code, and this code processes the HTML page through Document Object Model (DOM) methods. Therefore, the elements in the HTML page can be may be addressed and manipulated within the syntax of the programming language in use[23].

Legacy Java software for the web based applications now is moving to Ajax technique and already started to include features of Ajax so that all the logic behind is being executed by the server part, and the selected Ajax framework encapsulates the web rendering (the main function of the Ajax engine) as well as communication layers. In this manner, the Ajax can be thought as a unique approach in which the web based application can be thought that, it was written as if it was a desktop application which is typically more robust and efficient than its web based counterpart.

Relation with Ajax and XUL

The Ajax technique makes use of the JavaScript programming language when the XML User Interface Language (XUL) embeds also JavaScript into XML in order to define the interaction with the user interface. Since XUL is an XML user interface markup language developed by the Mozilla, the two distinct systems work on the Firefox web browser. Briefly, the Ajax technique allows communicating with the server part from a remote web page when the XUL displays a user interface either in a local computer or through the internet, within the same web browser (or in the client part generally).

The main difference is in the use of HTML tags to extend Ajax and to extend XUL. Therefore, ZK Ajax Framework is designed for allowing Ajax and XUL to communicate each other properly. XUL relies on multiple existing web standards and web technologies, including CSS, JavaScript, and DOM. Such reliance makes XUL relatively easy to learn for the developers with a background in web based application programming and design.

Mozilla provides experimental XULRunner which is a runtime environment to provide a common back-end for XUL applications as well as letting the developers to build their web based applications on top of the Mozilla application framework and of XUL in particular. Additionally, XUL provides a portable definition for common widgets, allowing them to move easily to any platform on which Mozilla applications are able to run. While the XUL serves primarily for constructing Mozilla applications and their extensions, it may also feature in web applications which are transferred over HTTP [36].

2.3 Wazaabi

This section presents Wazaabi's framework and how it is closely connected to EMF which can be called as an essential framework of the live modeling using EMF tools and meta-models. We will indicate the benefits of Wazaabi while how editing user interfaces is being done in Wazaabi. Wazaabi makes user interface design doable by closing the gap between the user interface design and running time of an application user interface.

2.3.1 What is Wazaabi?

Wazaabi is a set of Eclipse plugins allowing the use of EMF based models for building parts of an application GUI[2]. It is designed as an open source framework for building declarative user interface models dynamically. For example, user interface models in SWT, SWING, and JSF are supported by Wazaabi which can also be applied to other user interface technologies.

Regarding to its functionalities, we can mention about two issues which are different from each other. First, it is possible to build declarative user interfaces which implies that the designer is allowed to design the user interface declaratively. In Wazaabi, we can declare an instance of user interface model which is a representation of the real user interface. In other words, the user interface is a running instance of the model and it is a live model which means it is not necessary to generate code from the model itself. In addition, there is an engine responsible for rendering the model of target user interface technology like SWT, SWING. The following figure illustrates the architecture of Wazaabi as a stack structure;

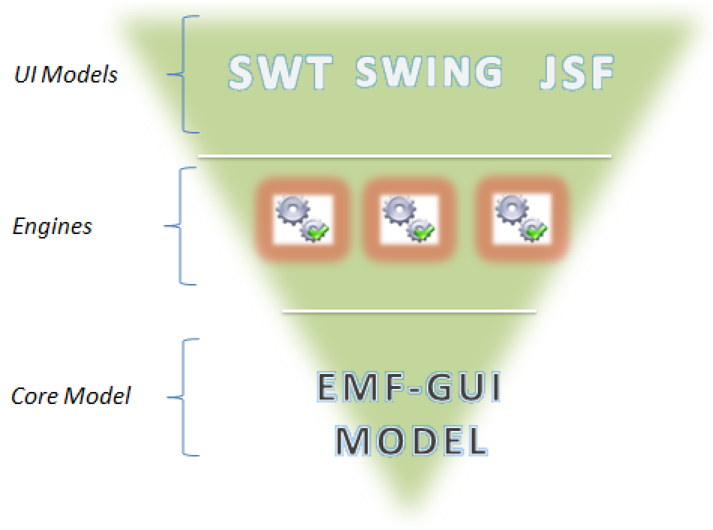


Figure 2.8: Wazaabi Stack Model

2.3.2 Wazaabi Framework

There are four different sections of Wazaabi Framework: the user interface models, the Editor, Architect and Engines. The user interface models represent the target user interface and they contain all the data which needs for building components of the target user interface[26].

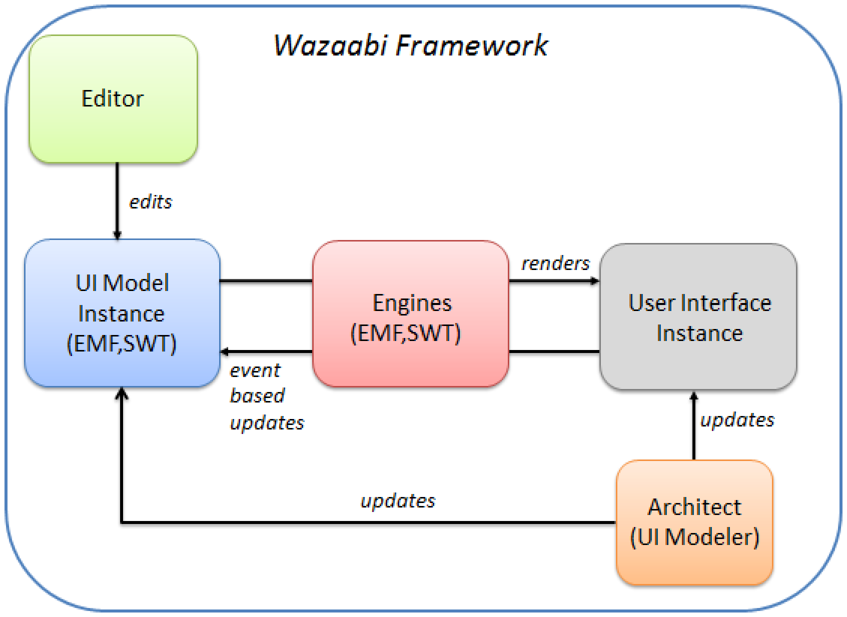


Figure 2.9: Wazaabi Framework

Meanwhile, the Editor serves as a generic EMF generated editor which is designed to support different kinds of user interface model. The other part is called as the Architect which is the user interface modeler working with the principle "WYSIWYG". User interface models are loaded by the Architect with the corresponding render for simulating the user interface rendering during the edition. As a last component, the Engines render user interface models for specific target user interface technology. See Figure 2.11. For example, an engine for SWT, SWING, Nebula can be provided. JSF and GWT will also be supported soon.

In Wazaabi, the model of a GUI can be called as an instance of an underlying model. Wazaabi architecture supports three GUI underlying models; SWT, SWING and JSF model. By defining an instance of those models, it is possible to create a model of a GUI which defines the GUI itself. Furthermore, there is a basic model called core model provided by Wazaabi. The other models inherit from this core model. The main goal is to separate the user interface and their behaviours of the target technology and deal with them independently.

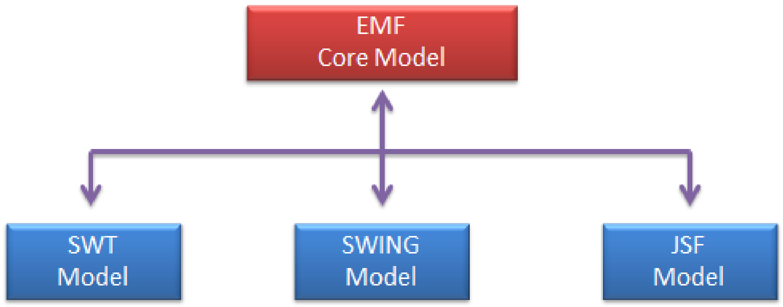


Figure 2.10: Platform Independent Models Hierarchy

Actually, the Wazaabi core model can be called as root model of GUI because it aims to identify different objects which compose a default GUI consisting of graphical elements like widgets. Most of them are abstract because of their platform dependent definition. At the same time, other elements which are platform independent like column, stylesheet are concrete and can become instances of the model for other derived platforms.

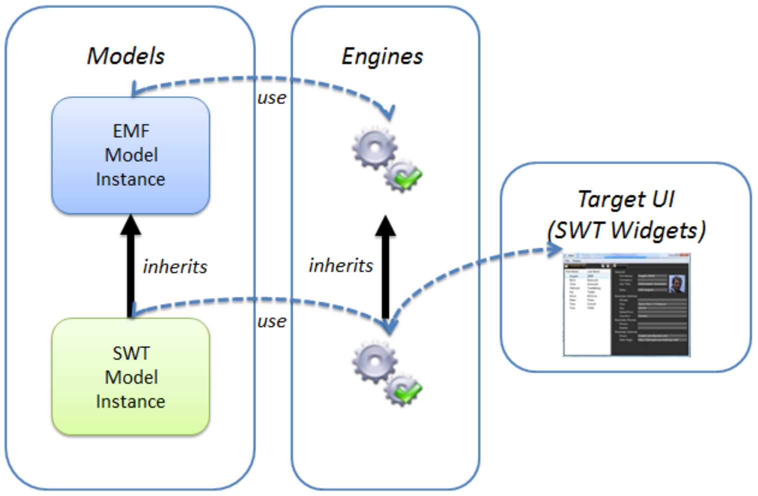
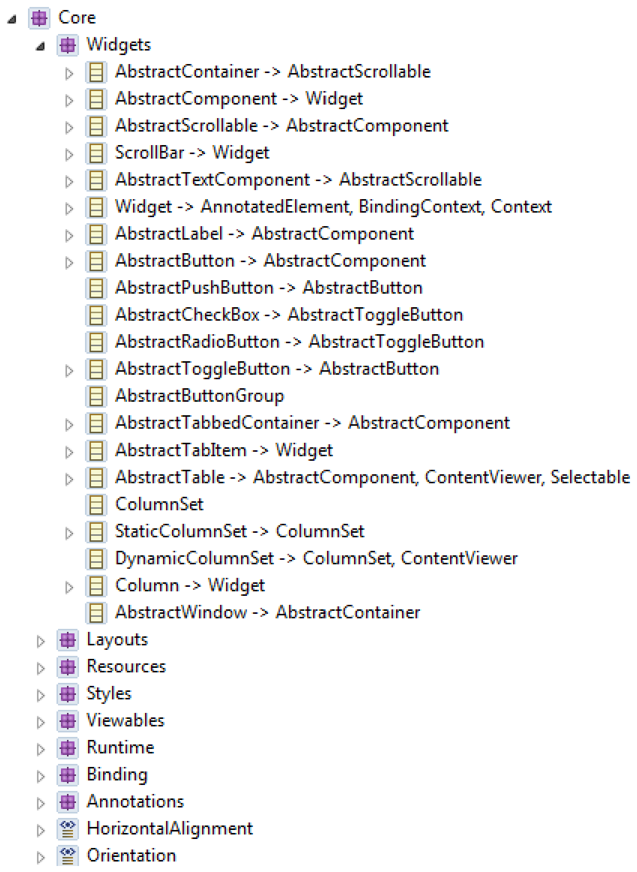


Figure 2.11: Relationship between Models & Engines[2]

In Wazaabi, everything belongs to a model: widgets, databinding and application workflow etc. There are five divided packages in the Wazaabi core model; Widgets, Layouts, Viewables, Styles, Actions, Resources. Shortly, the Widgets package keeps the abstract definition of the graphical elements of user interface like widget, layout, style, action. On the other hand, Layout package is responsible for locating the widgets.



```
Core
  Widgets
    AbstractContainer -> AbstractScrollable
    AbstractComponent -> Widget
    AbstractScrollable -> AbstractComponent
    ScrollBar -> Widget
    AbstractTextComponent -> AbstractScrollable
    Widget -> AnnotatedElement, BindingContext, Context
    AbstractLabel -> AbstractComponent
    AbstractButton -> AbstractComponent
    AbstractPushButton -> AbstractButton
    AbstractCheckBox -> AbstractToggleButton
    AbstractRadioButton -> AbstractToggleButton
    AbstractToggleButton -> AbstractButton
    AbstractButtonGroup
    AbstractTabbedContainer -> AbstractComponent
    AbstractTabItem -> Widget
    AbstractTable -> AbstractComponent, ContentViewer, Selectable
    ColumnSet
    StaticColumnSet -> ColumnSet
    DynamicColumnSet -> ColumnSet, ContentViewer
    Column -> Widget
    AbstractWindow -> AbstractContainer
  Layouts
  Resources
  Styles
  Viewables
  Runtime
  Binding
  Annotations
  HorizontalAlignment
  Orientation
```

Figure 2.12: Wazaabi Core Model

The Viewable package is responsible for Table elements. It deals with filling and rendering of table elements. Table which applies the Viewable interface, gets an object as an input from the underlying model. The styles package shows the CSS behaviour on EMF. The Actions package is responsible for related actions on button or actions performed by other elements. The Resources package handles fonts, colours and images.

2.3.3 Editing UI's in Wazaabi

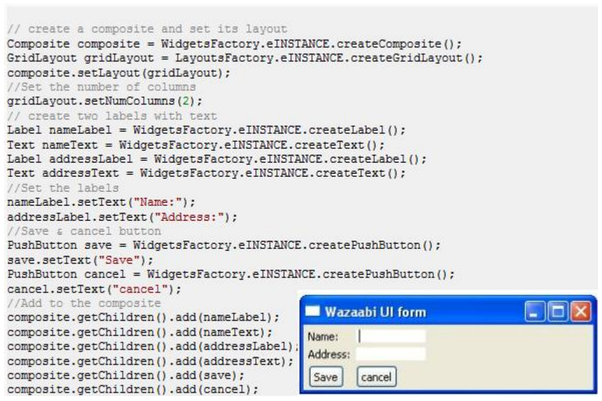
Practically, there are three ways to define GUIs in Wazaabi;

1. Graphically with the Architect (still developing).

2. Programming with code.

3. Building with declarative Editor.

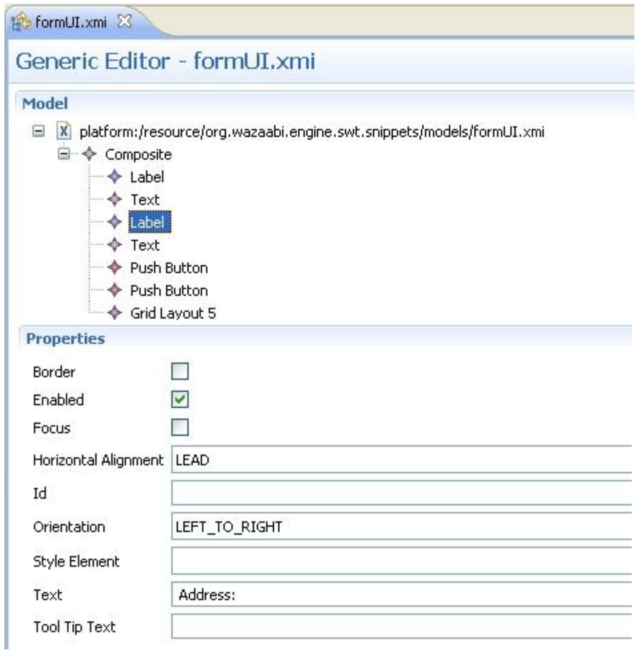
By now, we know that user interface models are rendered at runtime by targeted engines. To create a GUI, Architect can be used as a graphical editor. It is a still developing module and it is reported that it will be replaced instead of the editor itself. The user interface model is built behind and rendered by a specific engine which is loaded according to target user interface technology. At the same time, it is also possible for developers to directly build traditional user interfaces by programming[27].



```
// create a composite and set its layout
Composite composite = WidgetsFactory.eINSTANCE.createComposite();
GridLayout gridLayout = LayoutsFactory.eINSTANCE.createGridLayout();
composite.setLayout(gridLayout);
//Set the number of columns
gridLayout.setNumColumns(2);
// create two labels with text
Label nameLabel = WidgetsFactory.eINSTANCE.createLabel();
Text nameText = WidgetsFactory.eINSTANCE.createText();
Label addressLabel = WidgetsFactory.eINSTANCE.createLabel();
Text addressText = WidgetsFactory.eINSTANCE.createText();
//Set the labels
nameLabel.setText("Name:");
addressLabel.setText("Address:");
//Save & cancel button
PushButton save = WidgetsFactory.eINSTANCE.createPushButton();
save.setText("Save");
PushButton cancel = WidgetsFactory.eINSTANCE.createPushButton();
cancel.setText("cancel");
//Add to the composite
composite.getChildren().add(nameLabel);
composite.getChildren().add(nameText);
composite.getChildren().add(addressLabel);
composite.getChildren().add(addressText);
composite.getChildren().add(save);
composite.getChildren().add(cancel);
```

Figure 2.13: Editing with programming[27]

Editing declaratively means that we can declare the content of the GUI and change the design by declarative editor instead of programming the user interface such as XML-based user interface with the code.


formUI.xmi

Generic Editor - formUI.xmi

Model

```
platform:/resource/org.wazaabi.engine.swt.snippets/models/formUI.xmi
  Composite
    Label
    Text
    Label
    Text
    Push Button
    Push Button
    Grid Layout 5
```

Properties

Border	☐
Enabled	✓
Focus	☐
Horizontal Alignment	LEAD
Id	
Orientation	LEFT_TO_RIGHT
Style Element	
Text	Address:
Tool Tip Text	

Figure 2.14: Editing Declaratively[27]

The user interface of an application may include many different components, windows or forms. Documentation is generated as a separated document. Wazaabi lets the developer combine the document with the model by putting it inside the model.

To sum up, Wazaabi is an fully dynamic open source framework based on EMF models including one abstract model besides having concrete models. Developers changes and modifications reflect into models by updating the view.

2.4 Constraints of Existing Frameworks

Before closing to the State of the Art chapter by giving its summary, this section is where we are going to mention about the general constraints of the Ajax technique and the existing work of Wazaabi project respectively. While, these difficulties in both Ajax technique and Wazaabi typically concern only to developers, on the other hand some of these negative aspects that we will present here are still remaining unsolved.

2.4.1 Constraints of Ajax

Even though, the Ajax technique provides an ideal case in the development of web applications as well as it is a new technique which is growing rapidly, the new technology also comes with new issues that are needed to be solved with it. So, in this section, we would like to list down some issues in which Ajax has challenges[25]:

- **Increasement in Complexity:** The developers of the server part have to understand that the presentation logic is needed in the HTML client part as well as in the server part. Additionally, web application developers must have JavaScript technology skills.
- **Difficulty in Debuging, Testing and Maintenance:** JavaScript code is difficult to test, the automated testing for JavaScript is also difficult.
- **Immaturity of Toolkits and Frameworks:** Most of the frameworks are still in beta phase.
- **JavaScript Technology Dependency and Incompatibility:** It must be enabled for applications to function properly.
- **JavaScript Code can be Visible to Hackers:** If JavaScript code is poorly designed, then it can invite some security problems.

2.4.2 Constraints of Wazaabi

User interface design is divided into different practices and techniques. Live user interface models became necessary by the demand of many developers who are working with web technologies and these live models allow us to focus on just user interface design beyond the rest of the application architecture. For example, in Wazaabi, every changes of the model is reflected into the user interface immediately. Thus, changes can be made at run time. It does not show the details of the platform and developers do not need to know any specific knowledge about SWT or SWING. At first sight, here are the possible constraints of Wazaabi framework has:

- **Limited Number of Generic Elements:** The observations that we made from Wazaabi is that the framework has one concrete model, however it has extensibility to create other type of widgets. But, the user interface model elements of Wazaabi need to be extended rather using the core generic elements.

- **Incompatibility of Java Platforms:** It is realized that Wazaabi is able to work on J2ME when using SWT. But, the major constraint seems that most of the J2ME platforms are based on JDK 1.4. Therefore, it is not clear that some packages in the core should be changed or not for the further versions of JDK.

- **Ongoing Development of Framework:** Wazaabi 2.0 is recently introduced and still has developing modules.

- **IDE Platform Dependency:** It is not a standalone development environment. It can be installed only in Eclipse IDE as a plugin.

2.5 Chapter Summary

In this chapter, we started our discussion by giving a general information about Ajax technology in order to provide a fully understanding before we dive into the architechture and various frameworks of Ajax as a background information. That is to say, we elaborated what exactly the architecture of Ajax was, what the main components of Ajax were and how the Ajax technique made web based applications different from conventional ones by explaining the client server methodology in Ajax based web applications.

Then, we continued our discussion with examining the different types of Ajax frameworks as well as making a comparison between the existing and widely used Ajax frameworks briefly. At that point, we also gave brief information about Java based frameworks besides the former Ajax ones. In addition to the Java based frameworks, we explained the relation between Ajax and XML User Interface Language (XUL) briefly, since ZK Ajax framework utilizes XUL in the development of Ajax based web applications. After that, we started mentioning about the existing work of Wazaabi project, how it works and we explained the necessary parts of the framework to understand briefly which components does what. We also presented how the Wazaabi was closely dependent on EMF as well as mentioning about the advantages of Wazaabi while editing user interfaces and the logic behind making user interface design feasible in the user interface of an application at run time. Finally, the last issue of State of the Art chapter was providing the constraints of the existing frameworks, both the Ajax frameworks and the Wazaabi's framework itself as a background knowledge in this master thesis.

Chapter 3

Problem Analysis

This chapter gives a comprehensive analysis of the given problem. First, we are going to describe the problem briefly and continue to relate it with different modeling frameworks. Then, possible functional and non-functional requirements are elaborated by giving a short scenario about the desired situation.

3.1 Problem Definition

Model based user interface design is an important aspect of user interface design in order to acquire user choices for a system or to provide relevant information to the user in an effective way. Naturally, information about user input is required. It is usual for a human computer interaction scenario to have a GUI which has many different components interacting with the users to fulfill their requirements.

The advancement in user interface design techniques makes it possible to build well equipped GUIs which are able to make the data accessible for the user and gather vast amount of information about the user. This information from different inputs taking part in the user interface of the system comprises the context. But, there can be different contexts of use. Hence, while developing data-oriented applications, user interface should work as a layer of the current architecture providing concrete widgets from the abstract views and relating them to underlying domain data by using data binding. However, there are models which supports data binding, there exists a gap between modeling and the runtime architecture. Moreover, some frameworks which have already defined architecture may not support modeling.

3.2 Relation with Wazaabi

One of the most important ability of Eclipse Modeling Framework (EMF) with its tools is to view the runtime state of the application based on model-driven runtime architecture. EMF supports wide range of technologies for managing, navigating, querying, storing and sharing EMF object structures.

These technologies like EMF can enable and simplify the implementation of many interesting functionalities. Regarding to its functionalities, with the help of EMF, Wazaabi framework represents a java XUL engine for building concrete user interfaces by having an EMF based user interface model with the editor and runtime support based on the Eclipse platform.

The runtime allows the developer to run user interfaces within Eclipse, however it does not support all web components for web browser based user interfaces. Therefore, it is necessary to design and implement a web technology support for building web browser based user interfaces, for example by using Ajax technology.

While focusing on the Wazaabi framework intensively, we realized that Wazaabi gives an opportunity to build live user interface models on its framework. It is known that it supports targeted user interface's models based on EMF to build real user interface of an application, but it is obviously a question that should be answered how Wazaabi can support various user interface models rendered by its core engine. In addition, it is also necessary that how the Ajax tools and its framework can be integrated to Wazaabi. This will be mentioned in the next chapter in detail.

3.2.1 EMF Model

EMF allows the developers to create various domain models which contains data related to specific domain. For example, in your domain model, normally you can create classes as well as defining properties or adding new children to them. In eclipse while dealing with EMF, there is a difference between the underlying model and the current one. As we mentioned before, EMF has a model notion and a model is the instance of this core model which describes structure of the model. In fact, EMF has two models which are Ecore model and Genmodel. Ecore model shows the details of already created classes and lets us to define different elements. Genmodel has information about generation of the code and the path of the file. Basically, EMF provides the ability to declare model element;

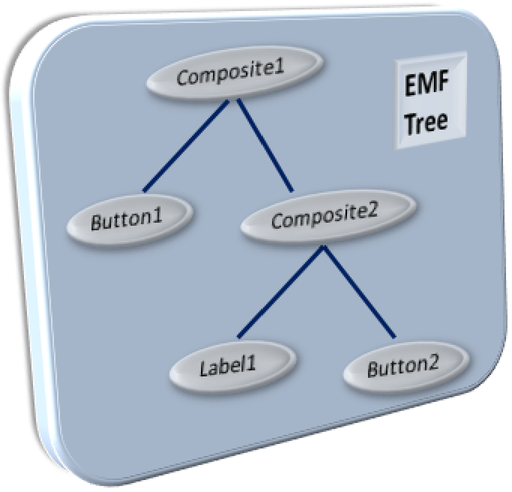


Figure 3.1: EMF Tree

The EMF tree shows a root element representing the whole model. This model can have more than one child which is based on the packages and thereby represents the classes they are belong to. Meanwhile, the attributes of these classes are represented by their children. As an advantage, EMF allows you to create explicit domain tree models as well as generating the Java code from the model at any time.

3.2.2 SWT Model

Since Wazaabi simplifies the complexity of creating underlying models of the existing platform, the designers do not need to know any specific knowledge about SWT. Thus, designers become developers who can edit different kinds of supported user interface models. With convertions, they are able to modify their model through an SWT model. Behind this simplification, the related engine renders a view of the EMF model by converting it to SWT model. See Figure 3.3. Then, this SWT model is rendered in a Wazaabi view which updates the real user interface.

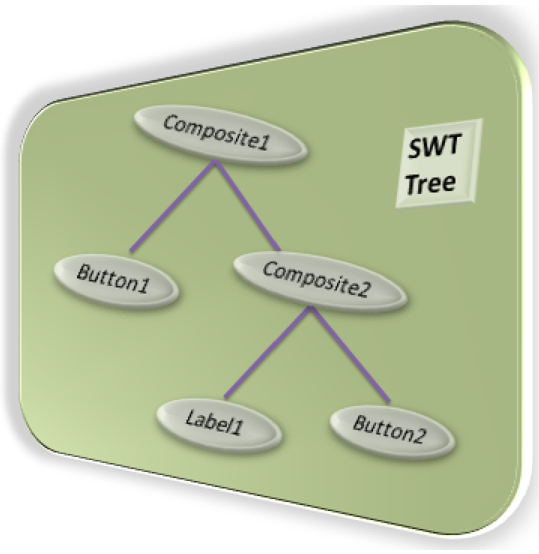


Figure 3.2: SWT Tree

3.2.3 Wazaabi Scenario

In this section, in order to gain a better understanding about the relationship between EMF and SWT models, we will give a scenario that can be considered as a SWT implementation for the rest of the section. This scenario allows the reader to elaborate the intended functionalities of the existing framework and how the model based support benefits the user during the design. This scenario will also be used to determine functional and non-functional requirements.

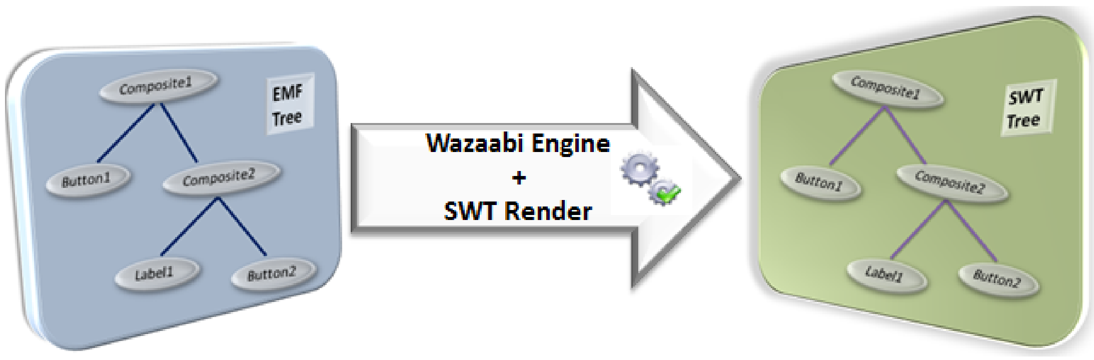


Figure 3.3: Relationship between EMF and SWT Trees

In this scenario, we assume the SWT Text widget as a user interface element. Then, we will show how the Editpart will deal with the SWT widget and how the user interface will be changed according to updates coming from the model side.

When modeling a user interface in Wazaabi, it means that an underlying model is designed. After the Wazaabi palette is loaded with user interface components of a target model, the Wazaabi engines are formed by Editpart (true controller) realizing the relationship between the user interface model and the real user interface.

The engines are designed with core Editpart connecting to the core model and they have target user interface Editpart (in this case for SWT) which connects to SWT model. A hierarchy can be seen clearly between the Abstract Editparts and Widgetviews.

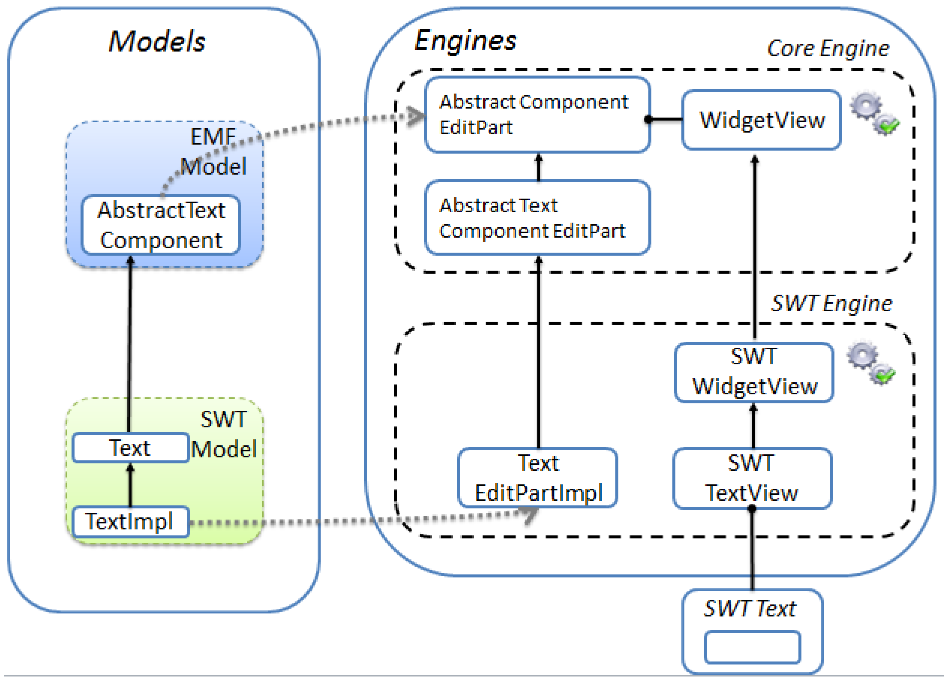


Figure 3.4: Wazaabi Scenario[2]

The Wazaabi editor allows the user to make live changes according to target model. On the other hand, Wazaabi works with two viewers; the Wazaabi viewer (hidden) and Graphical Editing Framework (GEF) based viewer (main editor). The target model (in this case SWT) is rendered in Wazaabi viewer then the same changes from this viewer is copied to GEF viewer.

Actually, Target model (it can be SWT or JSF) is rendered into the GEF editor, however it is not the main source model. The target model is in the View that is shown, but the real edited model is the source model (EMF tree) that is edited.

1. A user edits a user interface element from the palette in the editor.
2. A modification is issued on the editor.
3. This update in user interface is reflected to EMF tree by modifying the child on the parent component of the EMF tree automatically.

4. Render (Engine) traverses the EMF tree in order to find which node is changed.

5. EditPart listens any node changes in EMF tree and it makes sure that the View is updated.

6. When the View is updated, it informs the Render (Engine) that the model has modified. Then, the View updates the user interface according to changes in EMF tree.

7. Consequently, the Wazaabi viewer redraws the user interface and these changes are shown in the user interface dynamically.

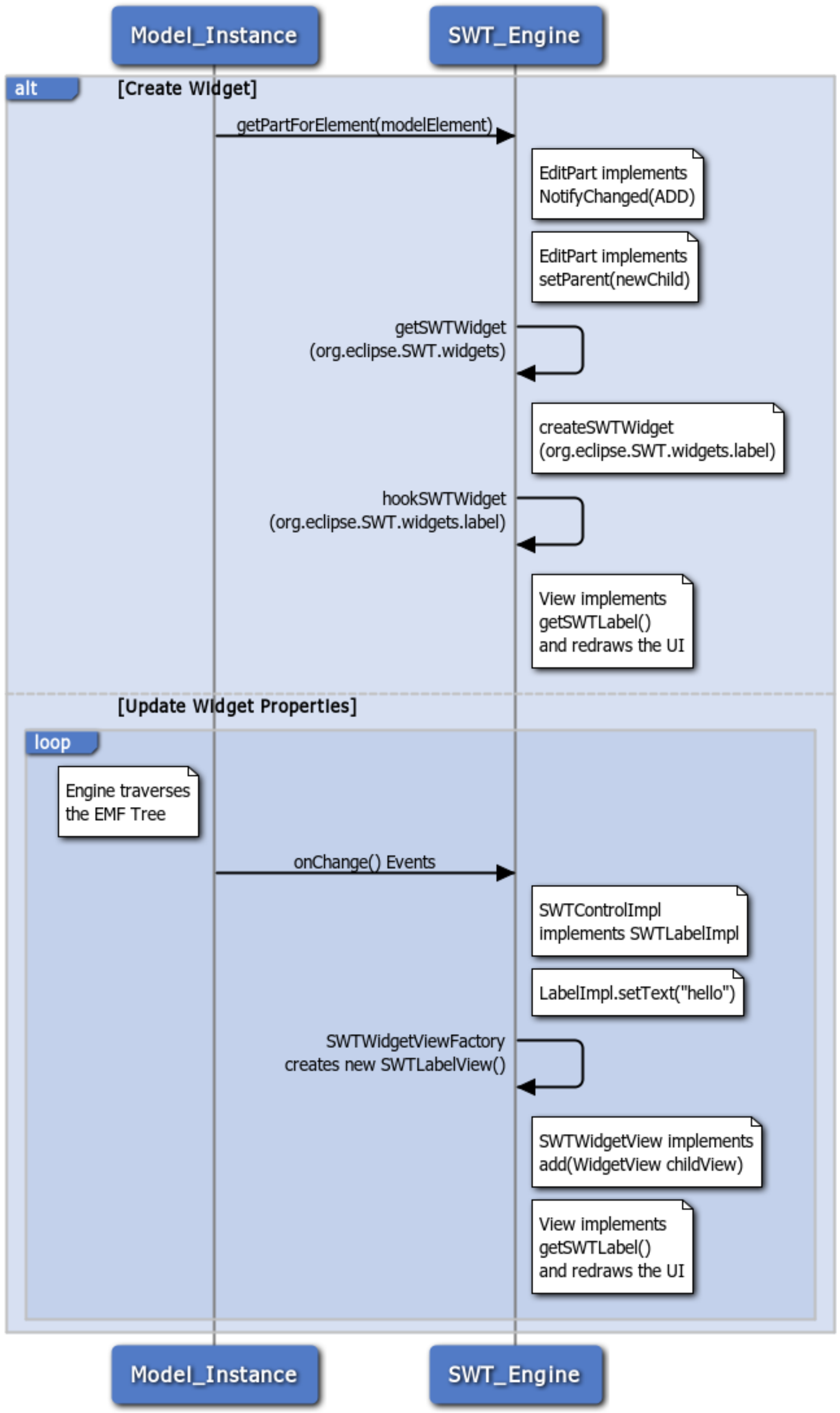

Model_Instance
SWT_Engine
alt
[Create Widget]
getPartForElement(modelElement)
EditPart implements NotifyChanged(ADD)
EditPart implements setParent(newChild)
getSWTWidget (org.eclipse.SWT.widgets)
createSWTWidget (org.eclipse.SWT.widgets.label)
hookSWTWidget (org.eclipse.SWT.widgets.label)
View implements getSWTLabel() and redraws the UI
[Update Widget Properties]
loop
Engine traverses the EMF Tree
onChange() Events
SWTControlImpl implements SWTLabelImpl
LabelImpl.setText("hello")
SWTWidgetViewFactory creates new SWTLabelView()
SWTWidgetView implements add(WidgetView childView)
View implements getSWTLabel() and redraws the UI
Model_Instance
SWT_Engine

Figure 3.5: Sequence Diagram of Wazaabi Scenario

With respect to the sequence diagram of Wazaabi scenario above, the following figure elaborates the hierarchical relationship between the crucial parts in Wazaabi. When a change is occured in the model, the following parts are affected by the change because any of the nodes in the model instance are mapped into the other parts in hierarchy respectively. For instance, when View of the component draws the SWT component, it needs to retrieve the corresponding Editpart of the current component.

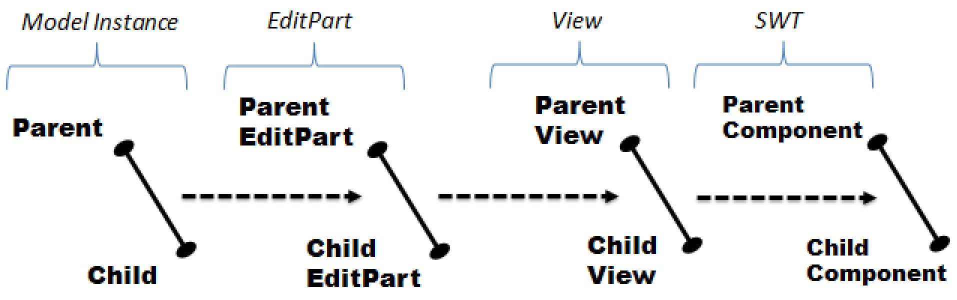


Figure 3.6: Hierarchical Structure of Wazaabi

3.3 Relation with Ajax

As we already mentioned about the Ajax technology which is a new trend in the field of web application development, such that it enables web based applications or web pages to call the web server without any need to refresh or leave the actual web page. This main mechanism is generally done in the background (asynchronously) without notifying to the user in order to provide both more user friendly and natural interaction in the user interface as well as making more interactive web based applications. Additionally, web pages which are embedded with snippets of Ajax code blocks, are able to access remote services such as web services or databases while avoiding to refresh the actual web page as well as preventing slow response times in the web based applications. One of the most important feature that Ajax based applications use is the XMLHttpRequest object which allows access to remote services either synchronously or asynchronously.

The main objective of the Ajax frameworks is to provide the engine with functions that are triggered by the events which are coming from the client part. Besides, there are other advantages in using Ajax frameworks. In the client part, variety of Ajax events will be triggered that will send various requests to the server part. In the server part, the requests will be processed according to the pre-defined rules of the engine, such as the data will be searched and transported. Eventually, the workload of the programming will be greatly reduced.

Without using any frameworks, the Ajax technique would be much more difficult to work with. Because, these frameworks are powerful, and they bring a great deal of functionality to the whole system. Plus, the Ajax frameworks are used to ease many of the technical problems that Ajax technique usually faces with.

3.3.1 Ajax Support

There are a number of ways in which Ajax technique can be used to support the web based applications that we already use today. If we take into account the desktop counterparts of the web based applications, we can easily understand the fact that the desktop applications are generally more robust and efficient than web based ones. While the web based applications have continued to become more advanced over the last decade, this fact remains that they are much less responsive than desktop tools. With the introduction of Ajax technique, the gap between the things that users experience on the desktop applications and the things that they experience on the web based applications is started to close. Real time suggestions of the Google Web Search can be a good example to this, if we think about how the suggestions for keywords are updated as fast as we type. Google Maps can be an another well-known example to this, as long as we use the cursor to scroll around the map and zoom in, all updates happens so quickly without waiting for new web pages to load.

At this point, we would like to put emphasis on the proper definition of Ajax again which is not technically a single technology itself. In fact, it is a group of technologies, and all of them can be used independently from each other. In order to understand how Ajax support the web based applications impressively, we would like to bring back how the existing web operates shortly.

With the classic web, most interactions are carried out by the user will initiate a request via HTTP. This request will go through the web server, and the server part will be responsible for processing the information such as gathering data, and communicating with multiple legacy systems. Once it has done all of these things, it will send the information back to the client part in the form of HTML.

It must be noted that, the classic web processes can be useful from a technical approach. Nevertheless, the majority of the users who use web based applications are not interested in technical issues as long as they do not make any sense to focus on the experience of the users. Because, the user is still waiting, while the server part performs various tasks in the background. Additionally, we all know that the users are always looking to gather information as rapidly as possible, and the waiting has always been an annoying thing that we want to avoid.

As we already mentioned before, the usage of the Ajax technique is very crucial for the web based applications to make them become faster and more interactive. For all these reasons, Ajax support is very significant, because it supports an engine that can act as a mediator between the server part and the user's web browser (or the client part generally). Incidentally, it can be thought that adding an Ajax engine to the web based applications could slow down the response time, in fact, this is not the case. As the user starts new sessions, the Ajax engine will be reloaded rather than the web browser. It will be concealed in a frame as well as it will assist in loading the user interface in order to communicate with the server part.

Finally, we can say that, the Ajax support provides to bring back the user friendly features of the desktop applications and utilize them over the web based applications as well as making the internet faster, more dynamic, and more interactive.

3.3.2 Proactive Ajax Approach

To bind user interface design and relevant data together is a result of the modeling. As we already mentioned before, the main objective of the existing work of Wazaabi project is to assist user interface design for the developers. It generally supports the variety of user interfaces which are all based on EMF model. Additionally, the Wazaabi has its own engine which is based on SWT, and it acts as a mediator between EMF models and SWT models of the same user interface. Before the implementation of our support to the existing project of Wazaabi, its general model can be figured as the following;

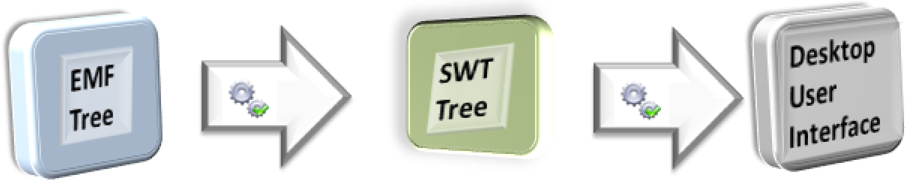


Figure 3.7: Existing Model

Since, the current version of the existing Wazaabi project is lack of web browser support and the main logic behind the architecture of the Ajax engine is to make use of an Ajax framework that we can reuse it every time when we want some asynchronous processing or when we need a smarter way to refresh information on the current web page. We would like to go beyond the idea in the current version of the Wazaabi project by combining it with Ajax support in order to use its functionalities in the web browser based user interfaces with the power of Ajax technology. Additionally, the proactive Ajax approach will also be the simplification of the code parts, where is needed for particular functions on the client part according to the main benefit of the Ajax engine.

As we already mentioned in the previous sections, the existing engine of the Wazaabi project is based on SWT. Even though, the existing work of Wazaabi project supports model based user interface design properly, it still needs to be extended with the Ajax technology in order to fully provide the required support for making its user interface to become similar to its counterparts which are running on the web based applications. Shortly, we would like to put this idea into practice by replacing the existing SWT engine of the Wazaabi with an Ajax engine by using ZK Ajax framework. Therefore, after the implementation of the Ajax support to the existing project of Wazaabi, the proactive model can be figured as the following;

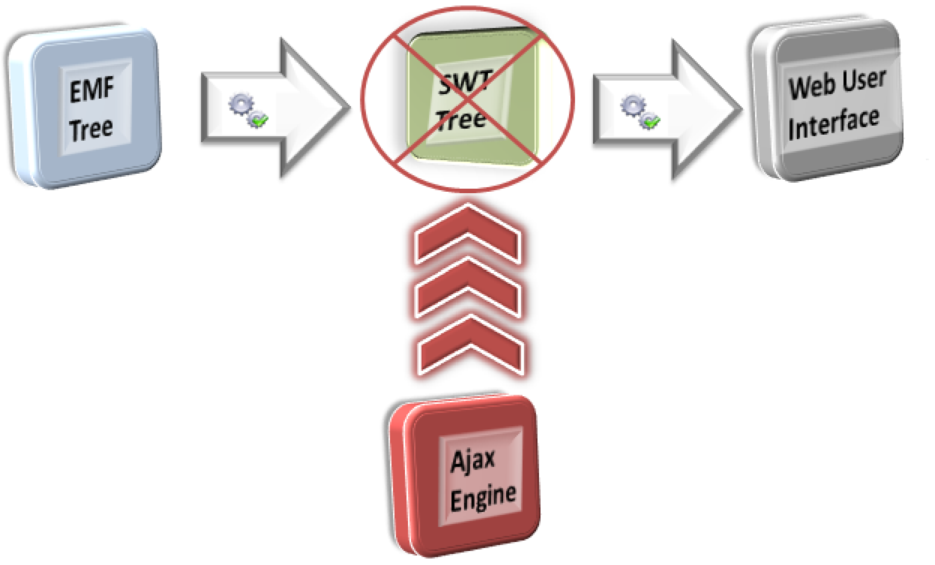


Figure 3.8: Proactive Ajax Approach

3.4 Ajax Scenarios & Sequence Diagrams

This section provides all of the scenarios in order to give as realistic as possible situations where the Ajax technique is being used. In order to illustrate the way of use, screenshots of the running web applications and their all feasible cases are also given in this section. Later on, the sequence diagrams of each scenario will also be provided in order to elaborate the Ajax logic behind the web applications that we selected to explain in a more detailed approach. Finally, to get a fully understanding of how the Ajax technique works in our scenarios which are based on various web applications, their all related code groups (XMLs and Java source codes) can be found in references which are located in the end of the entire report.

3.4.1 Ajax Scenarios

The Ajax scenarios is where we introduce generic possible cases about Ajax based web applications which can be found in most websites. We will also elaborate our knowledge about the logic of Ajax behind each scenario which are necessary to get a fully understanding of how they actually work step by step. Finally, this section also covers the relevant screenshots of the running Ajax based web applications which we selected to examine as our Ajax scenarios for making them more sense.

Scenario 1: Ajax Push and Animation: Drag & Drop List

The first scenario demostrates a realistic and widely used case where the items of two listboxes can be reordered by dragging and dropping the listbox items around dynamically by using its model. In other words, it is a comprehensive illustration of the ZK components can be draggable and droppable.

In this Ajax scenario, we have two seperated listboxes which both includes various dynamic (draggable and droppable) ZK items. According to the main logic behind the Ajax technique, any changes in the order or item(s) each listboxes are shown immediately without refreshing the entire web page. In this simple and widely used scenario, there are two main options for the user, which are either moving the item from one listbox to another by using both listboxes while doing the drag and drop operation or reordering the items in the current listbox by doing the same drag and drop operation in the same listbox. Therefore, we will elaborate them separately to make the first Ajax scenario more sense.

To make it more sense, we would like to illustrate how *ZK GWT* item can be moved from the left listbox to the right listbox, then how the new position of *ZK GWT* item can be reordered in the right listbox. The first case and the new position of the *ZK GWT* item are both shown in the figure below;

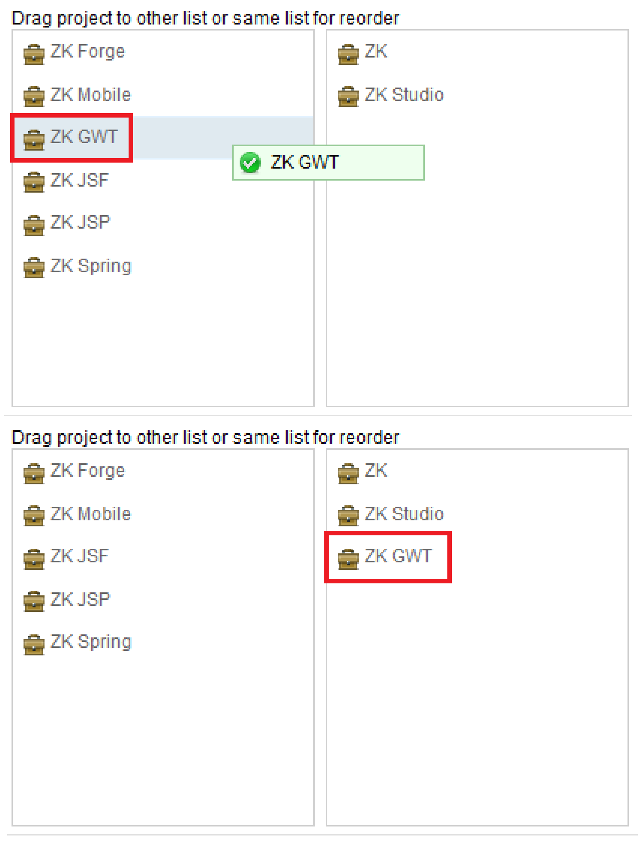


Figure 3.9: Drag and Drop Case 1

After moving the *ZK GWT* item to the right listbox, we would like to reorder the current items of the right listbox by dragging and dropping the *ZK GWT* item upward. The second case and the new position of the *ZK GWT* item are both shown in the figure below;

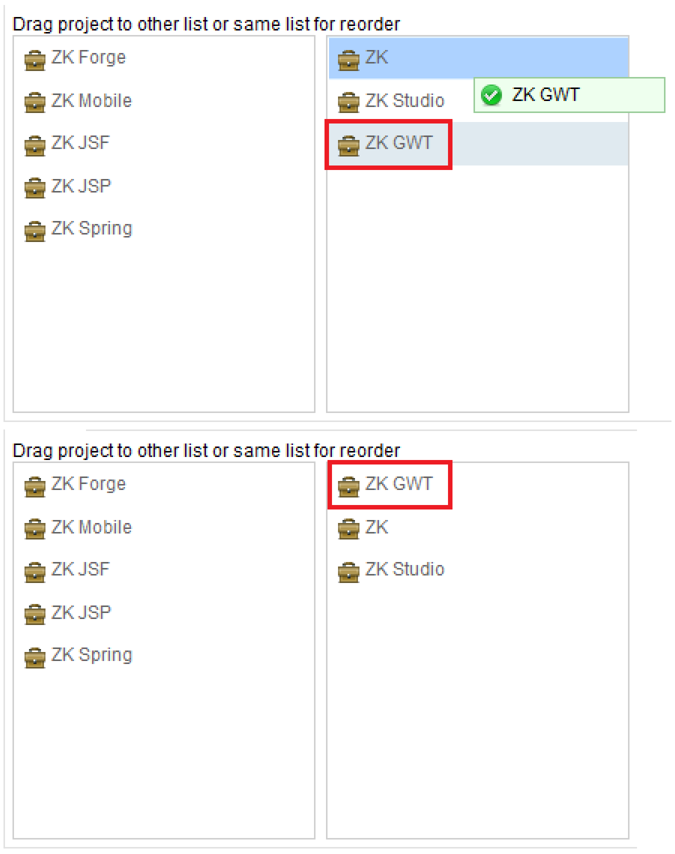


Figure 3.10: Drag and Drop Case 2

Apart from these two possible cases that we have shown in the first scenario above, there is an additional but, an exceptional case which can be tried to do by the user. Even though it is an exception, we think that it is as important to cover as the other two possible cases above. The user can try dragging and dropping the item towards the outside of the listboxes, despite of it is not allowed to do. Therefore, we would like to illustrate this exceptional case by taking into account in the figure below;

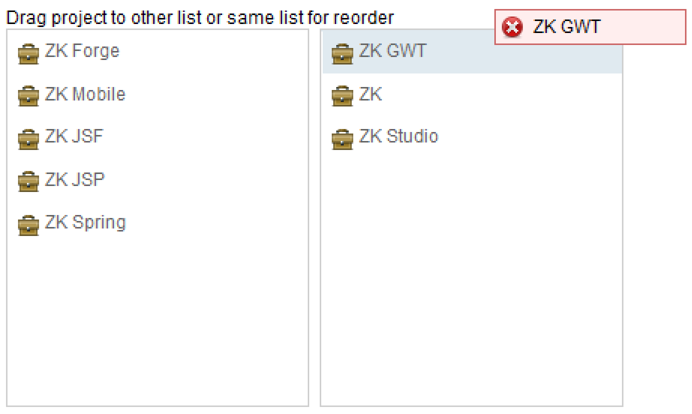


Figure 3.11: Drag and Drop (Exception)

In order to get a fully understanding of how actually the drag and drop list scenario works step by step, we are going to explain the Ajax logic behind it which is dependent on the ZK Ajax framework now. The sequence diagram of this scenario illustrates the steps of the mechanism in Figure 3.20.

1. The user selects a random item from one of the listboxes and either drags and drops it to the other listbox or reorders it in the current listbox.

2. Therefore, the onDrop event of the selected item is triggered and the ZK Client engine sends a request to the server part in order to execute the relevant method for triggered event.

3. In the server part, the ZK Asynchronous Update Engine receieves the request which is sent by the ZK Client engine.

4. Then, The the ZK Asynchronous Update Engine updates the event queue of the server part by adding the triggered event in order to find the relevant method of *move(event.dragged)* and sends it as a response to the client part.

5. The ZK Client engine receives the response and updates the content of the web page by executing the method of *move(event.dragged)* without refreshing the web page entirely.

At this point, we would like to give the XML structure of the *ZK GWT* item which we selected to illustrate our scenario. In order to implement the drag and drop feature properly, we need to assign the method of *move(event.dragged)* to the onDrop event of each item. The following figure shows how the XML structure of the *ZK GWT* item is;

```
<listbox id="left" height="250px" width="200px"
onDrop="move(event.dragged)" droppable="true" oddRowSclass="non-odd">
...
<listitem
draggable="true" droppable="true" onDrop="move(event.dragged)">
<listcell src="/img/Centigrade-Widget-Icons/Briefcase-16x16.png"
label="ZK_GWT"/> </listitem>
...
</listbox>
```

Figure 3.12: XML Code of ZK GWT Item

Additionally, we would like to mention about how the method of *move(event.dragged)* works. If the dragged item's parent exists in the same listbox, we insert it before its parent as a new sibling (the reordering case) when it is dropped. Otherwise, we insert it as a new child of the other listbox (the moving case) when it is dropped. Therefore, the method of *move(event.dragged)* is given in the following figure and the full version of the source code can be found in [28].

```
void move(Component dragged) {
if (self instanceof Listitem) {
if (dragged.getParent().getId().equals("right")) {
self.parent.insertBefore(dragged, self.getNextSibling());
} else {
self.parent.insertBefore(dragged, self.getNextSibling());
}
} else {
self.appendChild(dragged);
}
}
```

Figure 3.13: move(event.dragged) Method

Scenario 2: Ajax Events and Scripts: Keystroke Events

In the second Ajax scenario, we have a simple textbox which requires the user to enter a random password. As soon as the user presses particular keys, the relevant messageboxes are shown according to the pre-defined methods for each particular key. According to the main logic behind the Ajax technique, the relevant messageboxes are shown immediately without refreshing the entire web page. In order to be more specific, we would like to demostrate four main optional cases of the KeyStroke event which are all triggered after each particular key press of the user in the rest of the scenario. The user can press Enter, Escape (esc), Ctrl+A and F8 keys to receive relevant messageboxes for each key respectively. The resulted screens of the running methods which are all triggered by the KeyStroke event are shown in the figure below;

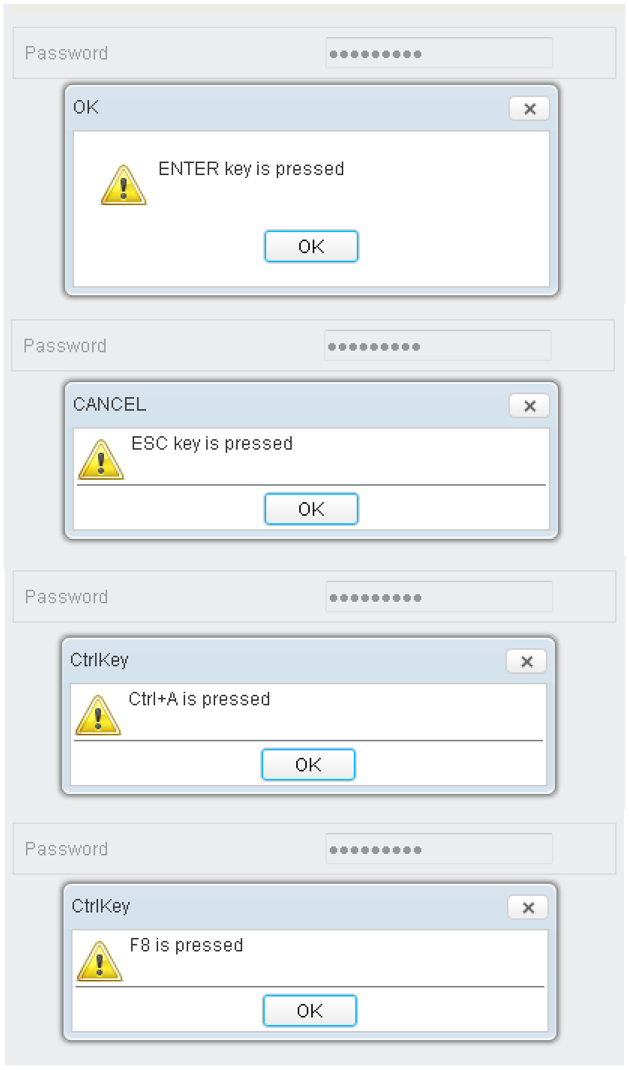


Figure 3.14: KeyStroke (Event)

In order to get a fully understanding of how actually the keystroke events scenario works step by step, we are going to explain the Ajax logic behind it which is dependent on the ZK Ajax framework now. The sequence diagram of this scenario illustrates the steps of the mechanism in Figure 3.21.

1. The user starts entering a random password by pressing keys from the keyboard as inputs until pressing particular keys which are Enter, Escape (esc), Ctrl+A and F8 keys.

2. As soon as the user presses a pre-defined key, the KeyEvent of the pressed key is triggered and the ZK Client engine sends a request to the server part in order to execute the relevant code part for the triggered event..

3. In the server part, the ZK Asynchronous Update Engine receieves the request which is sent by the ZK Client engine.

4. Then, The the ZK Asynchronous Update Engine updates the event queue of the server part by adding the triggered event in order to find the relevant method of *((KeyEvent) event)* and sends it as a response to the client part.

5. The ZK Client engine receives the response and updates the content of the web page by executing the method of *((KeyEvent) event)* without refreshing the web page entirely.

In this scenario, we have focused on the input as key presses by the user and have tried the particular keys Enter, Escape (esc), Ctrl+A and F8 keys in order to see the relevant response which are supposed to come from the server part. The KeyStroke events are only supported for the Enter, Escape (esc), Ctrl+A keys which are handled by onOK, onCancel, and onCtrlKey respectively. The F8 key is treated as the same with Ctrl+A key. In order to get a fully understanding of how exactly this Ajax scenario works, the required method of KeyStroke event which is embedded in the XML structure of the textbox component is given in the following figure and the full version of the source code can be found in [29].

```
<textbox id="inp" ctrlKeys="^a#f8" type="password"
value="123456789" width="150px">
<attribute name="onOK"><![CDATA[
Messagebox.show("ENTER␣key␣is␣pressed", "OK",
Messagebox.OK, Messagebox.EXCLAMATION);
self.focus();
]]></attribute>
<attribute name="onCancel"><![CDATA[
Messagebox.show("ESC␣key␣is␣pressed", "CANCEL",
Messagebox.OK, Messagebox.EXCLAMATION);
self.focus();
]]></attribute>
<attribute name="onCtrlKey"><![CDATA[
int keyCode = ((KeyEvent) event).getKeyCode();
String s = "";
switch(keyCode){
case 65: s = "Ctrl+A";break;
case 119: s = "F8";break;
}
Messagebox.show(s+"␣is␣pressed", "CtrlKey",
Messagebox.OK, Messagebox.EXCLAMATION);
inp.focus();
]]></attribute>
</textbox>
```

Figure 3.15: KeyStroke Event Method

Scenario 3: Inserting values with autocomplete feature of Combobox

In this scenario, we are going to present how to input in Combobox via autocomplete feature. Nowadays, almost every website uses autocomplete feature in their search options and this feature has quickly been popular and become a standard. Generally, there can be seen as a combination of a textbox with a listbox which allows us to select items from the list or to enter a new item. If the user enters a single letter into the textbox then suggested items related to user input are arranged in the listbox automatically.

To implement the autocomplete feature, we need to listen the onChanging event and change different child elements in the combolist according to related content of the input that the user is entering so new suggested items can show up automatically for each key pressing.

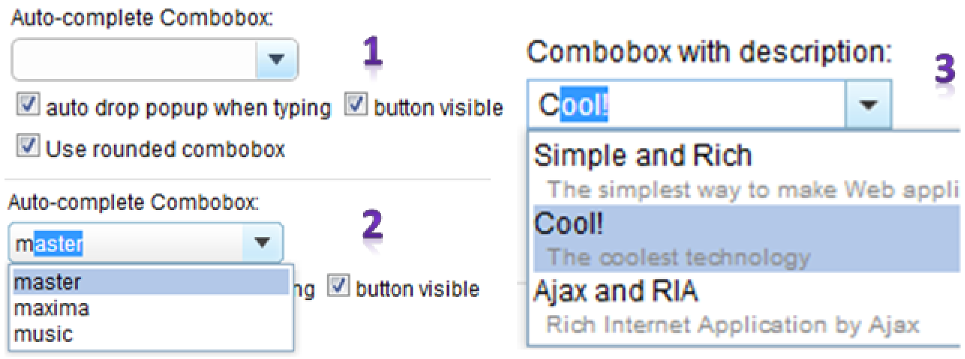


Figure 3.16: Autocomplete with Combobox

1. The user enters one or more letters or a text in the Combobox.
2. Each triggered events are detected and sent by the engine in the browser to the server.
3. The server side has a dictionary consisting of all possible suggestions that are used to speed up the search. Normally, the suggestions are loaded from a database or a web service.
4. The server receives and updates the content of the search and then sends a response to the client.
5. The engine at client receives the response and updates the content of the HTML page.

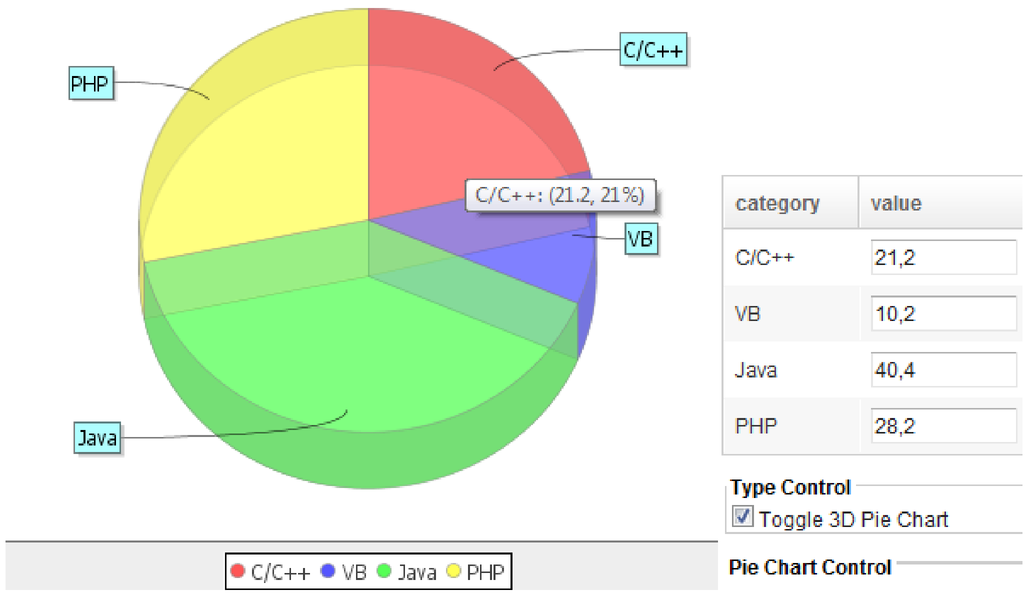
```
public void onChanging(InputEvent evt)
{
  if (!evt.isChangingBySelectBack())
    refresh(evt.getValue());
}
```

Figure 3.17: isChangingBySelectBack Method

As we can see in the code above, it is possible to provide autocomplete feature with the onChanging event. In the code, the suggestion list can be generated according to most hits on the search engines or an existing list can be loaded from a web service. Here is the part of the code that explains how to use the onChanging event to implement it with a combobox. The full version of the source code can be found in [30].

Scenario 4: Updating values of a Pie-Chart 3D

This scenario shows a realistic situation where a basic pie chart can be edited dynamically by using its model. In order to explain how the Ajax technology works, related code groups (XML and Java source code) are also given in [31].



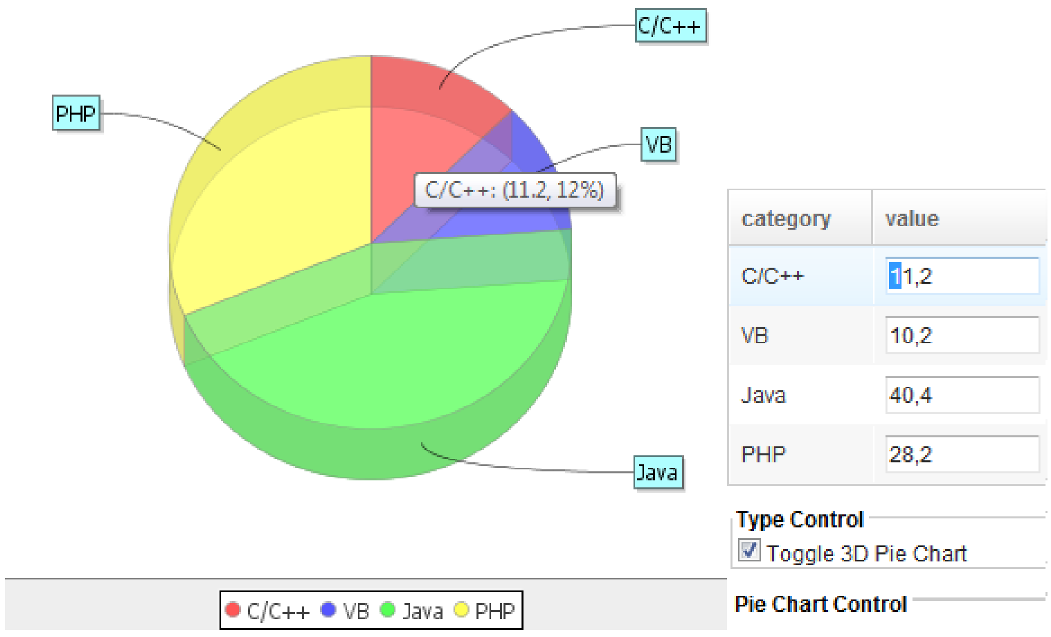
category	value
C/C++	21,2
VB	10,2
Java	40,4
PHP	28,2

Type Control
[✓] Toggle 3D Pie Chart

Pie Chart Control

Figure 3.18: Pie Chart 3D(Before)

As we see in the figure above, there are four programming languages as categories of a simple pie chart. Both categories have different values and they are displayed as set of data graphically. In this scenario, updates from the value of the category on the control panel affect the chart immediately without refreshing the site completely. As the changes are occurred over time, the chart will be redrawn continously. Actually, updating is based on the onChange event which is fired to trigger an update when the value is changed on the control panel. We can also hover the cursor on the categories or click the portions on the pie and get desired values of clicked category with a messagebox at the same time.



category	value
C/C++	11,2
VB	10,2
Java	40,4
PHP	28,2

Type Control
[✓] Toggle 3D Pie Chart

Pie Chart Control

Figure 3.19: Pie-Chart 3D(After)

Behind the scene, logic varies depending on which Ajax framework we used and in our case we are dealing with ZK Ajax Framework. Thus, we are going to explain the mechanism step by step in a recursive manner (it can be repeated during the updating process). The sequence diagram of this scenario illustrates the steps of the mechanism in Figure 3.23.

1. The loader gets and interprets the URL request and generates a HTML page which contains different standards including Javascript and ZK components at the server side.

2. The loader sends the HTML page to the client and its ZK engine.

The engine detects Javascript events occurring in the browser and it is responsible for receiving events and updating the content of the web page.

3. The triggered events are sent by ZK engine back to the server which has another ZK engine called AU (asynchronous update) engine.

4. The engine at server side takes this request and updates the properties of components then sends a response back to the client.

5. The engine at client side takes the response and updates the content in the DOM (Document Object Model) tree.

3.4.2 Sequence Diagrams

Scenario 1: Ajax Push and Animation: Drag and Drop List

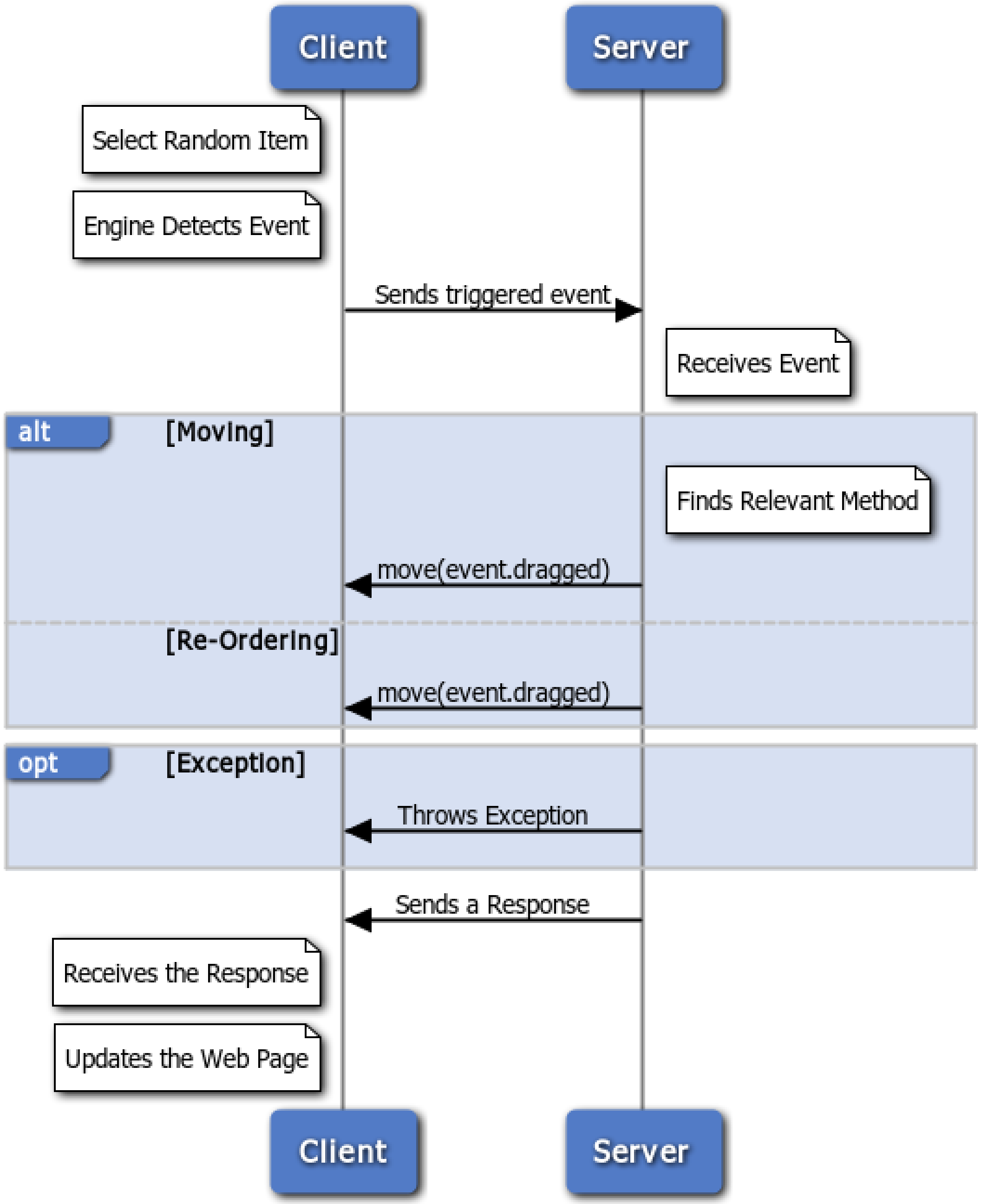


Figure 3.20: Sequence Diagram of Scenario 1

Scenario 2: Ajax Events and Scripts: Keystroke Events

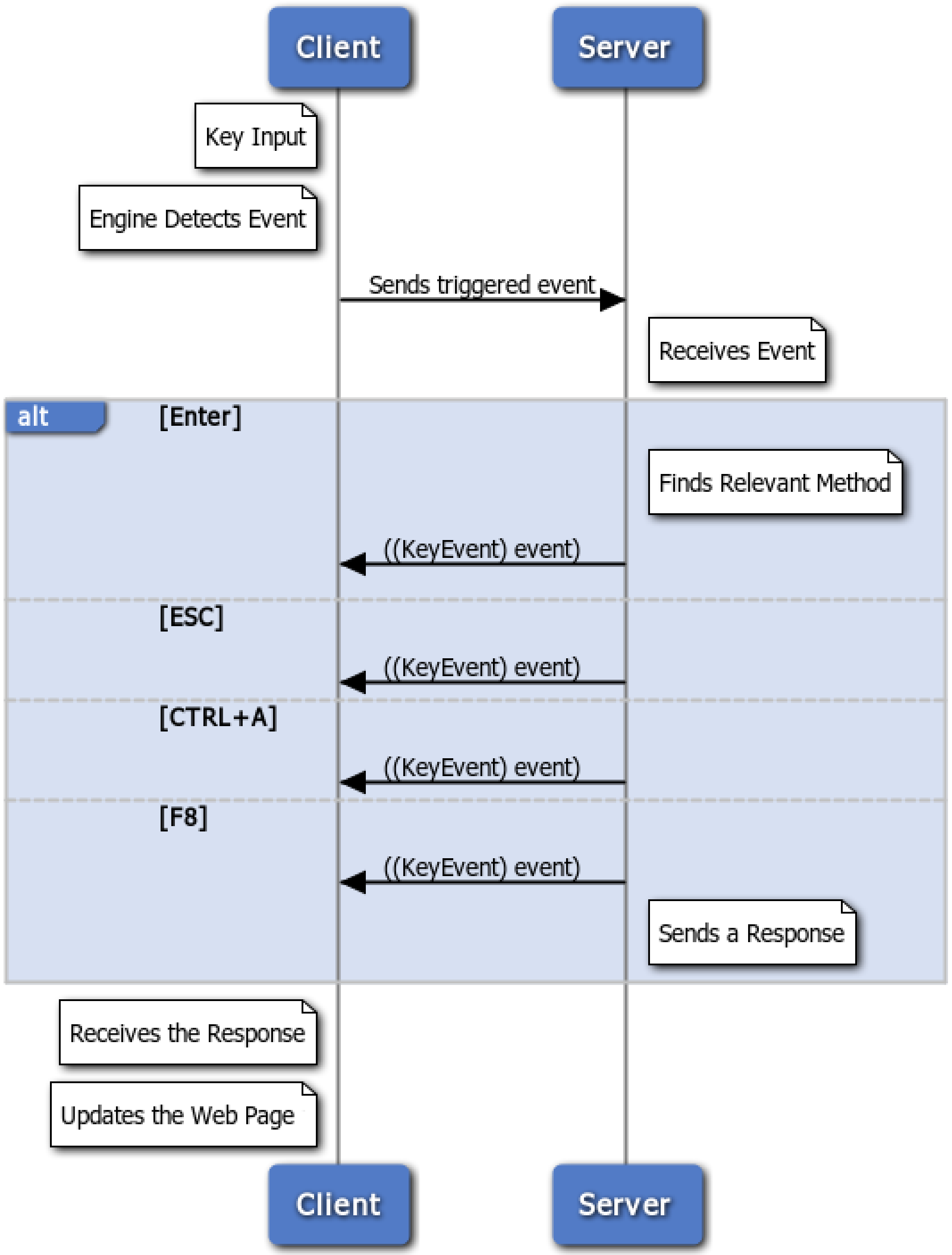


Figure 3.21: Sequence Diagram of Scenario 2

Scenario 3: Inserting values with autocomplete feature of Combobox

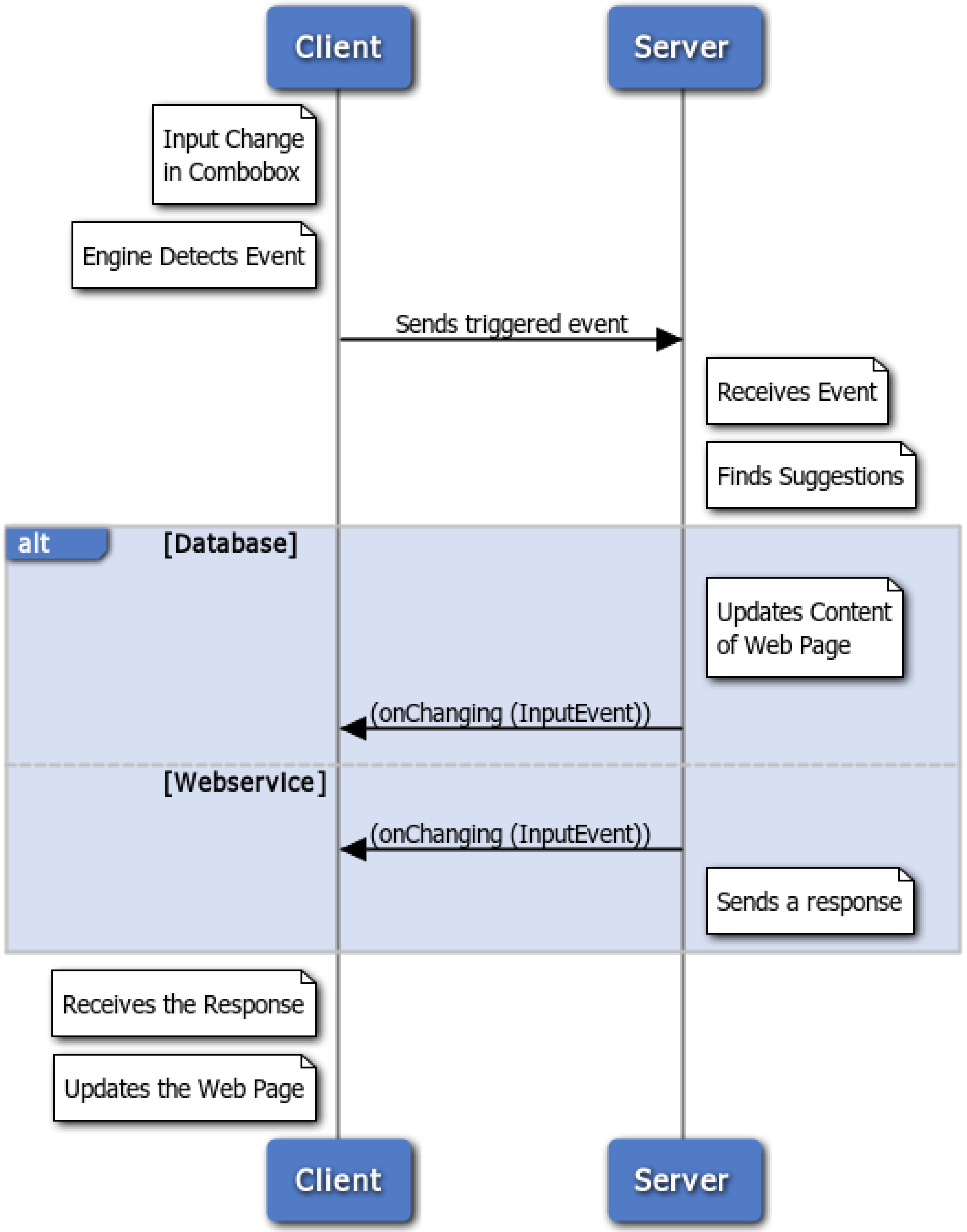


Figure 3.22: Sequence Diagram of Scenario 3

Scenario 4: Updating values of a Pie-Chart 3D

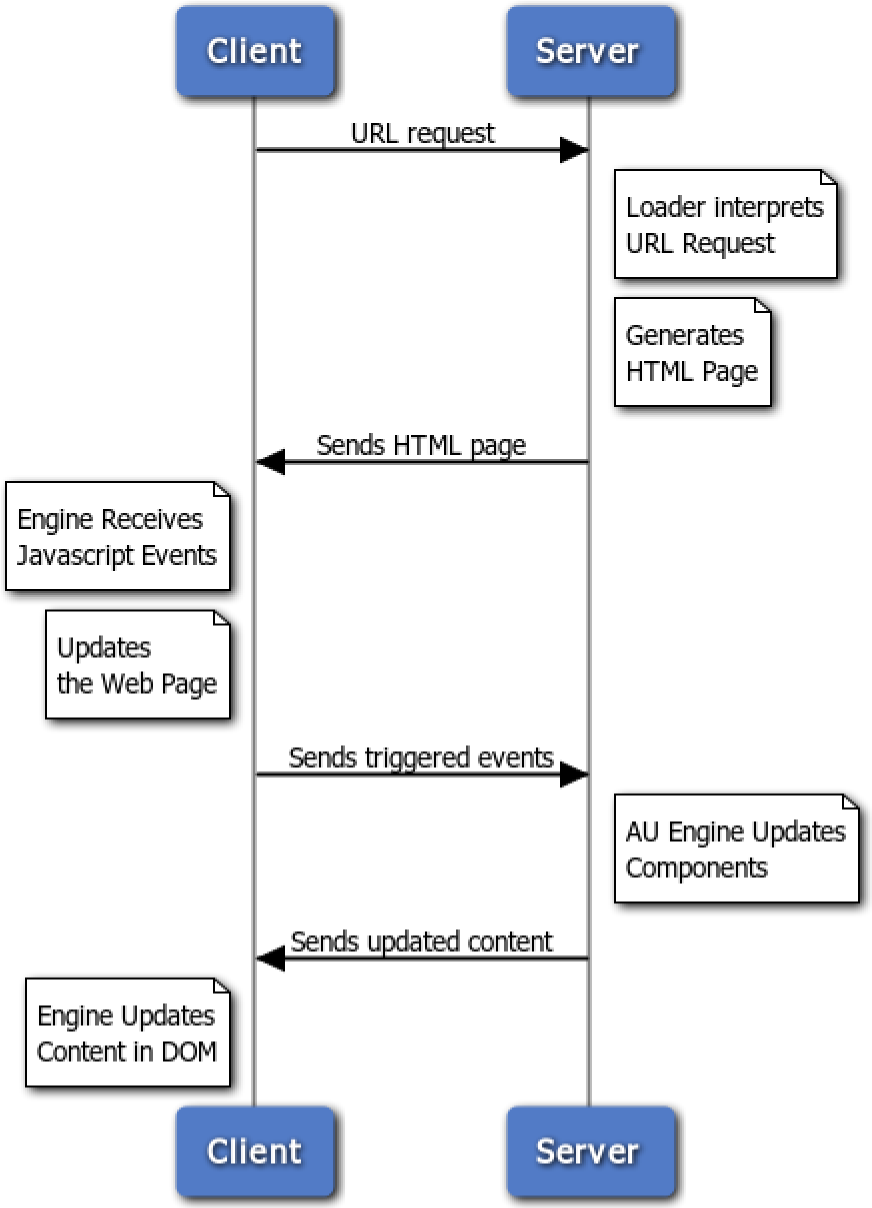


Figure 3.23: Sequence Diagram of Scenario 4

3.5 Requirement Specification

This section is separated into two parts in order to emphasize the situations in which the user can benefit from using the proposed support. Generally, these requirements that we found help us for gaining a better understanding about the actual needs of the problem. They are analyzed in two parts; the one for the functionalities of Ajax frameworks provided in various alternatives like ZK, and the other one is the specific requirements that are for the desired scenario (wanted situation) which is derived from the Wazaabi scenario that we stated in the last section. Both of them will be shown in the same requirements table which contains different abbreviations. Each requirement has a short id which symbolizes its name and requirement type. For example, FR-AJX1 means first functional requirement of Ajax framework which is decided to implement.

3.5.1 Functional Requirements

Requirement	Priority	Description
FR-AJX1	High	A new Ajax engine (render) is required.
FR-AJX2	High	Editparts must be changed specific to Ajax framework.
FR-AJX3	Medium	Ajax widgets must be integrated into the new engine.
FR-WZB1	High	Wazaabi must support to create new Ajax UIs.
FR-WZB2	Medium	Wazaabi must be able to edit different Ajax UI components.

Table 3.1: Functional Requirements

Before we begin to work properly with Ajax technique, JavaScript must be enabled on all kind of browsers of which Ajax works on all Internet Explorer 7.0 or newer and in Mozilla browser 3.0 or newer, Netscape 7.0 or newer and Safari 1.2 or newer versions. Browser scripting caching on all kind of web browsers must be activated. Even though, Ajax where JavaScript drives the data, has lot of advantages over other old HTML applications, there is a big drawback for Search Engine Optimization (SEO) of a website, because unfortunately the search engine does not index pages which are dynamically generated by Ajax.

According to the defined requirement FR-AJX1, we should first take into account that setting up a new Ajax engine (render) from the scratch is a very difficult thing to do. Because of this, there are lot of pre-made engines (render) where developers can download and use. The difference that will emerge with the implementation of Ajax will be the coding method, the type of data transmission between the client part and server part and the way in resulting the changes properly in the web browser based user interfaces.

For the defined requirement FR-AJX2, we all know that the Ajax technique is using XHTML in the view layer, Dom and SDOM for the data presentation. Therefore, it uses XML data exchange and XMLHttpRequest as the exchange engine which combines everything into one part. Traditionally, what happens in the HTML code will send JavaScript requests to the server part, then the server part will elaborate and process it or send out to an another scripting language which returns it to an HTML page which is supposed to be shown in client's web browser. This method usually retrieve the most up-to-date data from the server part, and it normally refreshes or reloads another HTML file.

Regarding to the defined requirement FR-AJX3, In the ZK Ajax framework, there exists a file which is called Widget Package Descriptor (WPD) [32] with an aim to describe the information of a package, such as its widget classes and external JavaScript files. In ZK Ajax framework, the word "package" stands for the root element which denotes the package name and the language it belongs to, while the word "widget" stands for the widget class name without the package name, and if the package contains multiple widgets it lists them one by one. The following figure shows how to define a sample widget in ZK and its required elements which are needed to be used within the XML.

```
<<package name="zul.grid" language="xul/html">
<widget name="SimpleLabel"/>
</package>
```

Figure 3.24: Sample Widget Definition: SimpleLabel

By doing so, we completed the configuration the basic implementation of our component even though, it does not have any interactive events for now. In that respect, the next logical step will be to start adding events to the component that we created.

According to the defined requirement FR-WZB1, Wazaabi needs to provide more target models implementation and it should allow the designer to create new user interfaces by editing target models. We understand that any changes by the designer triggers a model change. Since Wazaabi provides currently basic underlying model (core model) and three inherited models; SWT, SWING and JSF which let creating instances build a model for GUI, it is known that for a new inherited model of target technology (Ajax in our case) a concrete implementation of the new framework in the core must be provided.

For the requirement FR-WZB2, we need to define only specific elements for the target Ajax technology however we need to consider that most of the user interface elements provided in the core model are already defined as abstract which means that they are platform dependent.

3.5.2 Non-Functional Requirements

Requirement	**Priority**	**Description**
NF-AJX1	High	Ajax support must be integrated into Wazaabi.
NF-AJX2	Medium	Ajax must be able to identify current changes.
NF-AJX3	Low	Ajax Security Issues must be properly defined.
NF-WZB1	High	Wazaabi must conform to the technical constraints put by Ajax.
NF-WZB2	High	Target model(Ajax) must be shown in Web Browser.
NF-WZB3	Medium	An editor which allows changes in the Wazaabi Viewer.

Table 3.2: Non-Functional Requirements

Regarding to the defined requirement NF-AJX1, a new rendering framework which is totally based on Ajax technique is needed to be integrated into the existing framework of Wazaabi project. In order to accomplish this, our strategy is to replace a new Ajax engine which is based on ZK Ajax framework with the existing SWT engine as our main contribution for the implementation. Therefore, we aim at, the existing work of Wazaabi project will be supported in web browser based user interfaces through using ZK Ajax framework and its relevant components. To sum up, coping with the web browser based user interfaces which is the main drawback of Wazaabi project will be solved as the Ajax support will be fully integrated.

According to the defined requirement NF-AJX2, as a matter of fact in the Ajax technique, JavaScript requests from the client part are sent to the server part which is responsible to interpret them in order to update the current page without directing the user to a new page. Thus, the Ajax technique is a web application technology which is independent of web server software so that it creates a more ubiquitous and responsive user interaction in the web like the traditional desktop based applications. Put in another way, the client part can still use the Ajax based web application as it requests the information from the server part which is working in the background.

For the defined requirement NF-AJX3, security is always an important factor of Ajax technology that must be taken into consideration. In other words, Ajax technique is the subject of a lot of hype, so that the issue of security is something which the developers must consider carefully. This new technology is popular because of its ability to create pages which are highly dynamic and interactive. It has also been popularized because of its ability to generate pages that do not need to be reloaded. However, it has also been the subject of controversy due to its vulnerability to hackers. While the truth of this is up for debate, the issue of security is something that should be discussed, both by developers and companies that are interested in using Ajax technique for their web applications or web pages. Additionally, it should first be noted that Ajax technique is not the single most important factor in determining whether or not a web page or a web application will be secure. However, we must have knowledge of what it is responsible for. Ajax technique is a collection of technologies that are closely related to web browsers.

In NF-WZB1, we need to understand the technical constraints of Ajax well in order to integrate with Wazaabi properly. Wazaabi concept takes the real user interface into consideration as a declared model. It is already known that this model is rendered by the engine and any changes are detected by the editor but we need to customize these modules to support Ajax model so that a new requirement for how we are able to adapt with the components of Wazaabi arises. This is obviously a precise requirement because it directly refers to our main task in this project. The implementation of our Ajax framework will be indicated briefly in the solution chapter.

The requirement for how to show target model in Wazaabi viewer (NF-WZB3) and what kind of components (for ex: a web browser) or plugins we need to implement Ajax framework into Wazaabi (NF-WZB2) has also a high priority for this project.

3.6 Chapter Summary

We started the Problem Analysis chapter by giving the general definition of the problem that we were going to deal with. Then, we explained our focus by relating it with the existing work of Wazaabi project and the Ajax technique respectively. We elaborated Wazaabi by explaining its important concepts which were EMF and SWT models as well as providing a sample Wazaabi scenario which includes a SWT engine implementation in order to give a better understanding about the relationship between these models. Therefore, we elaborated the intended functionalities of the existing work of Wazaabi project and explained how the model based support could benefit the user interface design at runtime of an application. After that, we continued our discussion with explanation of the Ajax support as well mentioning about our intention by using the Ajax technique in order to achieve our ultimate goal in this master thesis and also we kept in mind that, what kind of Ajax scenarios can be adapted. At that point, we also explained the differences between the existing model of the Wazaabi which lacks of the web browser based user interfaces and our approach: Proactive Ajax Approach briefly. Then, we provided four different Ajax scenarios by giving all possible cases and relevant screenshots of the running Ajax based applications in order to give realistic situations where and how the Ajax technique can be used. Not only the details of four scenarios were discussed but also we gave the sequence diagrams for each scenario to have a better look about how Ajax technique were involved as well as elaborating the Ajax logic behind the web applications that we selected to explain in a more detailed approach. Finally, we moved on to the requirement specification section and had a quick look of the requirements table which clearly shows the need of how the Ajax support could be implemented. While exploring the requirements, we also considered how Ajax framework could support user interface scenarios as a more comprehensive analysis by giving the possible functional and non-functional requirements.

Chapter 4

Proposed Solution

The purpose of this chapter is revealing possible solutions under the lights of group's knowledge about the problem situation. In other words, possible solutions are found as the results of deep analysis of theoretical concepts by the group. Especially, an already given scenario which we named "Proactive Ajax Approach" will be considered throughout this project as a desired situation. We have tried to fulfill both the objectives of the project and how they can be accomplished in a sensible manner.

Next chapter will present the implementation phase of the selected solution in more detail. Therefore, we will keep on this chapter by giving a solution overview, a new engine definition and we will wrap up the chapter by taking pros of all possible solutions together in overall goals section that will also define our work for the implementation.

4.1 Solution Overview

This section is reserved to provide a brief description of the possible solution for the given project. For designing and implementing Ajax support for Wazaabi plugin has been explained and new user interfaces based on the core engine of Wazaabi has also been presented for supporting Ajax framework. For creating new user interfaces, a new Ajax model which shows detailed information about the components has also been proposed. In order to implement the framework to the existing work Wazaabi smoothly and to avoid expressing unnecessary information, we tried to keep the model simple as much as we can.

As we discussed in the chapter 3: Problem Analysis, the possible issues about how an Ajax support can be adapted and what kind of Ajax user interface components can be supported in Wazaabi are explained. In addition, we focused on clarifying the requirements that are rised from possible Ajax scenarios, thus we will be able to establish goals for the implementation by also analyzing the core points of these possible situations in different aspects both from Wazaabi and Ajax. To consolidate our Ajax approach which is going to support our implementation as well, we need to develop a set of specific goals related to giving scenarios which its details will be mentioned in the end of the chapter 4.

4.2 Desired Scenario

In this section, the possible scenario where an expected case can occur is given. The following scenario analyzes Ajax case step by step and shows the Ajax interactions between client and server parts as sequence diagram. Additionally, we will elaborate how the Ajax engine can support the existing work of Wazaabi and how the web user interface components in the web browser are manipulated by the new engine. The figure 4.3 illustrates the following steps;

1. A user edits an element in the web user interface by using web browser.
2. DOM tree is modified according to the changed element in the web user interface. This update in DOM tree is also reflected in the EMF tree.
3. A request is sent to the server part and the Engine traverses the EMF tree to find changes in the EMF tree.

4. Then, EditPart notices any node changes in the EMF tree and makes sure the WidgetView is updated.

5. Once the WidgetView is updated, EditPart informs the Ajax engine that the EMF tree has been modified.

6. Then, the Ajax WidgetView updates the changed component in the web user interface according to changes in the EMF tree.

7. Consequently, the Wazaabi Viewer reloads the web user interface and the change is shown in web browser dynamically.

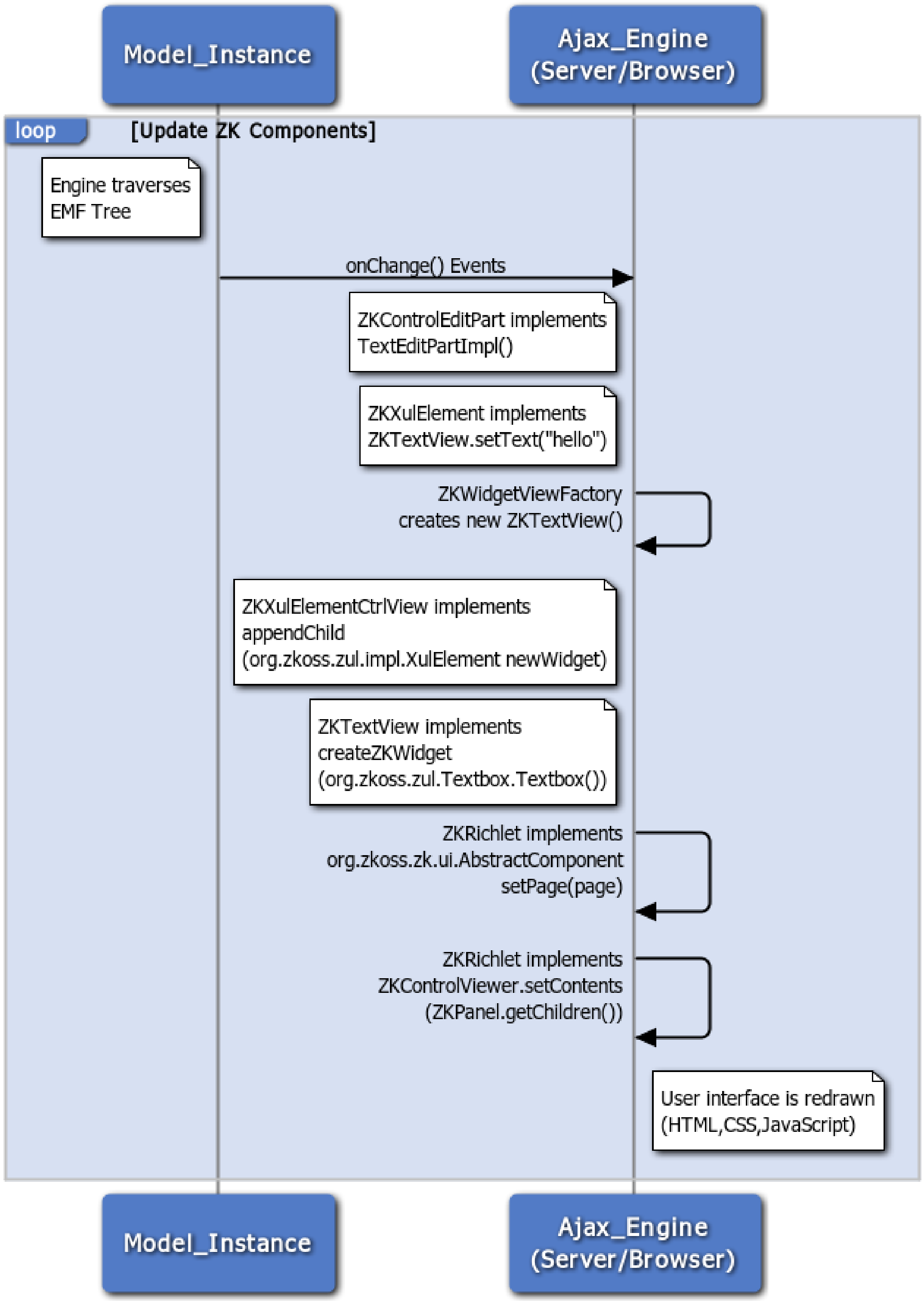

Model_Instance
Ajax_Engine (Server/Browser)
loop
[Update ZK Components]
Engine traverses EMF Tree
onChange() Events
ZKControlEditPart implements TextEditPartImpl()
ZKXulElement implements ZKTextView.setText("hello")
ZKWidgetViewFactory creates new ZKTextView()
ZKXulElementCtrlView implements appendChild (org.zkoss.zul.impl.XulElement newWidget)
ZKTextView implements createZKWidget (org.zkoss.zul.Textbox.Textbox())
ZKRichlet implements org.zkoss.zk.ui.AbstractComponent setPage(page)
ZKRichlet implements ZKControlViewer.setContents (ZKPanel.getChildren())
User interface is redrawn (HTML,CSS,JavaScript)
Model_Instance
Ajax_Engine (Server/Browser)

Figure 4.1: Sequence Diagram of Desired Scenario

According to the sequence diagram of desired scenario above, the following figure illustrates the hierarchical structure of ZK engine. In addition to the Wazaabi hierarchy 3.6that we mentioned previously in chapter 3, we introduce a ZK framework which is consisted of ZK Ajax components and a browser including DOM tree. As we said before, all nodes in the model instance are matched with the nodes of other parts in the newly implemented ZK engine.

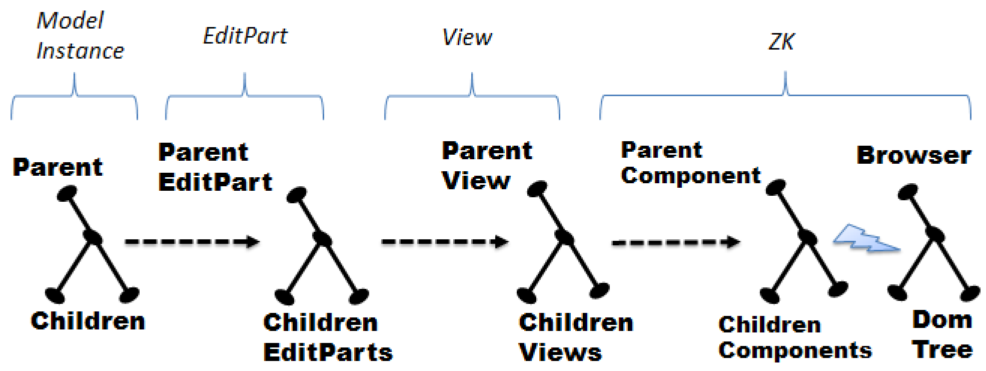


Figure 4.2: Hierarchical Structure of ZK Engine

4.3 Ajax Engine Design

The purpose of the new engine presented in this section is to give users an ability about how to create and edit user interfaces as well as their components based on the new Ajax engine. Since the new model we are going to create is based on EMF model, provides a way for us to associate components with each other in a tree hierarchy just like the previous existing SWT model. The point is that these components are gathered from a new resource containing Ajax libraries.

As we already mentioned before in the previous chapters, our main objective is to support the existing work of Wazaabi project by using an Ajax framework to build its dynamic web based application part, therefore making the Wazaabi capable of dealing with the web browser based user interfaces.

On the dynamic web based applications, the users can make requests of a database contained on a server part. Within an Ajax framework, data is sent to or read by the server part through the use of JavaScript. This allows users to make specific requests that are processed and implemented using dynamic factors like specific user input, time of day or search history to determine the content of the web page. While Ajax is primarily a client part set of technologies, the Ajax framework facilitates server part processing such as the finding and storage of data. The goal of the Ajax framework is to function as a communication layer between client (the web browser, the user) and server parts. This framework, which is specially designed to handle Ajax requests, lets Ajax perform its function of reducing user wait time while the web page accesses the server part.

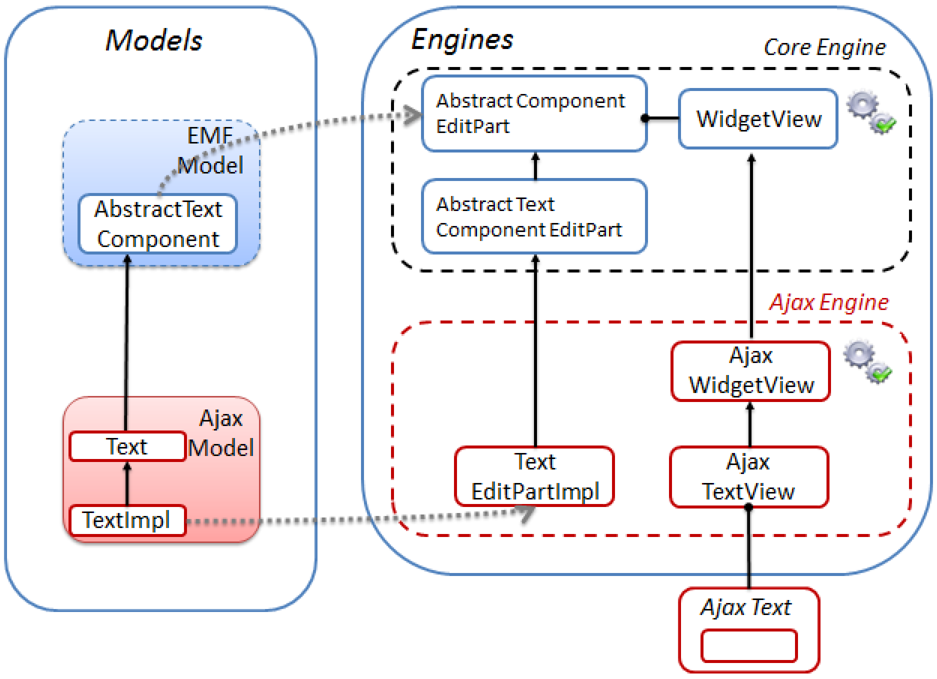


Figure 4.3: Ajax Scenario

We choose the server-driven approach for the Ajax framework which uses its components that are developed by the server part and using server part languages. The client part requests are communicated to the server part through Ajax techniques. Recall from the Wazaabi scenario in the Problem Analysis chapter, the existing work of Wazaabi project has an already existing engine which is based on SWT architecture in order to handle the interactions between its client and server parts. Now, we are going to give our own Ajax scenario which replaces the SWT based old engine with an Ajax based new engine in order to make the Wazaabi capable of handling with the web browser based user interfaces properly. Additionally, the interactions between the client and server parts in the Ajax scenarios was already given in the desired scenario of this chapter to make more sense for the readers.

4.4 Overall Goals for Implementation

These are the main goals which covers the core parts of our contribution. In addition, we are going to explain the important tasks which are emerged from specific requirements of the project that we already discussed. Generally, we concern about the implementation of how the Ajax components can be applied to the existing Wazaabi framework and with its abilities how the new system can fulfill the aforementioned requirements.

These goals are divided into three main following tasks and each one below represents an overview of what the future sections discuss in the next chapter.

Goal	**Description**
1	Binding Ajax support to Wazaabi framework.
2	Generate an Ajax engine inherited from Wazaabi core engine.
3	Integrating ZK Ajax components with Wazaabi framework.

Table 4.1: Overall Goals For Implementation

The Ajax support which we are going to explain, is planned for the work "Wazaabi framework" which is mentioned in chapter 2. The main purpose behind this support is to provide Ajax technology to Wazaabi framework by implementing it as a standard to be used easily. Relating to the Ajax support, one of the critical points is the typical Ajax architecture and how it should be implemented in this work. Generating an Ajax engine based on the core engine of Wazaabi is our main approach for the proposed solution. Moreover, our approach will handle the implementation by taking advantage of Wazaabi core engine as well as using an important Ajax framework called ZK. The field of ZK Ajax framework presents us a well structured Ajax architecture with various usable ZK components. In chapter 5, we will focus them with their properties separately. In our case, this Ajax framework is selected to be used for the implementation of Ajax components into Wazaabi framework. In addition, Ajax components will be defined in this new ZK Ajax engine by gathering already existing definitions from ZK Ajax libraries.

4.5 Chapter Summary

In this chapter, we started our discussion with the solution overview which gives a very brief description of the possible solution for the given project and how the new Ajax approach could be designed. Then, we continued our discussion with the desired scenario which simply depicts the desired case as in the Ajax engine integrated Wazaabi interactions. To make it more sense, we not only gave the step by step explanation of the desired case but also the sequence diagram of the desired case was provided as well. After that, we elaborated the main part of this chapter which was "Proactive Ajax Engine" based on our scenario which we previously named as "Proactive Ajax Approach" in the Problem Analysis chapter. Therefore, we depicted how using the new Ajax engine instead of the existing SWT engine can support the existing work of Wazaabi in handling with the web browser based user interfaces and how the web user interface components in the web browser are manipulated by the new Ajax engine by providing a figure of the Ajax scenario. Finally, we wanted to touch on the overall goals which were based on the analysis of the already defined requirements for the implementation chapter which will present the actual case for the selected solution in more detail.

Chapter 5

Implementation

In this chapter, we are going to discuss how the implementation phase can be adapted and the work was organized. A new approach which is planned to serve as a possible solution can handle the desired situations that are already explained in the scenarios. The aim is to fulfill most of the requirements in expected conditions by depicting various diagrams of the new integration. These diagrams are able to give the whole idea as a summary to the reader easily.

5.1 Implementation Overview

This section is reserved to provide a brief description of the implementation for the proposed solution which was explained in the previous chapter in more detail. As we already mentioned before, our ultimate goal in this master thesis is to support the existing work of Wazaabi project with Ajax technique in order to make it be able to handle with the web browser based user interfaces which is a major deficiency in its current version. Therefore, the existing engine of Wazaabi which is based on SWT will be replaced with an Ajax based engine to cope with this drawback. In order to do this, we are going to create our own Ajax engine by using the ZK Ajax framework and its relevant features. To integrate the new Ajax engine into the existing work of Wazaabi project properly as well as taking into consideration the requirements that are rised from Ajax, we will keep the model that we will deal with very simple as much as we can, and also focusing on the only relevant parts of the Wazaabi.

In the rest of this chapter, we are going to elaborate how the ZK Ajax framework can be used to support the existing work of Wazaabi project for the issue of web browser based user interfaces in more detail. Still, before we dig into the implementation of Ajax technique into the Wazaabi itself, we are going to provide information about the ZK Ajax framework, its architecture and components in more detail firstly, so that we will be able to relate them more easily with the Wazaabi in the rest of this chapter. And finally, to reinforce our Ajax implementation and the new approach for supporting Wazaabi with web browser based user interfaces, we will also mention about main development platforms, programming languages, software applications that we used to accomplish our main objectives in this master thesis one by one to make it more sense for the readers in the end of the chapter 5.

Before we dive into the architecture of the ZK framework, it is important to say that the aim of this introductory section of the whole implementation is reserved to provide an understanding of the basics and some important aspects of ZK framework which are essential for development with the Ajax framework.

5.2 ZK Ajax Framework & Components

Now, we are starting our discussion about the architecture of the ZK Ajax Framework briefly, then we will provide the information about the components of the ZK Ajax Framework by illustrating the system diagram.

ZK is an open source Ajax web application framework which is written in Java, and it enables creation of rich graphical user interfaces for web based applications without using JavaScript. ZK takes the so called server-driven approach that the content synchronization of components and the event pipelining between clients and servers are automatically done by the engine, and Ajax plumbing codes are completely transparent to web application developers[33].

The ZK framework is both component based and server-driven Ajax framework, which means it is driven through events while all processes are done at the server part. There are three essential components:

1. An Ajax based event-driven engine
2. A rich set of XUL and XHTML components
3. ZUML (ZK User Interface Markup Language)

The following figure shows the system diagram of the ZK Ajax Framework which depicts the general interactions between client and server parts in ZK.

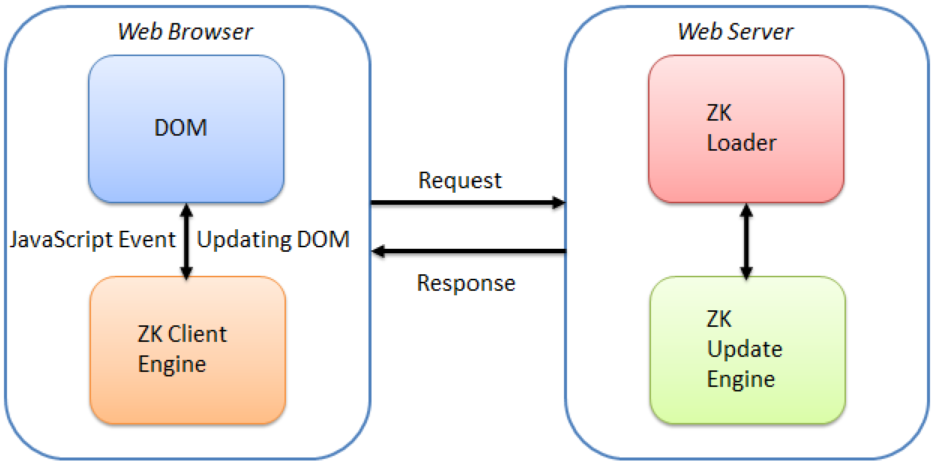


Figure 5.1: ZK Ajax Framework System Diagram

- XUL is the abbreviation for XML User Interface Markup Language. This "language" is not a new invention from the ZK team. It was originally defined by the Mozilla team [36]. The intention of Mozilla is to have a platform independent language to define user interfaces. And therefore, a ZK developer can benefit from the big community around XUL, and can use XUL widgets.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window
id="findfile-window"
 title="Find␣Files"
 orient="horizontal"
xmlns="http://www.mozilla.org/keymaster/gatekeeper
/there.is.only.xul">
<button id="find-button" label="Find"/>
<button id="cancel-button" label="Cancel"/>
</window>
```

Figure 5.2: Sample XML

- ZUML is the abbreviation for ZK User Interface Markup Language and it is based on XUL. There are some extensions to the XUL standard like the possibility of embedding program code. For the user, the application feels like a desktop based application. For the developer, the programming is like a standalone application. ZK Ajax framework uses an event-driven paradigm which encapsulates the request & response paradigm from a web based application. These technical infrastructure things are done through the Ajax framework. Therefore, no JavaScript usually needs to be written for the client part (the web browser). Here, it is important to say that the Ajax framework automatically includes some JavaScript libraries, and generates the JavaScript code for the user. Therefore, the developer has nothing to do with JavaScript.

The following figure show a sample event listener of the onChanging attribute for the textbox username in ZUML.

```
<?xml version="1.0" encoding="UTF-8"?>
<zk xmlns="http://www.zkoss.org/2005/zul">
<window title="My First window"
border="normal" width="400px">
<label value="Insert your name:" />
<textbox width="60%" id="username">
<!-- implement an event listener for the event onChanging -->
<attribute name="onChanging"><![CDATA[
if (event.getValue().length() > 0 && event.getValue()
.trim().length() > 0)
{
buttonToSayHello.setVisible(true);
}
else
{
buttonToSayHello.setVisible(false);
}
]]>
</attribute>
</textbox>
<button id="buttonToSayHello" label="Say Hello" visible="false">
<attribute name="onClick"><![CDATA[
alert('Hello '+username.value);
]]>
</attribute>
</button>
</window>
</zk>
```

Figure 5.3: onChanging Event Listener in ZUML

5.2.1 ZK Architecture

As we already mentioned before that the ZK Ajax framework uses the server-driven approach. Therefore we can say that, a ZK application runs at the server part. It could access the back-end resources, assemble user interface with its components, listen to the activity of the user, and then manipulate components to update the user interface. In order words, everything is done at the server part, because of being a server-driven Ajax framework. The synchronization of the states of the components between the web browser and the server part is done automatically by ZK Ajax framework in the background of the application.

While running at the server part, the application can access the full stack of Java technology. User activities are, including Ajax and Server Push, abstracted to event objects. User interfaces are composed by components which is one of the most productive approach to develop a modern web based application.

With the client-server fusion architecture of ZK Ajax framework, the application will not stop running at the server part. The application could enhance the interactivity by adding optional client part functionality, such as client part event handling, visual effect customizing, and even user interface composing without the server part code. ZK Ajax framework enables the seamless fusion ranging from pure server-driven to pure client-driven.

In order to understand the architecture of the ZK Ajax framework smoothly, we should mention about its three essential and internal components which makes it work properly. The main elements of the ZK Ajax framework are[34];

1. ZK Loader
2. ZK Client Engine
3. ZK AU (Asynchronous Update) Engine

ZK Loader

This component is responsible for the loading and interpretation of a ZK based web page. The selection of the web page is done based on the incoming URL request from the client part (the web browser). The result is rendered as an HTML based web page and sent back to the the client part (the web browser).

ZK Client Engine

This component sends “ZK Requests” to the server part, and receives "ZK Responses" from the server part. With these responses, the DOM (Document Object Model) tree on the client part is updated. Therefore, we could call it as the client part of the Ajax.

ZK AU (Asynchronous Update) Engine

The ZK AU Engine simply constitutes the server part of the Ajax. The following figure shows the interactions between these three essential components of the ZK Ajax framework.

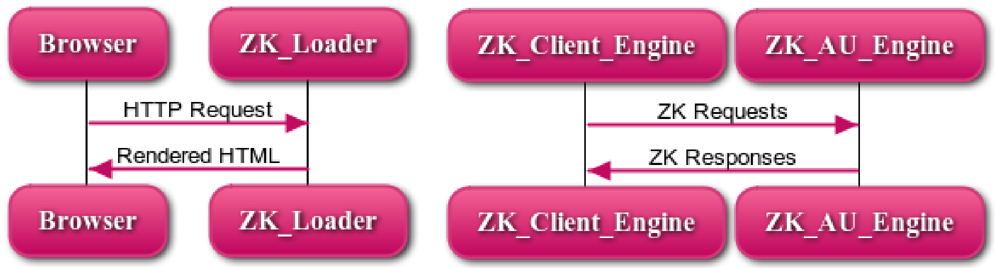


Figure 5.4: Interactions of the ZK Archtitecture

As we can see from the figure above, the sending of ZK Requests and ZK Responses, and the resulting updates constitute the mentioned event driven programming model. On the user side, there are only HTML pages. By using ZK, these pages are build on desktops, pages, and components[35].

- **Desktop (Framework: org.zkoss.zk.ui.Desktop)** It is possible for a ZUML page to include another ZUML page. This is because, these pages are available under the same URL and they are grouped into a desktop. Therefore, a desktop is a container for the pages.

- **Page (Framework: org.zkoss.zk.ui.Page)** It contains the components. Therefore, a page is a container for the components. If the ZK loader is interpreting a ZUML page, a org.zkoss.zk.ui.Page is created.

- **Component (Framework: org.zkoss.zk.ui.Component)** A component is a user interface object such as button, textbox, or label. Besides being a Java object on the server, the component has a visual representation in the web browser. This is created at the moment of attachment to a page. At the moment of detachment, the visual part is removed. A component can have children. Otherwise, a component without any parent is called the root component.

The following figure shows the architecture of ZK framework as well as the main interactions between the web browser (client) side and the server side.

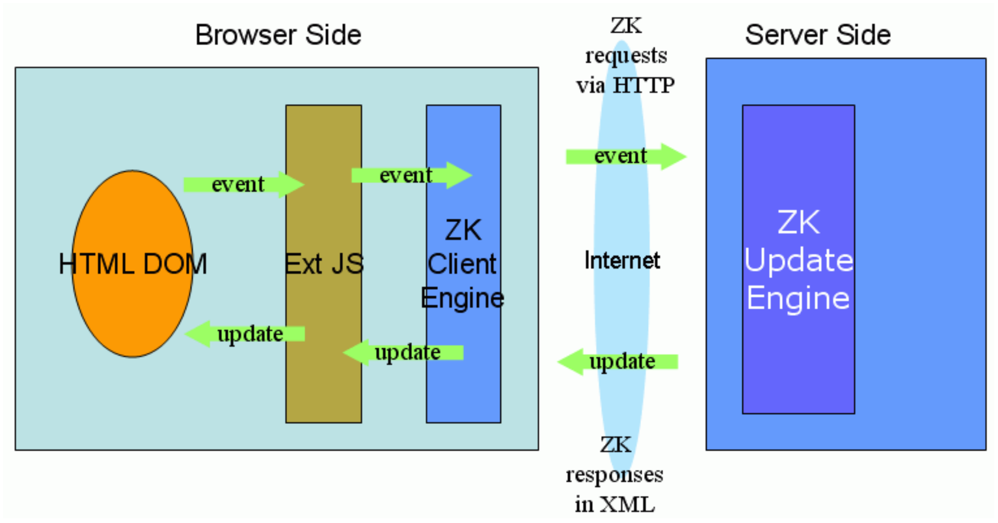


Figure 5.5: ZK Framework Architecture[37]

Since the ZK Ajax framework utilizes the server-driven logic, the ZK Ajax framework based application runs at the server part. Put in another way, everything is done at the server part by the ZK Update Engine. The synchronization of the ZK requests via HTTP and ZK responses in XML between the web browser and the server part is done automatically by ZK Ajax framework in the background as sending events and updates.

Furthermore, the following figure shows real interactions between the web browser (client) part and the server part in great detail, while a ZK Ajax framework based application is generating an HTML page. In the following figure, ZK Layout Engine is a representative of the ZK Update Engine which resides in the server part;

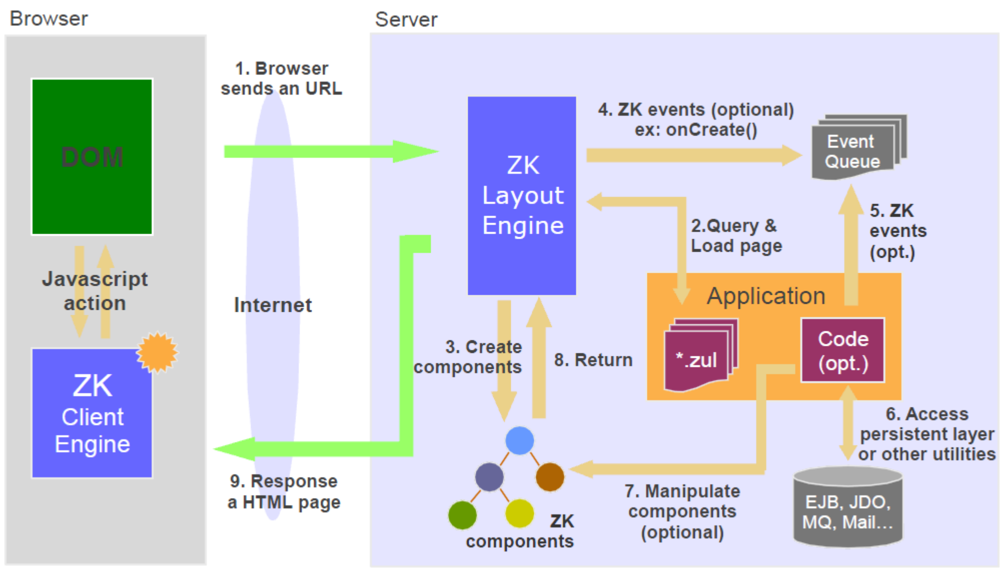


Figure 5.6: ZK Generating an HTML Page[35]

5.2.2 ZK Components

Every user interface object in ZK Ajax framework is represented with a component and a widget[33]. Consequently, building the user interface means compiling the whole components. In order to change the user interface, we need to modify states of the components and relationship between them. In other words, a component is inherently a user interface object, for example, it can be a window component or a button component. Moreover, it represents a graphical view and shows behaviours of a specific user interface object. A component can be a Java object which is running at server side or it can be created in the web browser when it is attached to the page. On the other hand, a widget which runs at the client side is a JavaScript object. It allows interaction with the user by representing the user interface object and manipulating events at the client. These two objects work together to provide a fully interactive user interface experience to the user.

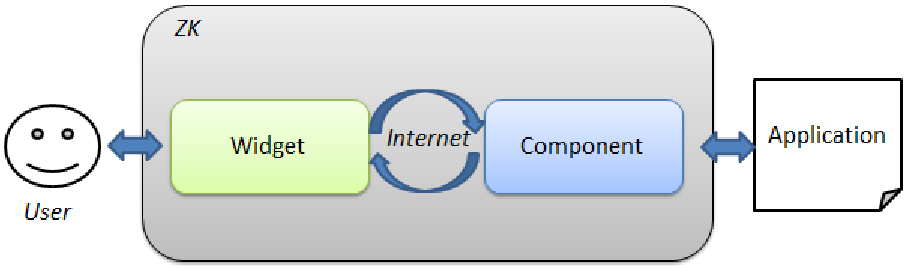


Figure 5.7: Relationship between Components & Widgets

In the figure above, the widget detects the events and sends a corresponding request to the component. Then, the component communicates with the application and sends a response back to the widget to update. When the attribute of a component is changed, then the widget of the related component is triggered at the client side and its graphics are automatically changed[38].

The figures below give a better understanding about how the ZK components work and show the notification between the server and the client when the button widget is clicked at the client side. An event called onClick is invoked and notifies the application.

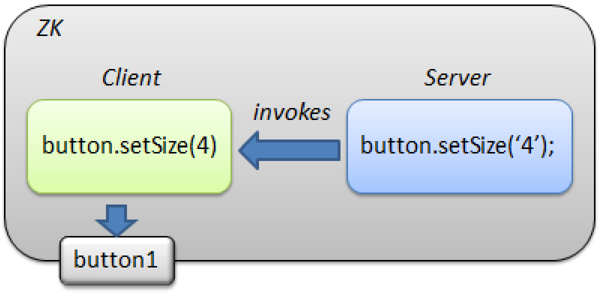


Figure 5.8: Invoking setSize() Method

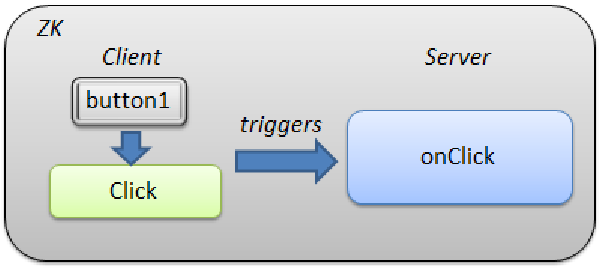


Figure 5.9: Triggering onClick Event

Every ZK components are obtained from org.zkoss.zk.ui.Component. It is possible to manage graphical interface of an application at client side by editing these components or changing their properties. Pages which come from the class of org.zkoss.zk.ui.Page, comprise components that are distributed in a web browser. When the ZK loader interprets a ZUML page, a page is created and components are added or removed in that page. The collection of pages compose a desktop during providing the same URL request. While we are using the application, pages can be added to the desktop or removed.

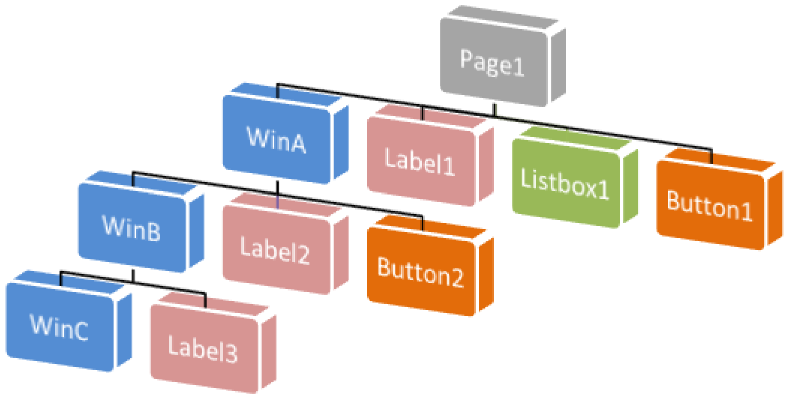


Figure 5.10: Relationship between components

The figure above illustrates an example of how the component are structured in ID spaces which are composed of a space owner and their children. In this case, the window A is the owner of the ID space and there are three window components on this Page1. Both of the ID spaces contains label and button component. In some cases, the same component can be located in two different ID spaces. If we want to get a component from one ID space to another, the getFellow() method should be used. For example;

```
getFellow("winB").getFellow("label3")
```

Figure 5.11: getFellow() Method

Here is the structure of various components; As we see in the figure below, page component is the another space owner. Actually, there are three ID spaces; two window components: winA, winB and a page component, Page1.

```
<?page id="page1"?>
<zk>
  <window id="winA">
     <window id="winB">
       <window id="winC">
       </window>
       <label id="label3" value="Label␣3"/>
     </window>
     <label id="label2" value="Label␣2"/>
     <button id="button2">
  </window>
  <label id="label1" value="Label␣1"/>
  <listbox id="listbox1">
  <button id="button1">
</zk>
```

Figure 5.12: ID Space Example

Class Hierarchy of Components

As the hierarchy of components is seen on the Figure 5.13, components are derived from AbstractComponent class. As you can see, different classes provide different functionalities. For example, A Java class called XulElement is used to provide elements including various methods for attributes of the components such as get and set. HtmlBasedComponent Java class is used for defining some attributes of HTML based components such as width, height and style.

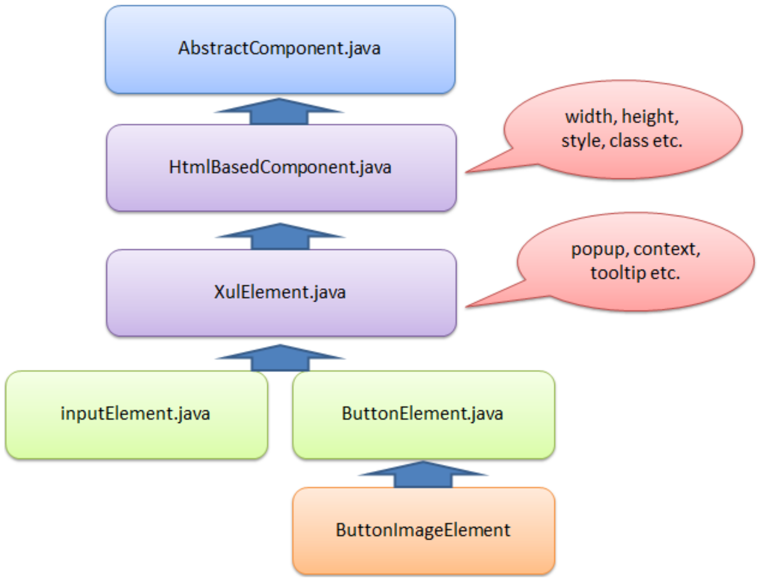


Figure 5.13: ZK Component Class Hierarchy

We already mentioned that a user interface object is composed of a server side component written in Java and a client side widget written in Javascript. Moreover it is also known that the Java class of the component is derived from AbstractComponent and several classes in the Figure 5.13. Now, we are going to talk about how a simple component is implemented and how user interfaces are created.

Implementing the Components

In order to implement a component, a specific class name should be identified and the attributes of the components should be decided. An example for the component attribute is shown in the Figure 5.14.

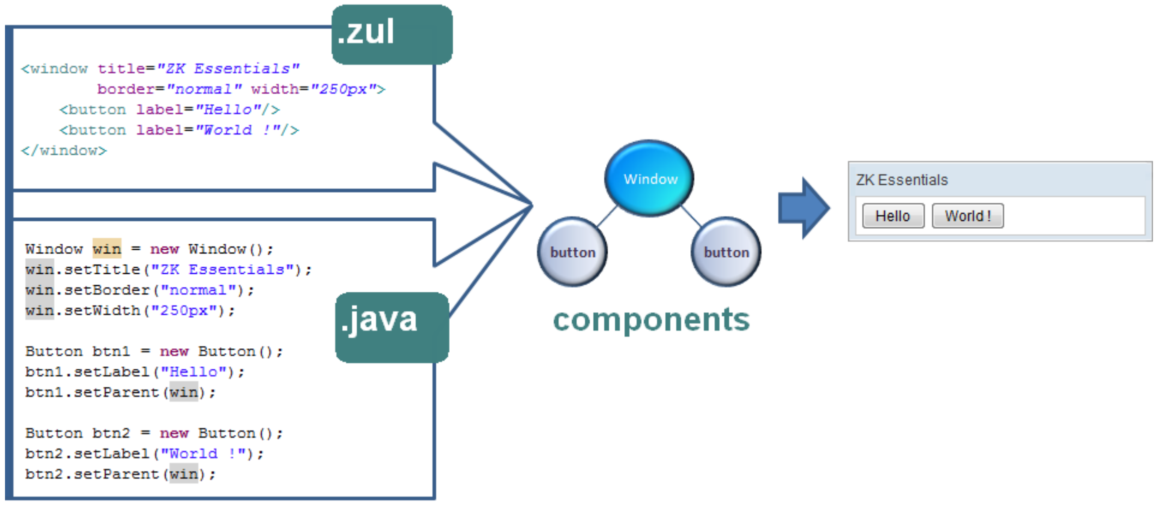
```
package org.implement;
public class SimpleLabel extends
org.zkoss.zk.ui.HtmlBasedComponent
{
  private String _value = ""; // a data member
  public String getValue() {
  return _value;
  }
  public void setValue(String value){
    if (!_value.equals(value))
    { _value = value;
      smartUpdate("value",_value);
    }
  }
  protected void renderProperties
 (org.zkoss.zk.ui.sys.ContentRenderer renderer)
  throws java.io.IOException
  { super.renderProperties(renderer);
    render(renderer, "value", _value);
  }
}
```

Figure 5.14: Implementing a ZK Component

In the code above, there are get and set methods for getting and setting the value of the attribute called setValue() and getValue(). The setValue() method informs the client by using the function smartUpdate() from AbstractComponent class. It has two parameters; String for the name of the attribute and object for the value. This function updates values of specific attributes of the associated widget at client side so the DOM tree can be changed by the client. The renderProperties() method is used to render all required attributes which are sent to client for creating the widget when the associated component is attached to the page. To render all attributes, super.renderProperties should be called at first before using the render method.

Composing User Interfaces in ZK

There are two approaches to compose user interface in ZK Ajax framework; one is normal Java approach by defining user interface in pure Java[39] and the other one is XML-based approach. As we already mentioned, the declaration language which is ZUML (ZK User Interface Markup Language) is based on XML. Basically, the ZK Loader is informed by the XML instructions about what kind of components are supposed to create and how their properties are defined. For example, what should be written for the text attribute of the button is decided.



```
<window title="ZK Essentials"
        border="normal" width="250px">
    <button label="Hello"/>
    <button label="World !"/>
</window>
```

```
Window win = new Window();
win.setTitle("ZK Essentials");
win.setBorder("normal");
win.setWidth("250px");

Button btn1 = new Button();
btn1.setLabel("Hello");
btn1.setParent(win);

Button btn2 = new Button();
btn2.setLabel("World !");
btn2.setParent(win);
```

Figure 5.15: Ways of composing UI in ZK[40]

Now, we are going to explain the life cycle of a ZUML page, how ZK components are created in the mechanism of ZK Ajax framework.

The life cycle of a ZUML page;

1. Page initialization : In this first phase, the processing instructions are executed by ZK and the initialization of the page is started.

2. Component creation : ZK Loader interprets ZUML page. When the components are created in the page, for each element, their attributes are specified in ZUML page one by one. The ZK loader processed any

elements and the procedure are repeated to initialize the content of additional elements.

3. Event processing : ZK calls listener for each event which is waiting for the desktop.
4. Rendering : ZK embeds the components into a HTML page and sends it to the browser.

Updating Pages

There are three steps for the ZK AU engine at server side to receive requests from client side;

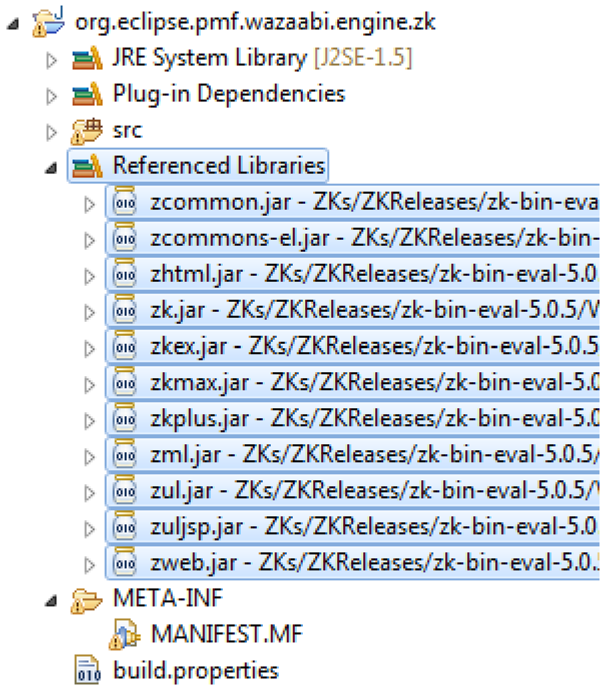
1. Request processing : Requests are stored in queues by ZK AU engine. For the same desktop, they are processed sequentially. ZK AU engine updates the content of related components.
2. Event processing : Since related events are posted in the queue, they are also processed sequentially for each unit.
3. Rendering : ZK renders the updated components and send specific responses back to the client side.

After these three steps, the ZK engine at client side updates the DOM tree in the web browser according to type of the responses. This update can recreate the whole component or just change one of its attributes.

5.3 Binding Ajax Support to Wazaabi

This section is one of the most important sections in the entire thesis because it elaborates the core idea of how we are going to implement our selected Ajax framework (ZK) to Wazaabi, the framework which we studied on previous chapters.

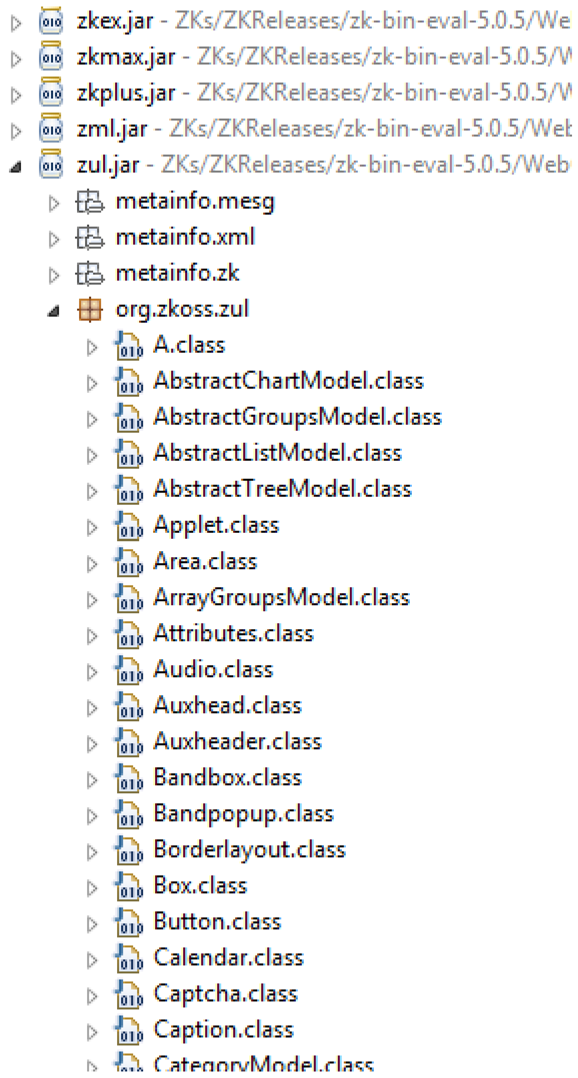
Since there are several advantages of using ZK Ajax framework such as providing detailed specification of declarative rich user interfaces, ZK Ajax framework also includes support for embedding various scripts which are crucial for user interface and back-end programming. Moreover, this Ajax framework is feasible enough for integrating with existing web frameworks such as Hibernate and Spring via a Servlet filter, JSF support and a JSP tag library. During the implementation, we should also pay attention to the server side logic. This is where asynchronous update comes forward. After all, ZK Ajax framework lets us deal with these updates at the server side by abstracting them via its components.



```
org.eclipse.pmf.wazaabi.engine.zk
    JRE System Library [J2SE-1.5]
    Plug-in Dependencies
    src
    Referenced Libraries
        zcommon.jar - ZKs/ZKReleases/zk-bin-eva
        zcommons-el.jar - ZKs/ZKReleases/zk-bin-
        zhtml.jar - ZKs/ZKReleases/zk-bin-eval-5.0
        zk.jar - ZKs/ZKReleases/zk-bin-eval-5.0.5/V
        zkex.jar - ZKs/ZKReleases/zk-bin-eval-5.0.5
        zkmax.jar - ZKs/ZKReleases/zk-bin-eval-5.0
        zkplus.jar - ZKs/ZKReleases/zk-bin-eval-5.0
        zml.jar - ZKs/ZKReleases/zk-bin-eval-5.0.5/
        zul.jar - ZKs/ZKReleases/zk-bin-eval-5.0.5/V
        zuljsp.jar - ZKs/ZKReleases/zk-bin-eval-5.0
        zweb.jar - ZKs/ZKReleases/zk-bin-eval-5.0.
    META-INF
        MANIFEST.MF
    build.properties
```

Figure 5.16: Importing ZK Libraries

As it is seen in the figure above, it is easy to integrate ZK with other frameworks like Wazaabi. ZK provides a web support in the view layer of Wazaabi. Embedding a component is important because importing ZK libraries and specifying Java classes to embed ZK component as a native component help us to deal with using these native elements without knowing the existence of ZK. The following figure illustrates the existing native ZK components in org.zkoss.zul package;



```
▷ zkex.jar - ZKs/ZKReleases/zk-bin-eval-5.0.5/We
▷ zkmax.jar - ZKs/ZKReleases/zk-bin-eval-5.0.5/V
▷ zkplus.jar - ZKs/ZKReleases/zk-bin-eval-5.0.5/V
▷ zml.jar - ZKs/ZKReleases/zk-bin-eval-5.0.5/Wel
◢ zul.jar - ZKs/ZKReleases/zk-bin-eval-5.0.5/Web
    ▷ metainfo.mesg
    ▷ metainfo.xml
    ▷ metainfo.zk
    ◢ org.zkoss.zul
        ▷ A.class
        ▷ AbstractChartModel.class
        ▷ AbstractGroupsModel.class
        ▷ AbstractListModel.class
        ▷ AbstractTreeModel.class
        ▷ Applet.class
        ▷ Area.class
        ▷ ArrayGroupsModel.class
        ▷ Attributes.class
        ▷ Audio.class
        ▷ Auxhead.class
        ▷ Auxheader.class
        ▷ Bandbox.class
        ▷ Bandpopup.class
        ▷ Borderlayout.class
        ▷ Box.class
        ▷ Button.class
        ▷ Calendar.class
        ▷ Captcha.class
        ▷ Caption.class
        ▷ CategoryModel.class
```

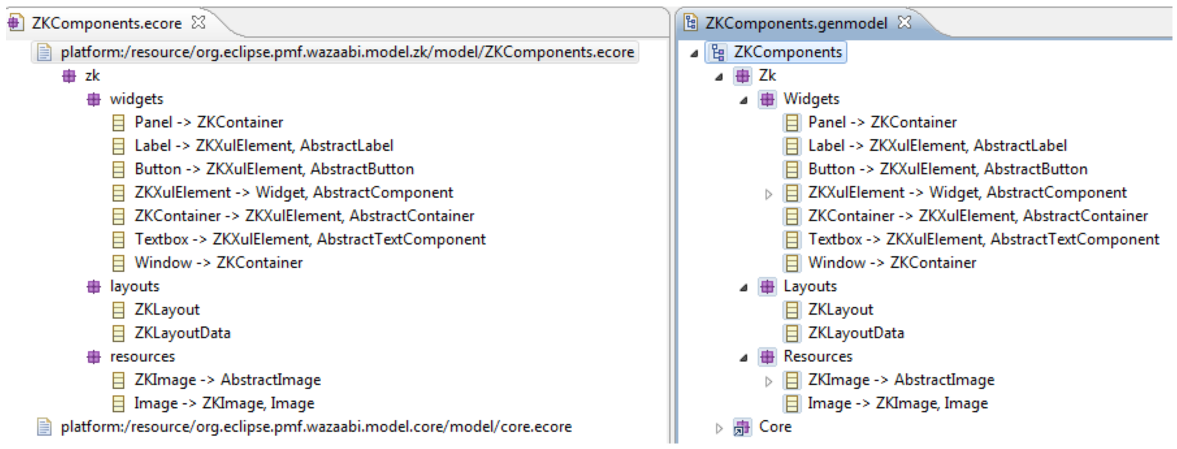
Figure 5.17: Package org.zkoss.zul containing Components

After importing the referenced libraries that are needed for ZK Ajax framework, it is possible to move on to the next level which shows the way of implementation for this possible integration between these frameworks.

5.4 Integrating ZK Framework with Wazaabi

This section covers the design and specifications of the new engine in order to give a better understanding about how we are going to support different Ajax components for rendering.

Firstly, we need to create a specific Ajax model based on EMF ecore for the selected Ajax framework (ZK) including its components. Secondly, we need to create a genmodel so that we can generate the code of the ecore model itself. As we said before, all elements of the user interface are binded to a specific model instance which is the subset of and inherited from the core model of Wazaabi. This new ecore model consists of different packages containing layouts, resources, and widgets like Textbox or Button which is an Abstract component controlled by ZK control class named ZKXulElement.



```
ZKComponents.ecore
  platform:/resource/org.eclipse.pmf.wazaabi.model.zk/model/ZKComponents.ecore
    zk
      widgets
        Panel -> ZKContainer
        Label -> ZKXulElement, AbstractLabel
        Button -> ZKXulElement, AbstractButton
        ZKXulElement -> Widget, AbstractComponent
        ZKContainer -> ZKXulElement, AbstractContainer
        Textbox -> ZKXulElement, AbstractTextComponent
        Window -> ZKContainer
      layouts
        ZKLayout
        ZKLayoutData
      resources
        ZKImage -> AbstractImage
        Image -> ZKImage, Image
  platform:/resource/org.eclipse.pmf.wazaabi.model.core/model/core.ecore
```

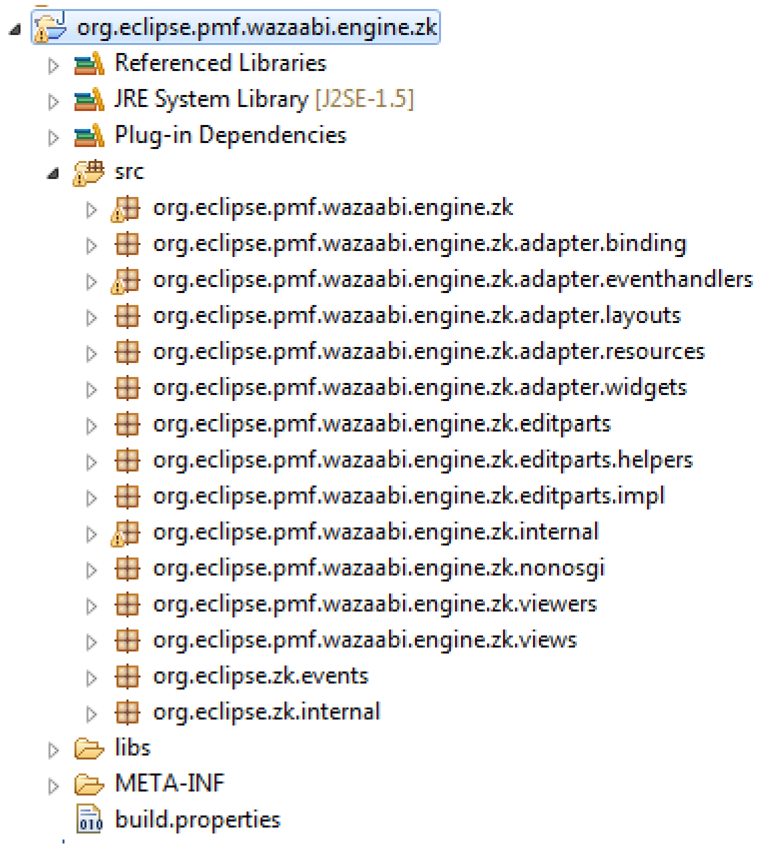
```
ZKComponents.genmodel
  ZKComponents
    Zk
      Widgets
        Panel -> ZKContainer
        Label -> ZKXulElement, AbstractLabel
        Button -> ZKXulElement, AbstractButton
        ZKXulElement -> Widget, AbstractComponent
        ZKContainer -> ZKXulElement, AbstractContainer
        Textbox -> ZKXulElement, AbstractTextComponent
        Window -> ZKContainer
      Layouts
        ZKLayout
        ZKLayoutData
      Resources
        ZKImage -> AbstractImage
        Image -> ZKImage, Image
    Core
```

Figure 5.18: Ecore Model of ZK Ajax Support

In widgets package, the definitions of the widgets are inherited from abstract ones which are located in the core model. The resources and layouts package are created for image and layout elements, whereas the ZKLayout is formed to locate the widgets. In addition, Panel element of ZK Ajax framework is integrated into the ecore model as a container called ZKContainer which is also inherited from AbstractContainer to wrap other elements in its structure.

5.4.1 Ajax Engine Implementation

The ZK Ajax engine is built same as the Ajax model is. Moreover, it is built for rendering the model instance in order to create the real user interface. Mainly, the project "org.eclipse.pmf.wazaabi.engine.zk" contains the ZK Ajax engine's behaviour for user interfaces based on Ajax and contains the real implementation of the Editparts for the target Ajax technology (ZK). Figure5.19 illustrates the general configuration and packages that are needed for the new ZK Ajax engine project.



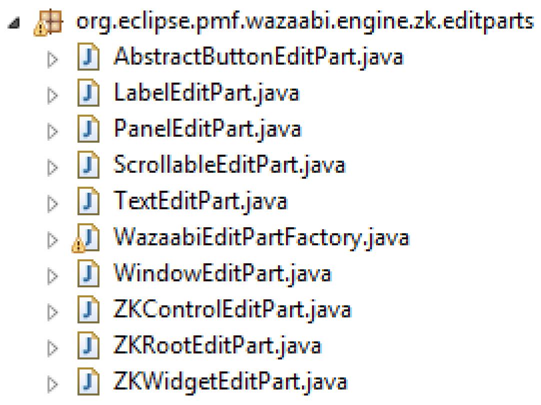
```
org.eclipse.pmf.wazaabi.engine.zk
    Referenced Libraries
    JRE System Library [J2SE-1.5]
    Plug-in Dependencies
    src
        org.eclipse.pmf.wazaabi.engine.zk
        org.eclipse.pmf.wazaabi.engine.zk.adapter.binding
        org.eclipse.pmf.wazaabi.engine.zk.adapter.eventhandlers
        org.eclipse.pmf.wazaabi.engine.zk.adapter.layouts
        org.eclipse.pmf.wazaabi.engine.zk.adapter.resources
        org.eclipse.pmf.wazaabi.engine.zk.adapter.widgets
        org.eclipse.pmf.wazaabi.engine.zk.editparts
        org.eclipse.pmf.wazaabi.engine.zk.editparts.helpers
        org.eclipse.pmf.wazaabi.engine.zk.editparts.impl
        org.eclipse.pmf.wazaabi.engine.zk.internal
        org.eclipse.pmf.wazaabi.engine.zk.nonosgi
        org.eclipse.pmf.wazaabi.engine.zk.viewers
        org.eclipse.pmf.wazaabi.engine.zk.views
        org.eclipse.zk.events
        org.eclipse.zk.internal
    libs
    META-INF
    build.properties
```

Figure 5.19: Packages of ZK Engine

In the figure above, we can declare that the most crucial packages in order of importance for the engine are; the editparts to make the engine awake for changes in ZK components as well as creating new ones, the editpart helpers to make the view updated according to these changes, the views for updating the real user interface according to changes in the existing model, and lastly, the adapters including event handlers to bind them all changing behaviours properly. The detailed code information is attached to the AppendixA.

EditParts

All the editparts for each elements that constitute interfaces, are inherited from one control editpart called ZKControlEditPart which extends to ZK-WidgetEditPart. Moreover, that interface also extends to WidgetEditPart which is used in the core engine for EditPart creations. ZKWidgetEditPart has an abstract method which calls the EditPart helper.



```
org.eclipse.pmf.wazaabi.engine.zk.editparts
    AbstractButtonEditPart.java
    LabelEditPart.java
    PanelEditPart.java
    ScrollableEditPart.java
    TextEditPart.java
    WazaabiEditPartFactory.java
    WindowEditPart.java
    ZKControlEditPart.java
    ZKRootEditPart.java
    ZKWidgetEditPart.java
```

Figure 5.20: ZK Engine EditParts

WazaabiEditPartFactory creates editparts for the specific elements according to the model instance by mapping and stores the model element in the related editpart. If the element is matched with the model element of instance, then it automatically returns a new LabelEditPart.

```
if (modelElement instanceof
     org.eclipse.pmf.wazaabi.model.zk.widgets.Label){
 return new LabelEditPartImpl();
}
```

Figure 5.21: Comparing the Model Element

EditPart helpers are supportive classes to get the model elements and their styles from the model. They are controlled by the ZKControlEditPartHelper. For example, LabelEditPartHelper gets the element and returns it into ZK Label component type. ZKControlEditPartHelper controls the editparts by checking the widget model and notifies any node changes immediately. It uses hooking and unhooking widget methods from its super class called ZK-WidgetEditPartHelper.

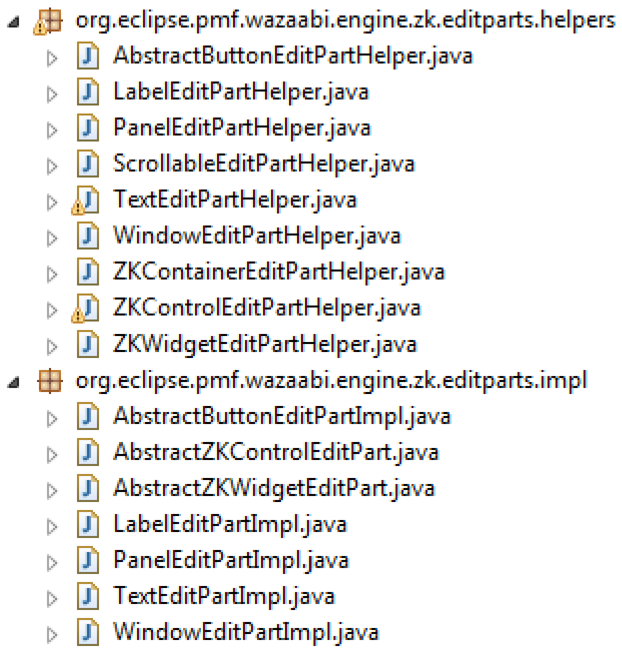

```
org.eclipse.pmf.wazaabi.engine.zk.editparts.helpers
    AbstractButtonEditPartHelper.java
    LabelEditPartHelper.java
    PanelEditPartHelper.java
    ScrollableEditPartHelper.java
    TextEditPartHelper.java
    WindowEditPartHelper.java
    ZKContainerEditPartHelper.java
    ZKControlEditPartHelper.java
    ZKWidgetEditPartHelper.java
org.eclipse.pmf.wazaabi.engine.zk.editparts.impl
    AbstractButtonEditPartImpl.java
    AbstractZKControlEditPart.java
    AbstractZKWidgetEditPart.java
    LabelEditPartImpl.java
    PanelEditPartImpl.java
    TextEditPartImpl.java
    WindowEditPartImpl.java
```

Figure 5.22: ZK Engine EditParts.Helpers and EditParts.Impl

These three abstract classes in "org.eclipse.pmf.wazaabi.engine.zk.editparts.impl" have various declared methods like hook(), unhook() widgets, notifychanged() which are used by their derived classes in editparts package. For example, AbstractZKWidgetEditPart has methods to notify the new elements in the model, gets the ZK widgets and hooks them into the appropriate view. The following code group shows the widget hook process to the view;

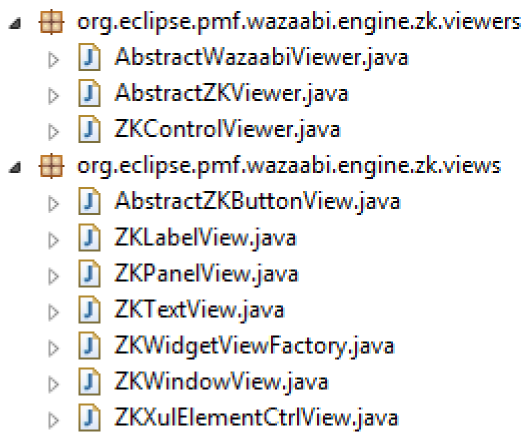
```
public void hookWidgetView(WidgetView widgetView){
 super.hookWidgetView(widgetView);
 getEditPartHelper().hookZKWidget(
        ((ZKXulElementCtrlView) widgetView).getZKWidget());
}
```

Figure 5.23: Hooking widget to View

Views

The views package of ZK Ajax engine is where we implemented our main methods that are being used for creating the views of selected ZK components which are Window, Panel, AbstractButton, Label, and Textbox. Moreover, the methods that we created in each Java class of ZK components are also aimed to deal with editing such as setting and getting the text attributes of the selected ZK components. In order to adhere the existing code base of Wazaabi views, we wanted to keep the methods of each component seperately. Additionally, the five selected ZK components intensively correlate with the XulElement control view which is our own super class of all ZK components according to the type hierarchy of ZK Ajax framework. Moreover, the XulElement control view is itself dependent on the widget view of the core engine from Wazaabi as well as AbstractButton, Label, and Textbox are dependend on AbstractButton view, Label view, and Text view of the core engine from Wazaabi respectively.

The main functionality of XulElementCtrlView is to make sure that a new child (ZK widget) is binded to its corresponding parent in its own tree typed widget hierarchy according to the ZK ecore model. As we already mentioned before, all ZK components of the user interface are binded to a specific model instance which is inherited from the core model of Wazaabi. Another important task of the XulElementCtrlView is to create and get ZK widgets in the type of AbstractComponent which is also the super class of XulElement with regard to the type hierarchy of the components in ZK Ajax framework. Finally, the WidgetViewFactory which is also intensively dependent on the view part of the Wazaabi core engine and that is an another crucial part in views package of ZK Ajax engine. Its ultimate functionality is to create instances of new widget views for each five selected ZK components according to their separate editparts. The packages of ZK views and viewers can be seen in the following figure;



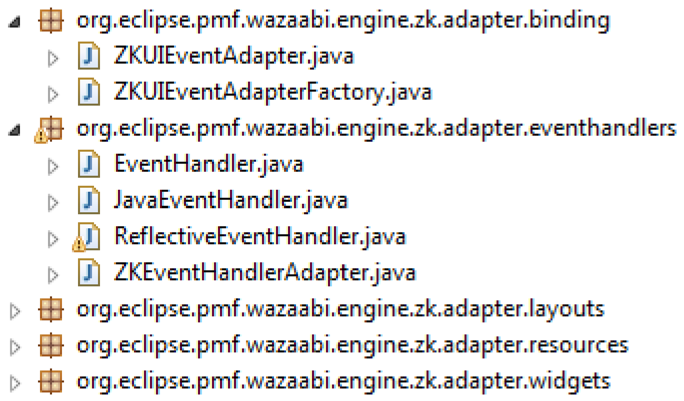
```
org.eclipse.pmf.wazaabi.engine.zk.viewers
    AbstractWazaabiViewer.java
    AbstractZKViewer.java
    ZKControlViewer.java
org.eclipse.pmf.wazaabi.engine.zk.views
    AbstractZKButtonView.java
    ZKLabelView.java
    ZKPanelView.java
    ZKTextView.java
    ZKWidgetViewFactory.java
    ZKWindowView.java
    ZKXulElementCtrlView.java
```

Figure 5.24: Views and Viewers

In the viewers package of ZK Ajax engine, we implemented our methods which are concerning the parents of each child widgets such as getting the parent of a specific widget in the type of AbstractComponent which is the main super class of all ZK components according to the type hierarchy of ZK Ajax framework. Besides the main functionality of AbstractZKViewer, the main functionality of AbstractWazaabiViewer is to initialize the EditPartFactory which is intensively dependent on the WazaabiEditPartFactory from the Wazaabi core engine. Lastly, the ZKControlViewer which has an important role in the viewers package of ZK Ajax engine, aims to deal with setting the RootEditPart for the main parent of all ZK widgets which is defined as a window in the AbstractComponent type as well as setting the contents of the window. Ultimately, to provide a better understanding of how the viewers package of ZK Ajax engine are structured and coded, its all related source codes can be found in the appendix A which is located in the end of the entire report.

Adapters

The adapters package of ZK Ajax engine has five main subpackages which are in turn binding, eventhandlers, layouts, resources, and widgets. Because of we have focused on only bindings and eventhandlers in the integration of ZK Ajax framework with the existing code base of Wazaabi project, we are going to examine the first two subpackages of the adapters. However, the entire view of all adapter packages of ZK Ajax engine can be seen in the following figure.



```
org.eclipse.pmf.wazaabi.engine.zk.adapter.binding
    ZKUIEventAdapter.java
    ZKUIEventAdapterFactory.java
org.eclipse.pmf.wazaabi.engine.zk.adapter.eventhandlers
    EventHandler.java
    JavaEventHandler.java
    ReflectiveEventHandler.java
    ZKEventHandlerAdapter.java
org.eclipse.pmf.wazaabi.engine.zk.adapter.layouts
org.eclipse.pmf.wazaabi.engine.zk.adapter.resources
org.eclipse.pmf.wazaabi.engine.zk.adapter.widgets
```

Figure 5.25: Adapter.Binding and Adapter.Eventhandlers

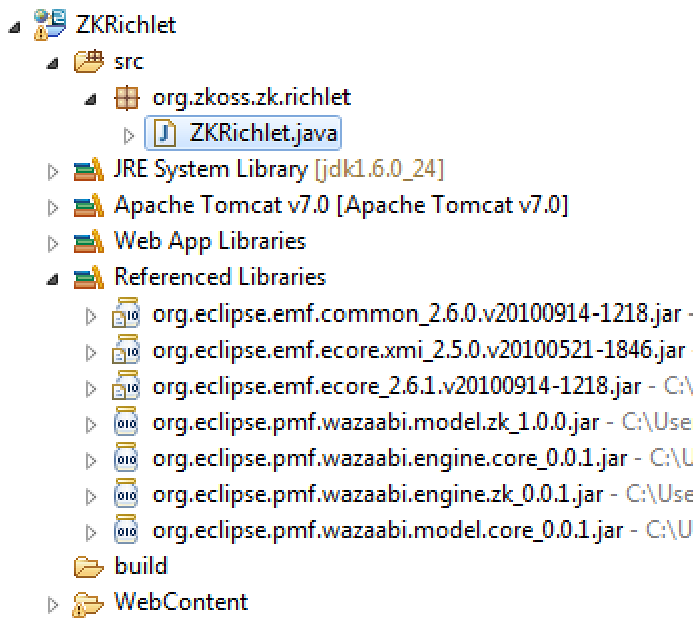
In the binding subpackage of adapters, we have focused on user interface event adapter which is dependent on the abstract user interface event adapter from the Wazaabi core engine, as well as user interface event adapter factory which is dependent on the platform specific component factory from the Wazaabi core engine. The main functionality of user interface event adapter is to provide getControl() function which returns the specific widget edit part in the type of desktop from the user interface library of ZK Ajax framework. Besides, it also deals with hooking and unhooking the current widget, with regard to the current event type. Apart from this, the factory is responsible for creating a new user interface event adapter.

In the eventhandlers subpackage of adapters, the ZK event handler adapter has not only the same methods with user interface event adapter from the binding subpackage, but also it has a NotifyChanged() method which has a vital role in the whole implementation. As we already explained in the sequence diagram of Wazaabi, the NotifyChanged() method uses both the EMF libraries of Eclipse and the existing Wazaabi core model. Additionally, it has a role to inform relevant edit parts in case of an alteration in the EMF trees of Wazaabi components such as adding a new element, or deleting an existing element, or updating an attribute of an existing element. In our implementation, they are replaced with the ZK components for adhering our ultimate goal in this master thesis. Ultimately, to provide a better understanding of how the adapters package of ZK Ajax engine are structured and coded, its all related source codes can be found in the appendix A which is located in the end of the entire report.

5.4.2 Execution of ZK Ajax Engine via Richlet

In this section, we are going to elaborate how our ZK Ajax engine will be run in a realistic case and the main requirements for having it run as a ZK project in Eclipse properly. To do this, we first needed to have our own executable .jar files of the Wazaabi model, core engine, ZK Ajax engine, and core model. Therefore, we already exported our source codes as executable .jar files and imported them as referenced web application libraries in a sample ZK project which can run on the localhost from Apache Tomcat Server 7 which provides a Java servlet for the web environment.

After an investigation from the web as well as information obtained from our supervisor, we had to implement a ZKRichlet which is a small Java code block that creates all necessary components in response to user request. First, we used the "org.zkoss.zk.ui.Richlet" interface, then we declared the association of our richlet with the URL for the actual implementation. The appearance of ZKRichlet project and the required libraries to run it properly are shown in the following figure;



ZKRichlet
src
org.zkoss.zk.richlet
ZKRichlet.java
JRE System Library [jdk1.6.0_24]
Apache Tomcat v7.0 [Apache Tomcat v7.0]
Web App Libraries
Referenced Libraries
org.eclipse.emf.common_2.6.0.v20100914-1218.jar -
org.eclipse.emf.ecore.xmi_2.5.0.v20100521-1846.jar
org.eclipse.emf.ecore_2.6.1.v20100914-1218.jar - C:\
org.eclipse.pmf.wazaabi.model.zk_1.0.0.jar - C:\Use
org.eclipse.pmf.wazaabi.engine.core_0.0.1.jar - C:\U
org.eclipse.pmf.wazaabi.engine.zk_0.0.1.jar - C:\Use
org.eclipse.pmf.wazaabi.model.core_0.0.1.jar - C:\U
build
WebContent

Figure 5.26: ZKRichlet as ZK Project

After that, we had to map the URL to our ZKRichlet via two successive steps which are turning on the ZKRichlet and then mapping URL pattern to it. By default, the ZKRichlet was disabled. In order to enable it, we have added the following declaration by editing the generated web.xml file in the WEB-INF folder of our ZK project[41]. The required declaration to enable ZKRichlet is shown in the following figure;

```
<servlet-mapping>
<servlet-name>zkLoader</servlet-name>
<url-pattern>/zk/*</url-pattern>
</servlet-mapping>
```

Figure 5.27: Turning on ZKRichlet

Once the ZKRichlet was enabled, we had to map a URL pattern to it by editing the generated zk.xml file in the WEB-INF folder of our ZK project [41]. So that, we could request our ZKRichlet by running the ZK application on Apache Tomcat Server 7 and visiting "http://localhost:8080/ZKRichlet/zk/test" in our web browser The required declaration to map a URL to the ZKRichlet is shown in the following figure;

```
<richlet>
<richlet-name>Test</richlet-name>
<richlet-class>org.zkoss.zk.richlet.ZKRichlet</richlet-class>
</richlet>

<richlet-mapping>
<richlet-name>Test</richlet-name>
<url-pattern>/test</url-pattern>
</richlet-mapping>
```

Figure 5.28: Mapping URL Pattern to ZKRichlet

After that, it is necessary to load ZK model for embedding definitions and properties of the ZK components such as Label, Panel, Textbox and Button. By doing this, we can set the model elements from the resource as a content for the viewer. In order to get a fully understanding, this will be explained in more detail in the evaluation scenarios. In the following figure, the code block in ZKRichlet which are used to load the EMF resource is given;

```
Resource.Factory.Registry.INSTANCE
.getExtensionToFactoryMap()
.put("zkmodel", new XMIResourceFactoryImpl());
org.eclipse.pmf.wazaabi.model.zk.widgets
.WidgetsPackage.eINSTANCE.eClass();
Resource resource = new ResourceSetImpl().getResource(URI
   .createURI("models/ZKRichlet.zkmodel"), true);
viewer.setContents(resource.getEObject("/"));
```

Figure 5.29: Loading the EMF Resource

In addition, the figure below shows the structure of ZK model instance by using sample reflective ecore model editor in Eclipse;

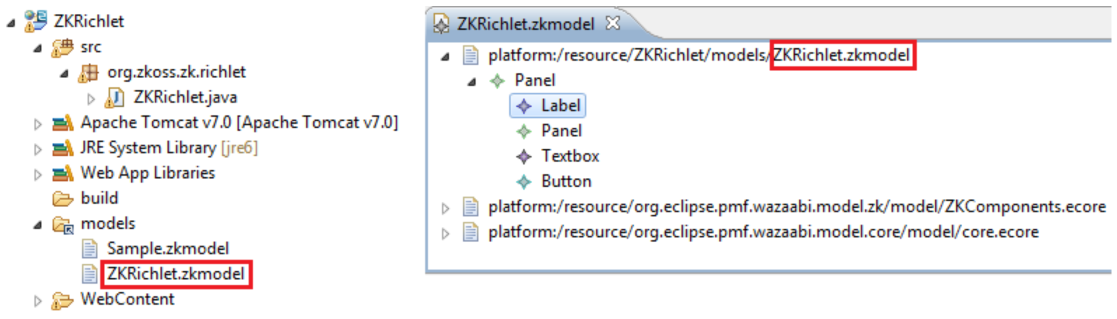

ZKRichlet
src
org.zkoss.zk.richlet
ZKRichlet.java
Apache Tomcat v7.0 [Apache Tomcat v7.0]
JRE System Library [jre6]
Web App Libraries
build
models
Sample.zkmodel
ZKRichlet.zkmodel
WebContent
ZKRichlet.zkmodel
platform:/resource/ZKRichlet/models/ZKRichlet.zkmodel
Panel
Label
Panel
Textbox
Button
platform:/resource/org.eclipse.pmf.wazaabi.model.zk/model/ZKComponents.ecore
platform:/resource/org.eclipse.pmf.wazaabi.model.core/model/core.ecore

Figure 5.30: ZKRichlet.zkmodel

5.5 Software Technologies & Tools

This section discusses the different programs (software applications and technologies) which were used during the project.

5.5.1 Development Platforms & Softwares

This section would describe the different development platforms and software applications which were used during the development phase.

Eclipse Helios

The popular development platform Eclipse created by the open source group with the same name. It provides additional features for Java and web development to the existing Eclipse application such as code completion and manual lookup. Eclipse was used by the members of the development team for the implementation of the Java client part.

ZK Studio

ZK studio of Potix corporation is an open source Ajax application development environment having user interface development tools which supports WYSIWYG visual development.

Apache Tomcat 7

Apache Tomcat is an open source servlet container which provides Java servlets and support HTTP web server environment. It contains web tools for XML configuration files and server management.

Latex

Latex is a document markup language and document preparation system. Latex simplifies the process of organizing and compiling large individual documents into a single contiguous representation. The project report is written in several different text editors and then compiled into a single document by using latex. Several smaller documents were also written using this technique like the meeting minutes and agendas. This was done to preserve formatting across some documents produced during the project. We used LyX as our document processor for the project report.

SVN

SVN is an abbreviation for Subversion, which is a version tracking system for all types of files. SVN also resolves most situations where several different people might be writing on the same document by allowing different changes to be merged together without loss of data. Merging features works in trivial cases and for text files only. For binary files no merging is done. For text files the merging needs to be done manually if the operation is not trivial. Our project requires simultaneous development of both the project report and the source code, which makes SVN feasible. The repository was used for distributing background material, meeting summaries, images, figures and the project report. The group members uses different tools for connecting to the SVN repository during the project, including through the Eclipse plugin Subclipse and using the command line client.

Mozilla Firefox

Mozilla Firefox is a web browser released by Mozilla Corporation. It is one of the most widely used free and open source browser which has functions that can be added through extensions which are created by developers.

Microsoft Internet Explorer

Internet Explorer (IE) is Microsoft's free web browser provided by Microsoft Windows operating systems. It is the most widely used graphical web browser since 1999.

Google Chrome

Google Chrome is a web browser developed by Google that uses the WebKit layout engine. As 2011, Google Chrome is the third most widely used browser with a 12 percent worldwide usage share of web browsers, according to Net Applications.

5.5.2 Programing languages

This section describes the different programming languages utilized for development. It also includes markup languages such as HTML and XML.

HTML

HTML is an acronym for Hypertext Markup Language and is a markup language that only allows certain tags to be used and tags are only allowed to be used in certain scopes. HTML is the standard language for website development.

XML

XML is an acronym for e-Xtensible Markup Language and is a specification for representing data using markup language. It builds on the same ideas as those behind HTML. XML describes a data structure that is highly flexible and relatively easy to read yet simple for a computer to understand and process. Our solution uses XML in several places, the most prominent being the communication between the client side and server side.

JavaScript

JavaScript is a client side scripting language most often used for client side dynamic web development. JavaScript enables the creation of web applications that are able to respond to the users' inputs without requesting a new page from the server. This is especially useful when one wishes to create a page that is user friendly and does not require redirection to a new page each time the user performs an action.

Ajax

Asynchronous JavaScript and XML which can be abbreviated as Ajax is a technique used in web application development. Generally, Ajax technique is mainly used on the client part in order to create interactive web applications. By using Ajax, the client part of a web application can retrieve data asynchronously from the server part which works in the background without making any alterations of display and behavior of the existing pages.

5.5.3 Sharing Platforms & Tools

NTNU Student E-Mail Service

This service is used for sharing documents and communication about the status between group members and the supervisor with getting an access by using a browser or an email client such as Microsoft Outlook.

It’s Learning

It’s learning is a web based learning platform to support students with different kinds of academic education all around the world. We used for sharing the resources between the group members and posting the related articles we found.

Assembla

Assembla which is a collaborative project management service was used as a sharing platform for the documents and a repository of the source code related to the project.

GoogleDocs

GoogleDocs is a document sharing service from Google that allows uploading and editing documents online by using different computers. This service is used for making backups of the document versions in order to secure the documents of the project report.

Microsoft Word

Microsoft Word is a word processing program from Microsoft. It is used extensively for proofreading and writing documents before they were implemented in latex.

Notepad++

Notepad++ is a free editor which allows viewing the source code with syntax of many programming languages. It is used for editing Java source codes and XML configuration files.

5.6 Chapter Summary

We started the Implementation chapter by giving the implementation overview which gives a very brief description of how we could contribute to the existing work of Wazaabi project according to our main goals as well as taking into account the proposed solution chapter. Therefore, our aim of this chapter was to explain how we could support the existing work of Wazaabi project with Ajax technique in order to make it be able to deal with the web browser based user interfaces via setting up our own Ajax engine instead of using the existing Wazaabi engine which was based on SWT. Then, we continued our discussion with studying the architechture of ZK Ajax framework as well as its relevant and various components as a whole. We also elaborated the main interactions of the ZK archtitecture, the relationship between its components, the hierarchy of its components, how to implement ZK components, setting up user interfaces via ZK and updating web pages according to the ZK asynchronous update engine at server part which receives the requests from client part. At this point, according to our goals for the implementation phase which we already comprised, the significant details about the binding process of Ajax support to the existing framework of Wazaabi project was discussed in general. Then, as a main contribution to our project, the major aspects of the new Ajax approach was described in detail. Additionally, the section which we cover that how we could integrate the ZK framework with the existing work of Wazaabi project was one of the most important sections in the entire thesis report. Since, not only it puts emphasis on the main idea of our master thesis but also it elaborated the main possible contribution which we formerly studied on the previous chapters. At that point, we also mentioned about how we implemented our ZKRichlet for executing our own ZK Ajax engine within the Wazaabi framework in detail. Before we close the implementation chapter and dive into the evaluation section, we wanted to touch on briefly the different software technologies and development tools that we had used during the research, implementation and documentation phases of our master thesis.

Chapter 6

Evaluation

This chapter of the report is reserved for the evaluation of the processes within the project and the results that we have obtained. The purpose of this chapter is to verify how well our proposed solution and the objectives in the project are achieved. Doing a deep analysis of the results will benefit us by pointing out and analysing how different parts of the project work affected the overall results.

6.1 Evaluation Overview

The aim of the evaluation chapter is to illustrate the general assessment activities that have taken place right after the implementation part as well as documenting the test results about our contribution for the existing work of Wazaabi project. Therefore, the evaluation section may also be considered as a set of formal methods and assessment criterias which have to be followed for revealing to what extent the requirements and main goals are fulfilled in our master thesis.

In the rest of the evaluation chapter, our general scope will be to cover all the assessment criterias which are performed on the resulted work of our master thesis as well as fulfilling the requirement specifications. Since the final implementation is an Ajax contribution to an existing project in this field, our evaluation plan will be supposed to test new functionalities introduced by the resulted implementation under the category of two main evaluation scenarios. In other words, our testing activities will not be focused on existing functionalities of Wazaabi project.

The assessment criterias are not aimed to go into every detail of the existing implementation, but they are still concerning about larger parts of the project. In this manner, we can verify our resulted implementation is able to integrate Ajax technology into the existing work of Wazaabi project properly.

6.2 Evaluation Scenarios

This section is based on two main evaluation scenarios that shows the possible behaviours of the implemented ZK Ajax engine. The implemented solution is evaluated using these two scenarios which mention about adding or editing ZK Ajax components which come from model instance to be reflected in the user interface and updating properties of ZK Ajax components by using onClick events.

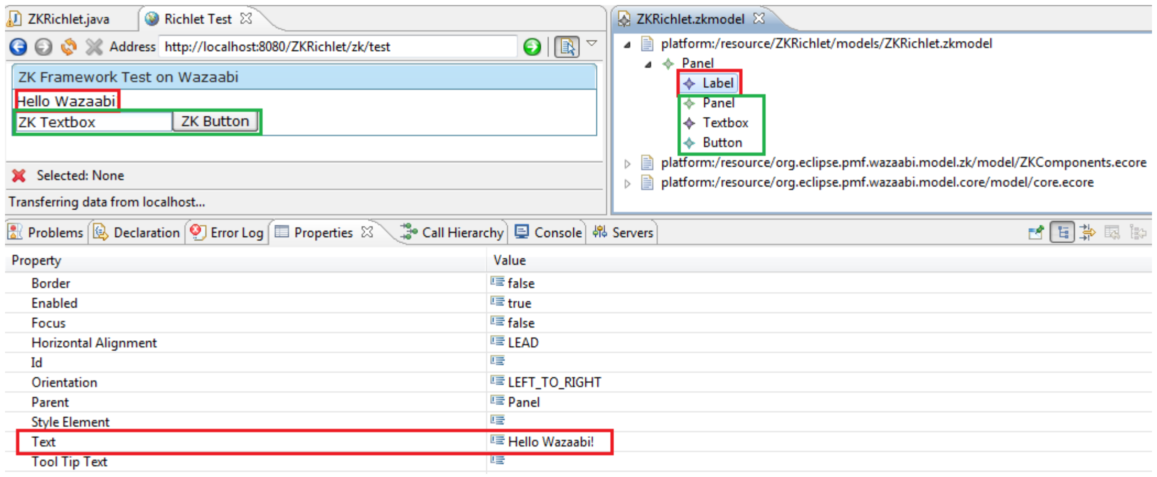
Scenario 1: Adding & Editing ZK Component

This scenario presents how to add or edit a ZK Ajax component in the user interface via our ZK Ajax engine. Basically, we can get the components by programming in the ZK richlet such as creating component instances one by one and setting their parent to panel as a ZK container.

```
ZKControlViewer viewer = new ZKControlViewer(window);
Panel panel = WidgetsFactory.eINSTANCE.createPanel();
final Label label = WidgetsFactory.eINSTANCE.createLabel();
label.setText("hello␣wazaabi!");
Button button = WidgetsFactory.eINSTANCE.createButton();
button.setText("ZK␣BUTTON");
label.setParent(panel);
button.setParent(panel);
viewer.setContents(panel);
```

Figure 6.1: Adding Button & Label via programming

However, adding or editing a model instance with ecore editor is more useful way than programming. Firstly, we need to implement a ZK richlet which is already explained in the previous chapter then we need to create a dynamic instance called "ZKRichlet.zkmodel" from the ZK model (org.eclipse.pmf.wazaabi.model.zk). By opening this created model instance via sample reflective ecore model editor, we are allowed to create new model elements and edit existing ones. Later, the EMF source of the created ZK model instance should be loaded in the richlet5.29. The main point is that this load process must be done in order to get the proper ZK Ajax components by the help of our ZK Ajax engine (org.eclipse.pmf.wazaabi.engine.zk) which updates the view according to the hierarchy of model elements in the model instance. In addition, setting the contents into the window by the help of ZK control viewer should be done in the richlet so that the ZK Ajax components can be retrieved properly. The adding a sibling next to child process is illustrated in the following figure;


ZKRichlet.java
Richlet Test
Address http://localhost:8080/ZKRichlet/zk/test
ZK Framework Test on Wazaabi
Hello Wazaabi
ZK Textbox
ZK Button
Selected: None
Transferring data from localhost...
ZKRichlet.zkmodel
platform:/resource/ZKRichlet/models/ZKRichlet.zkmodel
Panel
Label
Panel
Textbox
Button
platform:/resource/org.eclipse.pmf.wazaabi.model.zk/model/ZKComponents.ecore
platform:/resource/org.eclipse.pmf.wazaabi.model.core/model/core.ecore
Problems
Declaration
Error Log
Properties
Call Hierarchy
Console
Servers

Property	Value
Border	false
Enabled	true
Focus	false
Horizontal Alignment	LEAD
Id	
Orientation	LEFT_TO_RIGHT
Parent	Panel
Style Element	
Text	Hello Wazaabi!
Tool Tip Text	

Figure 6.2: Adding & Editing ZK Component via Editor

The places in the red mark show how to make changes in the properties of one of the ZK Ajax components. In this scenario, the text of the label component is changed from the default to “Hello Wazaabi!”. The detailed code group of this scenario can be found in Appendix A:A.2.

Scenario 2: Updating ZK Component via onClick Event

In the second evaluation scenario, we aim to illustrate how a ZK Ajax component can affect the user interface elements which are in type of Wazaabi model widgets. By doing this, we have a ZK button component where we imported and defined it as “org.zkoss.zul.Button” in our source code and a couple of various Wazaabi model widgets which are located in a panel element. In addition, we also used the EventListener feature of the ZK button as well as its onClick event in order to change text attribute of the label which is located in a panel element. To make it more sense as well as getting a fully understanding of how it performs, we would like to provide the code block which is responsible for changing the text attribute of the label element in the following figure;

```
final org.zkoss.zul.Button button =
new org.zkoss.zul.Button("Change␣Label");
button.addEventListener(Events.ON_CLICK,
new EventListener() {
public void onEvent(Event evt) {
label.setText("Goodbye␣wazaabi!");
}
});
```

Figure 6.3: onClick event of org.zkoss.zul.Button

According to the main logic behind the Ajax technique used in this evaluation scenario, the relevant property of the label which is "text" in this case will be changed and the change is shown immediately as the user clicks on the ZK Button named "Change Label" without refreshing the entire web page as expected.

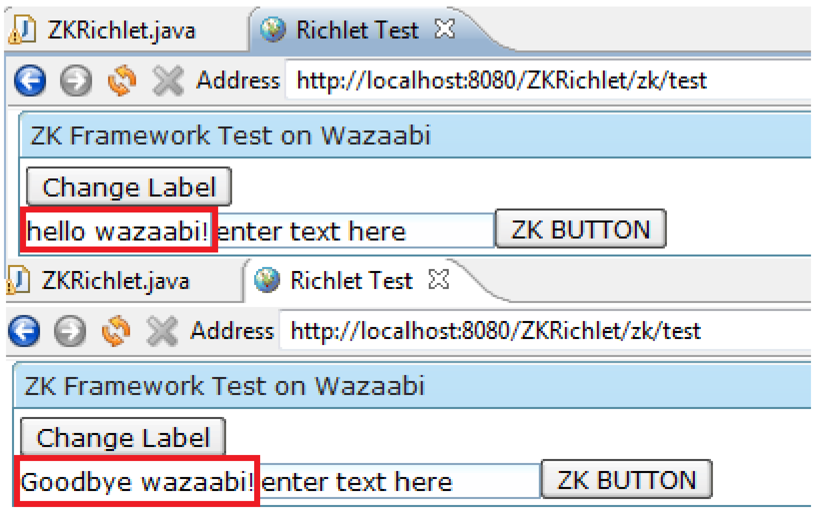


Figure 6.4: Changing ZK Label via onClick Event of ZK Button

As soon as we run our code on the server which works on localhost, the resulted screen that is triggered by the onClick event is depicted in the figure above.

6.3 Project Results

According to the motivation and the main objectives as well as the information and the feedback obtained from the supervisor during the time span of this master thesis, the result was expected to be the Proactive Ajax Approach which enables Ajax technology in order to make the Wazaabi project have support for the web browser based user interfaces. Therefore, our initial and ultimate aim was to embed Ajax technique into the existing work of Wazaabi project and make sure that the resulted implementation has support for the web browser based user interfaces which was a deficiency in the Wazaabi formerly.

In the project results section, our general scope will be to elaborate all results from our former research, our own Ajax approach3.3.2, the resulted implementation with Wazaabi5.4.1, and even our own experiences in supporting the web based user interfaces by using Ajax technique3.4.1. In this manner, we can verify that our contribution to the existing work of Wazaabi project was based on binding ZK Ajax framework to the Wazaabi as an extension, like opening a new chapter in a book as well as adhering to our ultimate aim in supporting web based user interfaces successfully in the Wazaabi. Now, we are going to illustrate the accomplishment status of the requirements that we are supposed to achieve. The table below covers both functional and non-functional requirements of the current project.

Requirement	Priority	Succeeded	Description
FR-AJX1	High	✓	A new Ajax engine (render) is required.
FR-AJX2	High	✓	Editparts must be changed specific to Ajax framework.
FR-AJX3	Medium	✓	Ajax widgets must be integrated into the new engine.
FR-WZB1	High	✓	Wazaabi must support to create new Ajax UIs.
FR-WZB2	Medium	✓	Wazaabi must be able to edit different Ajax UI components.
NF-AJX1	High	✓	Ajax support must be integrated into Wazaabi.
NF-AJX2	Medium	✓	Ajax must be able to identify current changes.
NF-AJX3	Low	✘	Ajax Security Issues must be properly defined.
NF-WZB1	High	✓	Wazaabi must conform to the technical constraints put by Ajax.
NF-WZB2	High	✓	Target model(Ajax) must be shown in Web Browser.
NF-WZB3	Medium	✘	An editor which allows changes in the Wazaabi Viewer.

Table 6.1: Fulfillment Table of Requirements

To adhere our ultimate goal in this master thesis, we created a new Ajax engine (a new rendering system) which was embeded into the Wazaabi's framework and replaced with the existing SWT engine by using ZK Ajax framework in the implementation phase of this master thesis. In our new Ajax engine, Editpart module is also created by matching with existing ZK Ajax components. Then, it is fully integrated into the new Ajax engine whereas Editparts for controling ZK Ajax components is recreated according to the newly created ZK ecore model and binded to Wazaabi core engine. Thus, the other Ajax requirement FR-AJX3 for our engine is also satisfied. Despite not all Ajax widgets are used, most important Ajax widgets as examples are integrated to the engine. This leads us to create other Ajax widgets in the future in a more stable way.

With the help of our own ZK Ajax engine, Wazaabi can associate user interface information while creating a new user interface and editing different Ajax user interface components. Our newly created Views and Adapters in the ZK Ajax engine supports different kinds of Ajax user interface components like Textboxes, Labels, Buttons and Panels as container. ZKEditparts takes important role while editing ZK Ajax components by making a bridge between the user interface and the ZK model instance. The structure of the engine is also well binded to the Wazaabi core engine so that we can say this kind of Ajax technology (ZK Ajax Framework) starts to be supported by Wazaabi. The requirement NF-AJX3 is dropped out of the project because our main focus has become implementing the engine, not testing the Ajax security issues while executing the engine. Although it is not in our scope, the security is an important topic which should be focused as a type of requirement that may be fulfilled in the future work.

Naturally, every framework has different structure and used methods in their own architecture so that Wazaabi must have conformed to the technical constraints put by Ajax. In other words, there were conditions which ZK Ajax framework did not have any methods to support these conditions. It is also realized that there were incompatible parts which are irrelevant for our integration and we had to clarify the technical constraints to be more precise in our work. For example, we saw that not every methods of Wazaabi are fully matched with methods in the libraries of ZK Ajax framework.

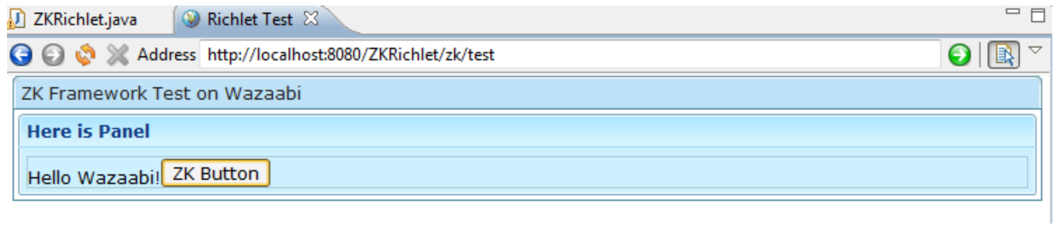


Figure 6.5: ZK Components in the ZK User Interface

With the execution of a richlet modified with our ZK Ajax engine, the instance of the target (Ajax) model has been displayed in a web browser properly. To implement the richlet, it needs Java code rather than XML and

ZK Loader which processes the richlet and creates components handled by the richlet. Integrated ZK Ajax components are loaded from model resource by the help of our ZK Ajax engine.

The engine needed to be aware of the model which is being used in and if possible posses the ability to adapt the user interface according to the model instance. Moreover, information about the user interface affects the model context. Our last requirement here was to supply an editor to allow changes in the Wazaabi Viewer but on the other hand as the project is evolved, the importance of this requirement is slightly decreased because it is still possible to utilize the existing reflective ecore model editor and palette as long as they are based on the EMF.

The extraction from the reviews on the requirement table show that through using ZK Ajax framework and its relevant features, the existing work of Wazaabi project was supported in web browser based user interfaces and this drawback of the Wazaabi was properly solved with Ajax technique.

6.4 Chapter Summary

We started this chapter with the evaluation of the succeeded objectives for measuring the success in this project. An introduction on the evaluated sections of the project is given to acquire an overview of the influential, both negative and positive aspects of the project. Especially, what kind of objectives are motivated and how they are studied under the light of researches about Ajax frameworks and Ajax techniques. We discussed a set of formal methods which are supposed to be followed to accomplish the tasks we gained in the beginning of our master thesis. Two related scenarios which are performed on the project work are evaluated as well as our own contribution: the ZK Ajax engine and the ZK model which introduces Proactive Ajax Approach. The evaluation continued with project results including the table of fulfillment of the requirements to verify our resulted implementation which integrates Ajax technique into the existing work of Wazaabi project properly. In other words, the significant results about the integration of frameworks are given to what extent the requirements are met throughout the implementation phase of the entire project. This leads us to know what contributions are properly made by us and what missing tasks are skipped and left for forthcoming improvements as a future work.

Chapter 7

Conclusion

This last chapter explains the contributions which the group performed throughout the project by summarizing the report and also gives refined reflections on all aspects of the project. In addition, there will be a part about the future work which shares further ideas for the possible future work including our positive and negative experiences about how we can do it in different ways in a later project.

7.1 Summary of the Thesis

In the rest of this paper, we are going to wrap up whole topics in the project chapters and give a short summary of what we have contributed with this master thesis. First, we defined our objectives with the help of our supervisor by discussing the existing Ajax frameworks and which one we are going to focus on. At the same time, we studied the Wazaabi framework which was totally a new framework for us and it can be said that there was so little documentation about the framework. Especially, we had difficulties while learning the concept and reading the source code. In chapter 2: State of the Art, we gave a general information about Ajax technology and mentioned different Ajax frameworks. We also gave a brief explanation of what we understand about Wazaabi framework. Moreover, the constraints of the both frameworks are explained. In order to make our task more clear, we have defined the problem relating with both existing frameworks and how models are supported in these frameworks, we gave four different Ajax scenarios to show the Ajax techniques which are used as a mechanism in various situations. Then, to see the big picture of the road ahead, we defined a requirement table which contains functional and non functional requirements of the given frameworks. These requirements reflected to our proposed solution in a sense that it created a guideline for our approach: Proactive Ajax Approach. The extension of the existing Wazaabi framework which we called "Proactive Ajax Approach" is a combination of all Ajax ideas that we we discussed during the project. The solution was to bind Ajax framework to Wazaabi to support Ajax user interface components. As a result of this approach, we created a new Ajax engine to implement into Wazaabi's framework. Last but not least, some of the components of the selected Ajax framework (ZK) were integrated into the existing framework (Wazaabi). Clearly, our own ZK Ajax engine with using Ajax techniques offered the required Ajax support for Wazaabi project which now can support web browser based user interfaces.

Our approach is a starting point for those wishing to embed Ajax frameworks into Wazaabi or integrate different web technologies to extend the Wazaabi framework. All in all, we gained a valuable experience in the area of Ajax technology as well as learning a new declarative user interface modeling framework called Wazaabi. Revisions between the draft and the final version of this report were influenced by discussions as well as comments made by our supervisor, Hallvard Trætteberg.

7.2 Future Work

In the last section, we are going to explain feasibility issues of the frameworks for the future improvements including shortcomings of the existing technologies that we analyzed during the project. We will also present our recommendation about what should be done in the future.

The first issue is about ZK Ajax components in the market to create Ajax based user interfaces. ZK provides this support by giving the framework releases periodically. The basic version has a disadvantage because it has only a small amount of components and to access to premium version which contains whole ZK components and tools requires license payment. Nevertheless, it is still possible to integrate different ZK components into Wazaabi framework like ComboBox, CheckBox, Toolbar, etc.

The second issue is that currently, most of the websites are accessed by computer users so that the user interfaces of web applications are designed for computer screens. But innovative web technologies are taking to web into the next level, from our computer screens to mobile devices. Developing user interfaces for mobile web applications becomes more and more popular. Most of the popular websites have their own mobile websites which makes them really accessable at any time. After analyzing the Wazaabi, we realized that the framework has drawbacks when it comes to developing mobile user interfaces. Neither its editor nor architect component (UI modeler) is ready for supporting mobile user interfaces. The Wazaabi framework may be modified in a sense that it can support editing user interfaces of mobile applications. In addition to the existing framework, new mobile user interface technologies can also be supported with their own components.

The last issue which needs to be considered for the future work is the security aspect of Ajax frameworks in creating dynamic and interactive web pages. As we already mentioned before this new technology of Ajax also brings drawbacks such as vulnerabilities occured during autogenerated pages without need to be refreshed.

Bibliography

[1] IDI, NTNU Master Thesis Website. (http://www.idi.ntnu.no/education/prosjektoppgaver.php?p_id=819) Accessed on 17th January 2011.

[2] Wazaabi Website. (http://wazaabi.org/) Accessed on 17th January 2011.

[3] Eclipse Modeling Website. (http://www.eclipse.org/modeling/emf/) Accessed on 21st January 2011.

[4] Wazaabi Website Documentation (http://wazaabi.org/documentation) Accessed on 17th January 2011.

[5] Wikipedia - Ajax Programming (http://en.wikipedia.org/wiki/Ajax_(programming)) Accessed on 19th January 2011.

[6] Hertel, M. (2007) Aspects of Ajax (http://www.mathertel.de/AJAX/AJAXeBook.aspx) Accessed on 28th January 2011.

[7] Johnson, D., White, A., Charland, A. (2008) Enterprise Ajax: Strategies for Building High Performance Web Applications. Prentice Hall. Accessed on 1st February 2011.

[8] What is Ajax Technology? (http://www.webhostdesignpost.com/website/webtechnology-ajax.html) Accessed on 3rd February 2011.

[9] AJAX Application Architecture (http://msdn.microsoft.com/en-us/magazine/cc163363.aspx) Accessed on 3rd February 2011.

[10] Schutta, N., Asleson, R. (2006) Pro Ajax and Java Frameworks. Apress. Accessed on 16th February 2011.

[11] Google Search Engine. (http://www.google.com) Accessed on 17th February 2011.

[12] Google Maps. (http://www.maps.google.com) Accessed on 17th February 2011.

[13] Google Translate. (http://www.translate.google.com) Accessed on 17th February 2011.

[14] Google Web Toolkit. (http://code.google.com/intl/en-EN/webtoolkit/) Accessed on 17th February 2011.

[15] Rich Ajax Platform. (http://www.eclipse.org/rap/) Accessed on 17th February 2011.

[16] Dojo Toolkit. (http://dojotoolkit.org/) Accessed on 17th February 2011.

[17] Vaadin Framework. (http://vaadin.com/home) Accessed on 17th February 2011.

[18] ZK Ajax Framework. (http://www.zkoss.org/) Accessed on 17th February 2011.

[19] Direct Web Remoting. (http://getahead.org/dwr) Accessed on 21st February 2011.

[20] Wikipedia Ajax Framework. (http://en.wikipedia.org/wiki/Ajax_framework) Accessed on 11th February 2011.

[21] Ajax Frameworks. (http://www.xul.fr/ajax-frameworks.html) Accessed on 12th February 2011.

[22] Ajax Frameworks. (http://ajaxwith.com/Ajax-Frameworks.html) Accessed on 12th February 2011.

[23] Open Source Ajax Frameworks. (http://www.java-source.net/open-source/ajax) Accessed on 13th February 2011.

[24] Ajax Programming. (http://www.effectivesoft.com/ajax_development.html) Accessed on 19th February 2011.

[25] What is AJAX? (http://www.pritambaldota.com/index.php/what-is-ajax/) Accessed on 24th February 2011.

[26] Wazaabi Framework Overview. (http://wazaabi.org/documentation/framework-overview) Accessed on 16th February 2011.

[27] Editing UIs in Wazaabi. (http://wazaabi.org/documentation/framework-overview/editing-wazaabi-ui) Accessed on 16th February 2011.

[28] Ajax Push & Animation: Drag & Drop List. (http://www.zkoss.org/zksandbox/#a2) Accessed on 2nd March 2011.

[29] Ajax Events & Scripts: Keystroke Events. (http://www.zkoss.org/zkdemo/event/keystroke_event) Accessed on 3rd March 2011.

[30] Ajax Comboxes: AutoComplete Combobox. (http://www.zkoss.org/zkdemo/combobox/autocomplete) Accessed on 2nd March 2011.

[31] Ajax Charts: PieChart. (http://www.zkoss.org/zkdemo/chart/pie_chart) Accessed on 3rd March 2011.

[32] ZK Client Side Reference: Widget Package Descriptor. (http://books.zkoss.org/wiki/ZK_Client-side_Reference/Widget_Package_Descriptor) Accessed on 5th March 2011.

[33] ZK Ajax Framework Developers Reference. (http://books.zkoss.org/wiki/ZK_Developer%27s_Reference) Accessed on 13th March 2011.

[34] Stauble, M., Schumacher, H. (2008) ZK Developer's Guide. Packt Publishing. Accessed on 17th March 2011.

[35] Chen, H., Cheng, R. (2007) ZK Ajax Without JavaScript Framework. Apress. Accessed on 20th March 2011.

[36] The project page of XUL (http://www.mozilla.org/projects/xul/) Accessed on 25th March 2011.

[37] ZK Ajax Framework Architecture. (http://books.zkoss.org/wiki/Small_Talks/2007/July/Behind_The_Scene:_Integrating_Ext_Grid) Accessed on 26th March 2011.

[38] ZK Essentials. (http://books.zkoss.org/wiki/ZK_Essentials) Accessed on 26th March 2011.

[39] ZK Getting Started Tutorial. (http://books.zkoss.org/wiki/ZK_Getting_Started/Tutorial#Define_UI_in_pure_Java) Accessed on 25th March 2011.

[40] Composing UIs in ZK Framework. (http://books.zkoss.org/images/d/d6/ZKEssentials_Intro_Hello.png) Accessed on 26th March 2011.

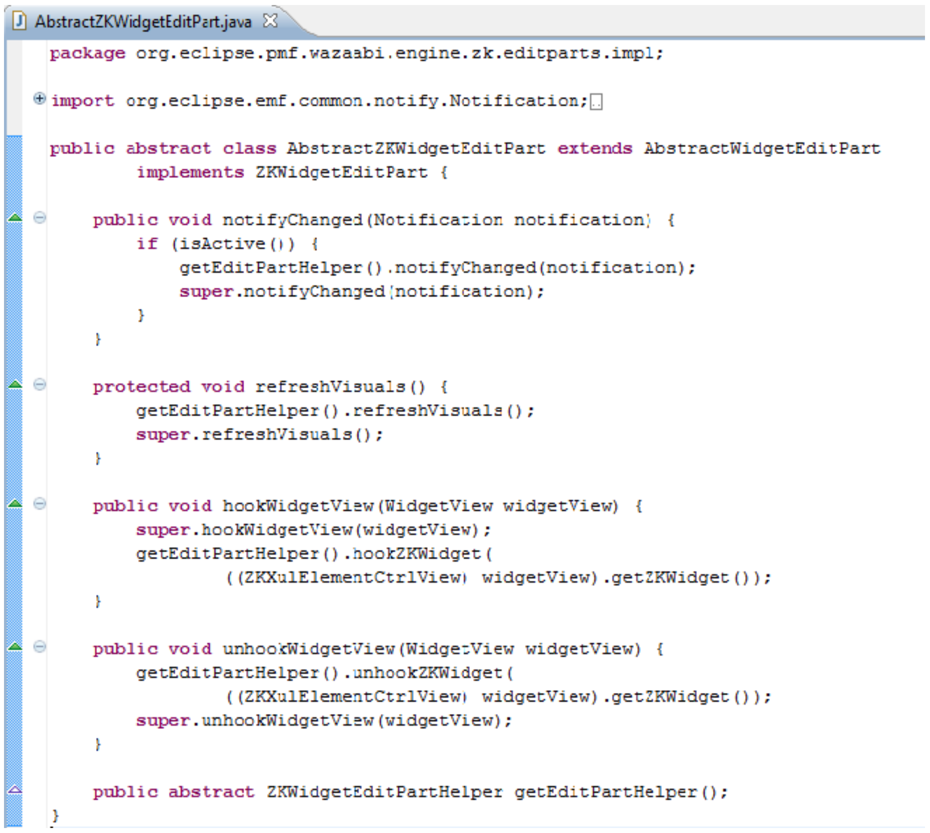
[41] Implementing a ZK Richlet (http://books.zkoss.org/wiki/ZK%20Developer's%20Reference/UI%20Composing/Richlet) Accessed on 12th May 2011.

Appendix A

Source Codes

A.1 ZK Engine

A.1.1 EditParts

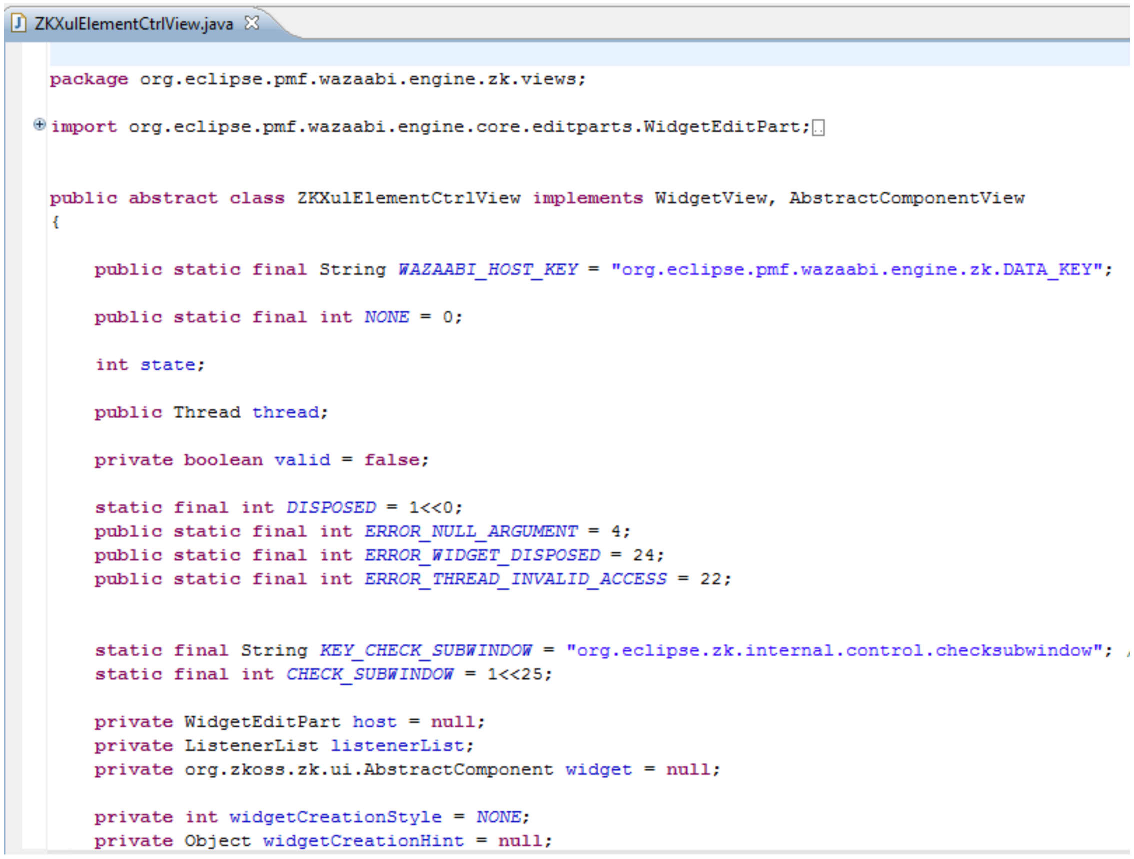


AbstractZKWidgetEditPart.java

```java
package org.eclipse.pmf.wazaabi.engine.zk.editparts.impl;

import org.eclipse.emf.common.notify.Notification;

public abstract class AbstractZKWidgetEditPart extends AbstractWidgetEditPart
        implements ZKWidgetEditPart {

    public void notifyChanged(Notification notification) {
        if (isActive()) {
            getEditPartHelper().notifyChanged(notification);
            super.notifyChanged(notification);
        }
    }

    protected void refreshVisuals() {
        getEditPartHelper().refreshVisuals();
        super.refreshVisuals();
    }

    public void hookWidgetView(WidgetView widgetView) {
        super.hookWidgetView(widgetView);
        getEditPartHelper().hookZKWidget(
                ((ZKXulElementCtrlView) widgetView).getZKWidget());
    }

    public void unhookWidgetView(WidgetView widgetView) {
        getEditPartHelper().unhookZKWidget(
                ((ZKXulElementCtrlView) widgetView).getZKWidget());
        super.unhookWidgetView(widgetView);
    }

    public abstract ZKWidgetEditPartHelper getEditPartHelper();
}
```

Figure A.1: Source Code of AbstractZKWidgetEditPart.java

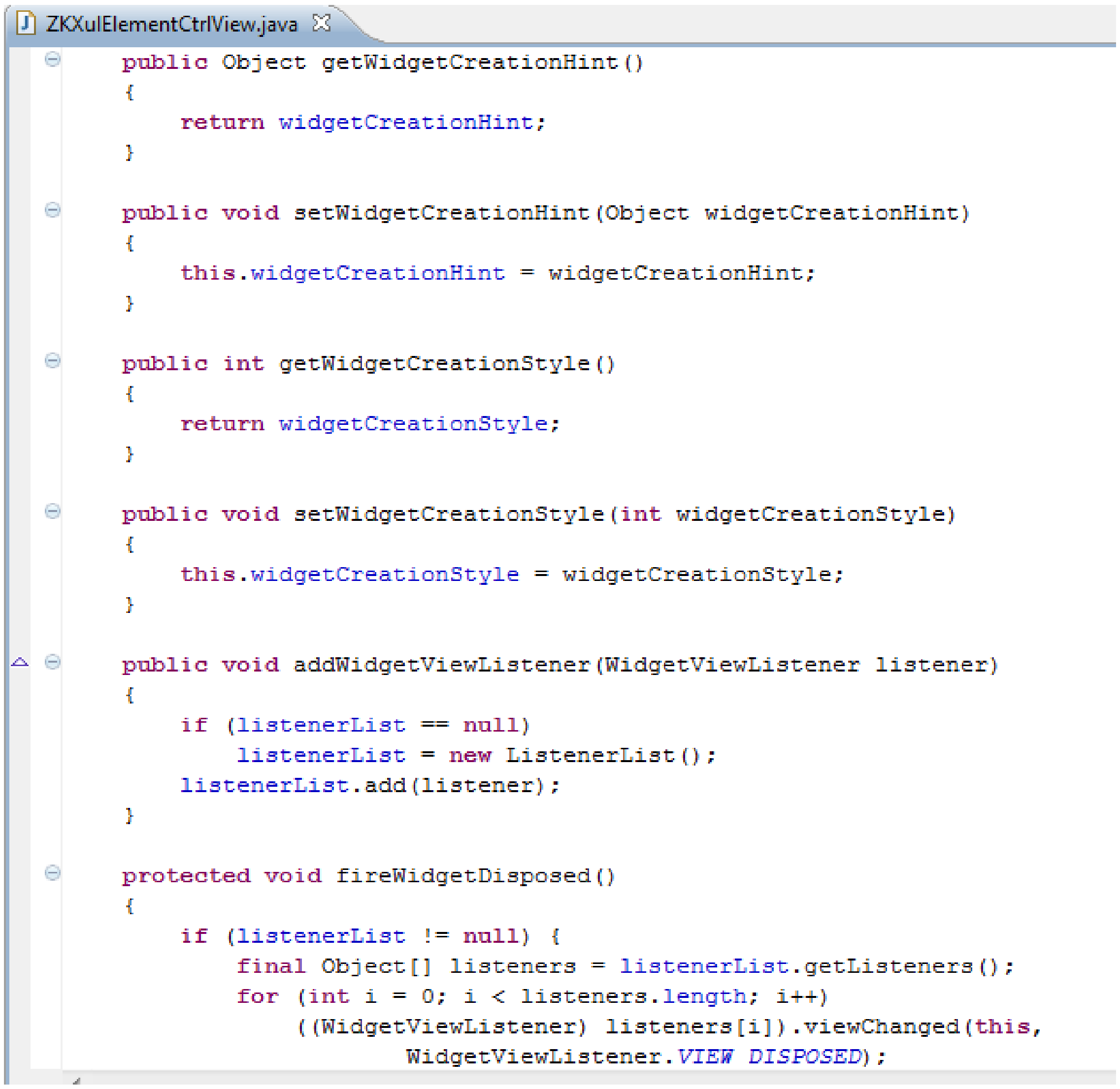
A.1.2 Views



ZKXulElementCtrlView.java

```
package org.eclipse.pmf.wazaabi.engine.zk.views;

import org.eclipse.pmf.wazaabi.engine.core.editparts.WidgetEditPart;


public abstract class ZKXulElementCtrlView implements WidgetView, AbstractComponentView
{

    public static final String WAZAABI_HOST_KEY = "org.eclipse.pmf.wazaabi.engine.zk.DATA_KEY";

    public static final int NONE = 0;

    int state;

    public Thread thread;

    private boolean valid = false;

    static final int DISPOSED = 1<<0;
    public static final int ERROR_NULL_ARGUMENT = 4;
    public static final int ERROR_WIDGET_DISPOSED = 24;
    public static final int ERROR_THREAD_INVALID_ACCESS = 22;


    static final String KEY_CHECK_SUBWINDOW = "org.eclipse.zk.internal.control.checksubwindow";
    static final int CHECK_SUBWINDOW = 1<<25;

    private WidgetEditPart host = null;
    private ListenerList listenerList;
    private org.zkoss.zk.ui.AbstractComponent widget = null;

    private int widgetCreationStyle = NONE;
    private Object widgetCreationHint = null;
```

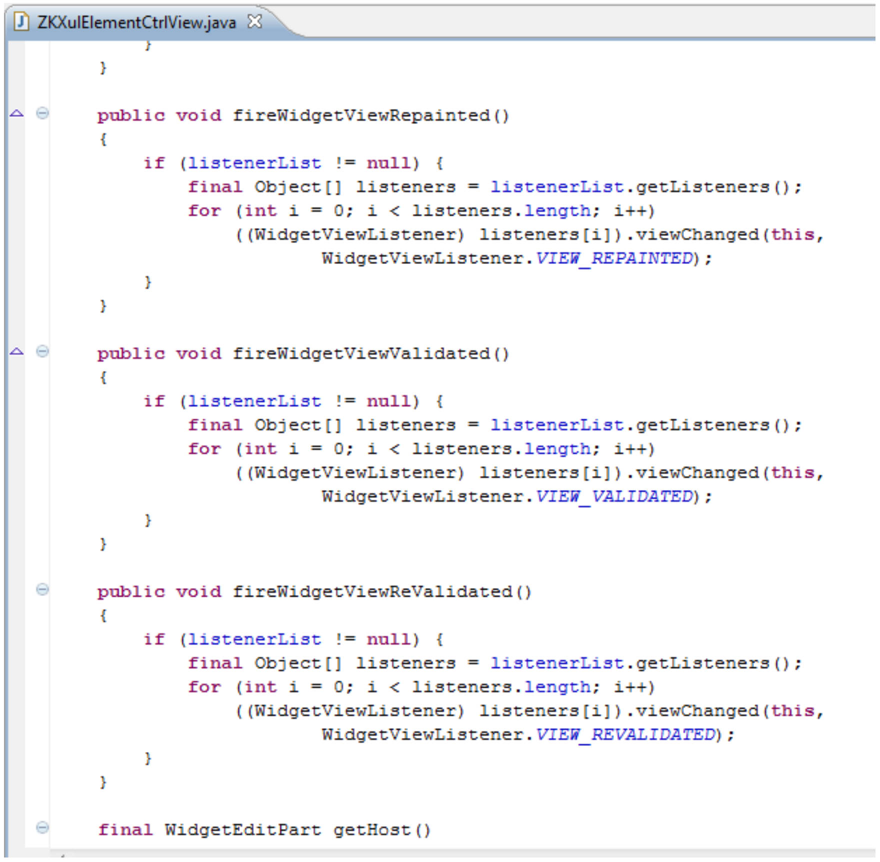
Figure A.2: Source Code of ZKXulElementCtrlView.java Part 1



ZKXulElementCtrlView.java

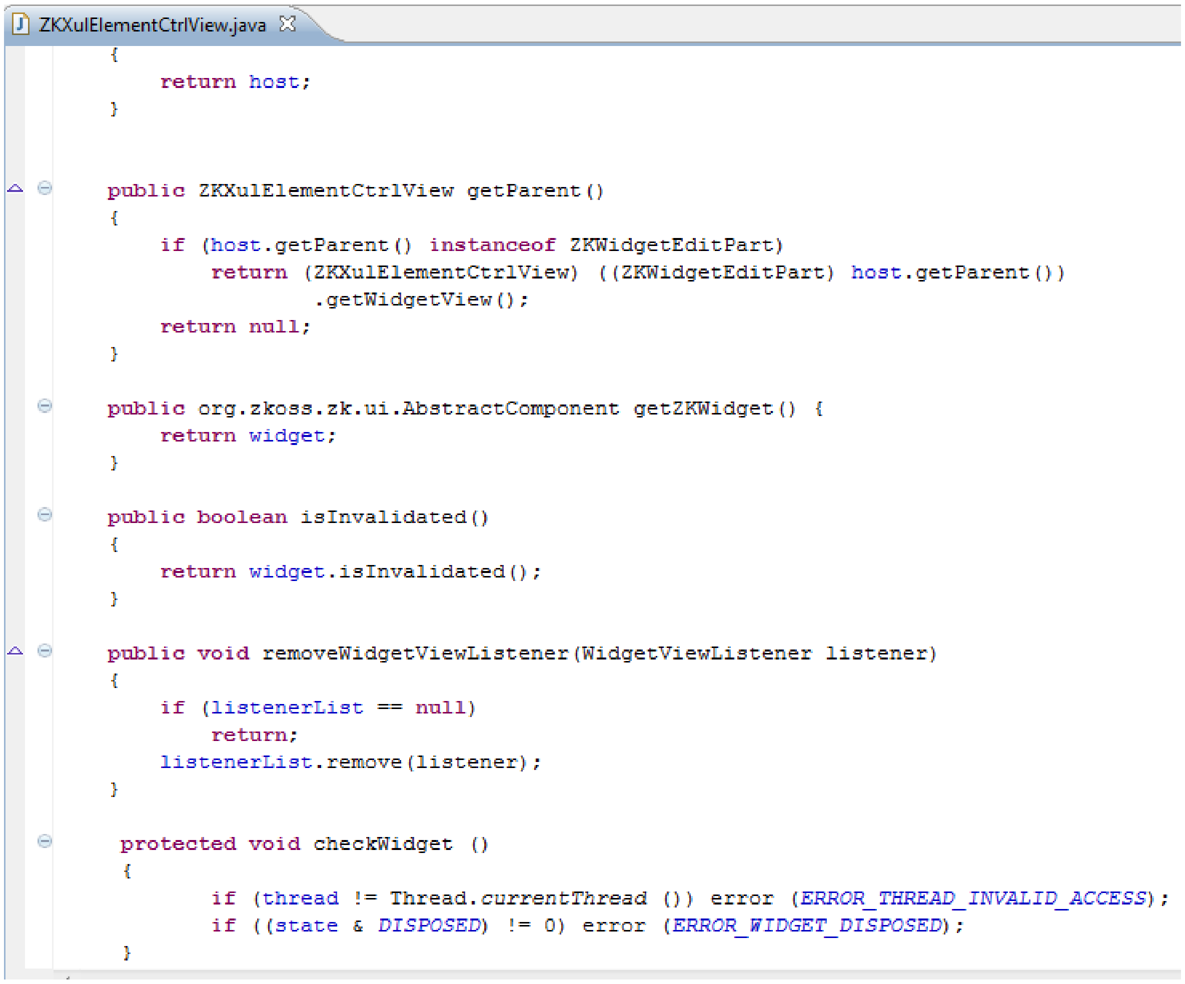
```
    public Object getWidgetCreationHint()
    {
        return widgetCreationHint;
    }

    public void setWidgetCreationHint(Object widgetCreationHint)
    {
        this.widgetCreationHint = widgetCreationHint;
    }

    public int getWidgetCreationStyle()
    {
        return widgetCreationStyle;
    }

    public void setWidgetCreationStyle(int widgetCreationStyle)
    {
        this.widgetCreationStyle = widgetCreationStyle;
    }

    public void addWidgetViewListener(WidgetViewListener listener)
    {
        if (listenerList == null)
            listenerList = new ListenerList();
        listenerList.add(listener);
    }

    protected void fireWidgetDisposed()
    {
        if (listenerList != null) {
            final Object[] listeners = listenerList.getListeners();
            for (int i = 0; i < listeners.length; i++)
                ((WidgetViewListener) listeners[i]).viewChanged(this,
                        WidgetViewListener.VIEW_DISPOSED);
```

Figure A.3: Source Code of ZKXulElementCtrlView.java Part 2



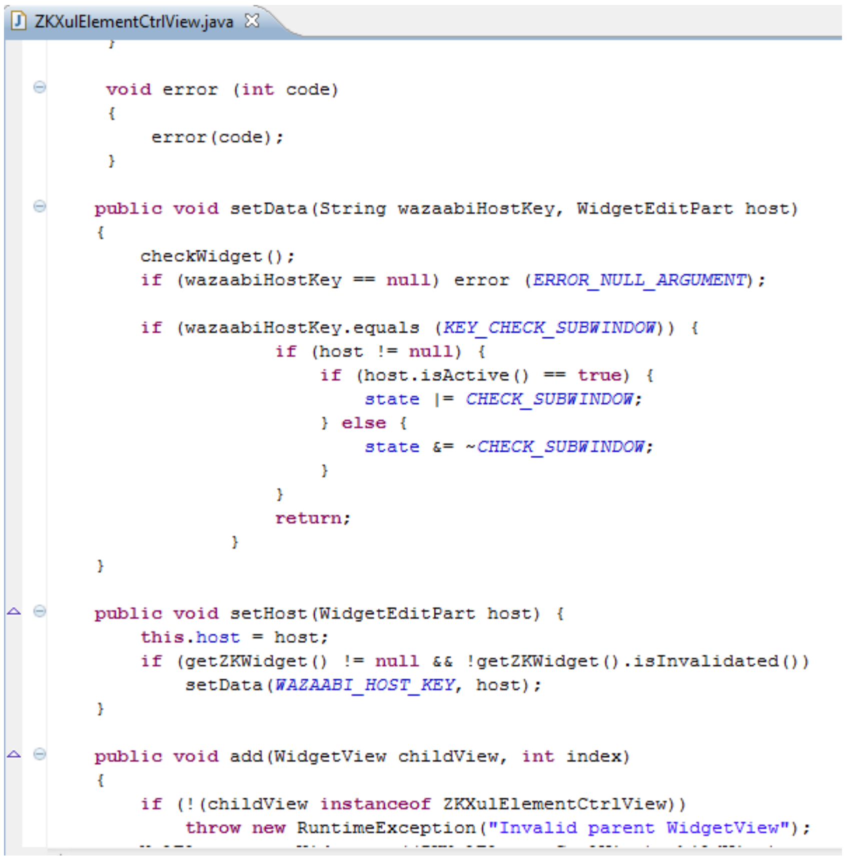
```
    }

    public void fireWidgetViewRepainted()
    {
        if (listenerList != null) {
            final Object[] listeners = listenerList.getListeners();
            for (int i = 0; i < listeners.length; i++)
                ((WidgetViewListener) listeners[i]).viewChanged(this,
                        WidgetViewListener.VIEW_REPAINTED);
        }
    }

    public void fireWidgetViewValidated()
    {
        if (listenerList != null) {
            final Object[] listeners = listenerList.getListeners();
            for (int i = 0; i < listeners.length; i++)
                ((WidgetViewListener) listeners[i]).viewChanged(this,
                        WidgetViewListener.VIEW_VALIDATED);
        }
    }

    public void fireWidgetViewReValidated()
    {
        if (listenerList != null) {
            final Object[] listeners = listenerList.getListeners();
            for (int i = 0; i < listeners.length; i++)
                ((WidgetViewListener) listeners[i]).viewChanged(this,
                        WidgetViewListener.VIEW_REVALIDATED);
        }
    }

    final WidgetEditPart getHost()
```

Figure A.4: Source Code of ZKXulElementCtrlView.java Part 3



```
ZKXulElementCtrlView.java
    {
        return host;
    }


    public ZKXulElementCtrlView getParent()
    {
        if (host.getParent() instanceof ZKWidgetEditPart)
            return (ZKXulElementCtrlView) ((ZKWidgetEditPart) host.getParent())
                    .getWidgetView();
        return null;
    }

    public org.zkoss.zk.ui.AbstractComponent getZKWidget() {
        return widget;
    }

    public boolean isInvalidated()
    {
        return widget.isInvalidated();
    }

    public void removeWidgetViewListener(WidgetViewListener listener)
    {
        if (listenerList == null)
            return;
        listenerList.remove(listener);
    }

     protected void checkWidget ()
     {
            if (thread != Thread.currentThread ()) error (ERROR_THREAD_INVALID_ACCESS);
            if ((state & DISPOSED) != 0) error (ERROR_WIDGET_DISPOSED);
     }
```

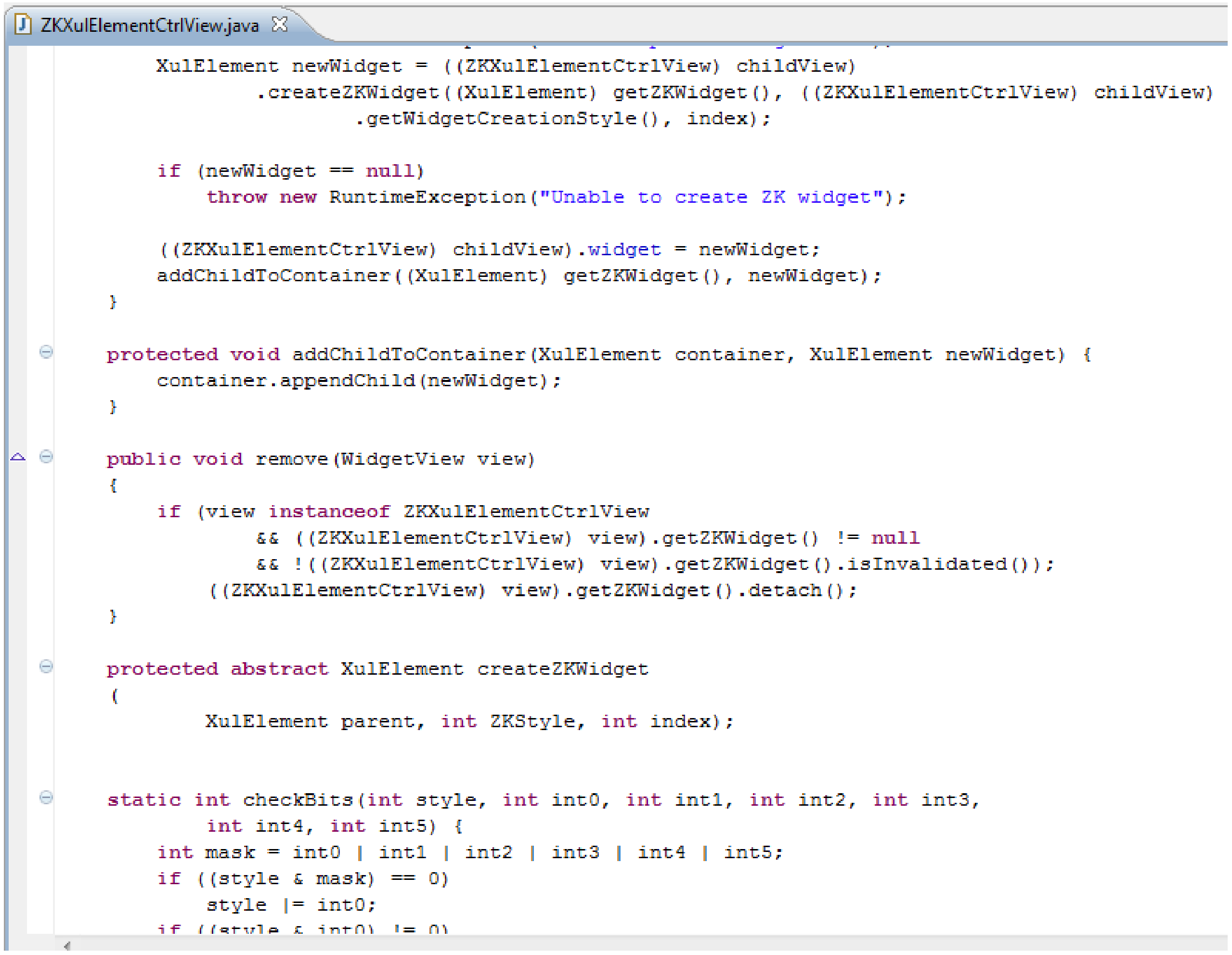
Figure A.5: Source Code of ZKXulElementCtrlView.java Part 4



ZKXulElementCtrlView.java

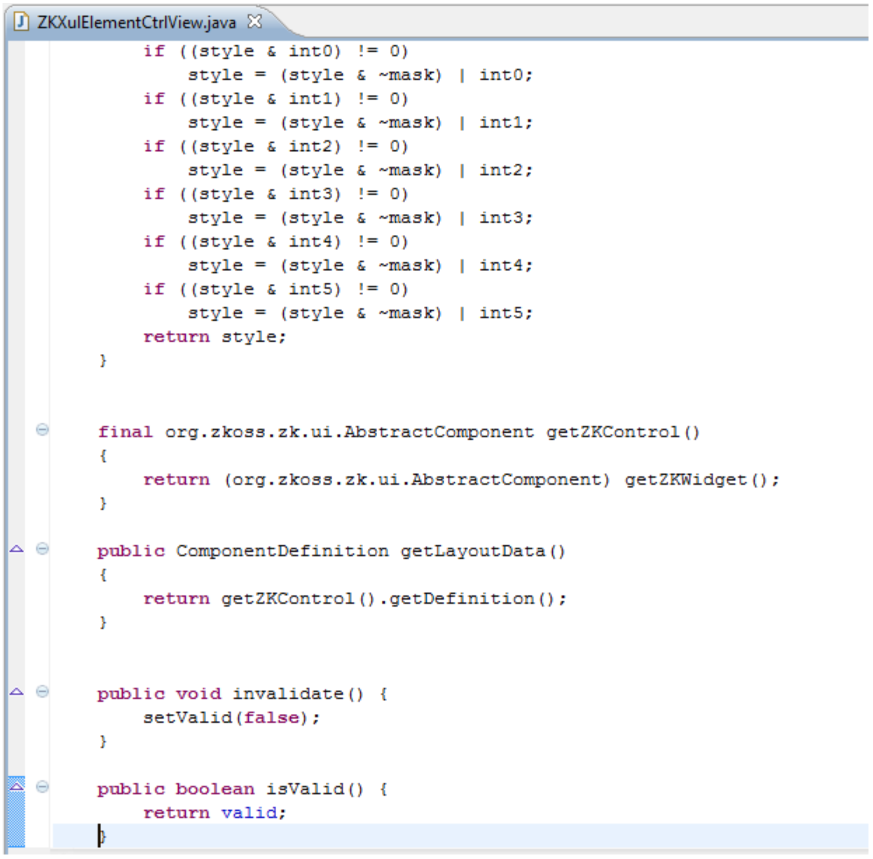
```
        }

         void error (int code)
         {
             error(code);
         }

        public void setData(String wazaabiHostKey, WidgetEditPart host)
        {
            checkWidget();
            if (wazaabiHostKey == null) error (ERROR_NULL_ARGUMENT);

            if (wazaabiHostKey.equals (KEY_CHECK_SUBWINDOW)) {
                        if (host != null) {
                            if (host.isActive() == true) {
                                state |= CHECK_SUBWINDOW;
                            } else {
                                state &= ~CHECK_SUBWINDOW;
                            }
                        }
                        return;
                    }
        }

        public void setHost(WidgetEditPart host) {
            this.host = host;
            if (getZKWidget() != null && !getZKWidget().isInvalidated())
                setData(WAZAABI_HOST_KEY, host);
        }

        public void add(WidgetView childView, int index)
        {
            if (!(childView instanceof ZKXulElementCtrlView))
                throw new RuntimeException("Invalid parent WidgetView");
```

Figure A.6: Source Code of ZKXulElementCtrlView.java Part 5



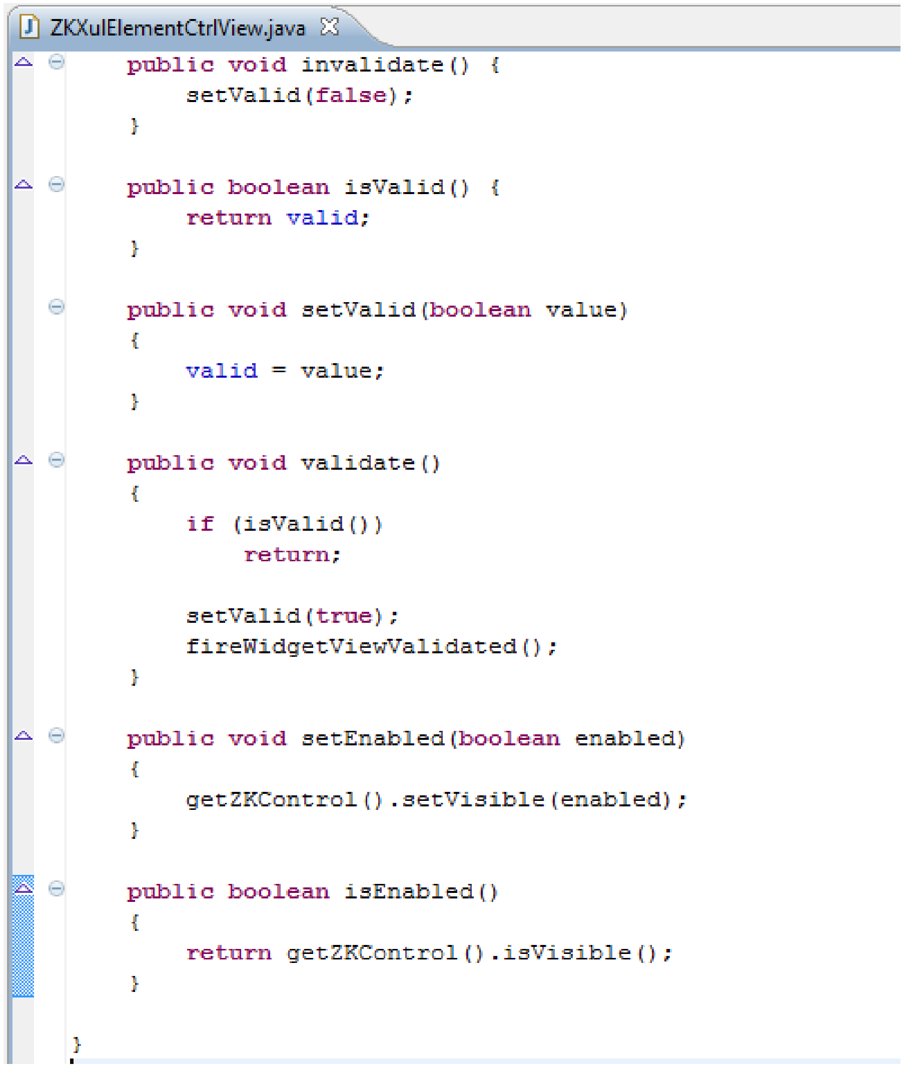
```
ZKXulElementCtrlView.java
        XulElement newWidget = ((ZKXulElementCtrlView) childView)
                .createZKWidget((XulElement) getZKWidget(), ((ZKXulElementCtrlView) childView)
                        .getWidgetCreationStyle(), index);

        if (newWidget == null)
            throw new RuntimeException("Unable to create ZK widget");

        ((ZKXulElementCtrlView) childView).widget = newWidget;
        addChildToContainer((XulElement) getZKWidget(), newWidget);
    }

    protected void addChildToContainer(XulElement container, XulElement newWidget) {
        container.appendChild(newWidget);
    }

    public void remove(WidgetView view)
    {
        if (view instanceof ZKXulElementCtrlView
                && ((ZKXulElementCtrlView) view).getZKWidget() != null
                && !((ZKXulElementCtrlView) view).getZKWidget().isInvalidated());
            ((ZKXulElementCtrlView) view).getZKWidget().detach();
    }

    protected abstract XulElement createZKWidget
    (
            XulElement parent, int ZKStyle, int index);


    static int checkBits(int style, int int0, int int1, int int2, int int3,
            int int4, int int5) {
        int mask = int0 | int1 | int2 | int3 | int4 | int5;
        if ((style & mask) == 0)
            style |= int0;
```

Figure A.7: Source Code of ZKXulElementCtrlView.java Part 6



```java
        if ((style & int0) != 0)
            style = (style & ~mask) | int0;
        if ((style & int1) != 0)
            style = (style & ~mask) | int1;
        if ((style & int2) != 0)
            style = (style & ~mask) | int2;
        if ((style & int3) != 0)
            style = (style & ~mask) | int3;
        if ((style & int4) != 0)
            style = (style & ~mask) | int4;
        if ((style & int5) != 0)
            style = (style & ~mask) | int5;
        return style;
    }


    final org.zkoss.zk.ui.AbstractComponent getZKControl()
    {
        return (org.zkoss.zk.ui.AbstractComponent) getZKWidget();
    }

    public ComponentDefinition getLayoutData()
    {
        return getZKControl().getDefinition();
    }


    public void invalidate() {
        setValid(false);
    }

    public boolean isValid() {
        return valid;
    }
```

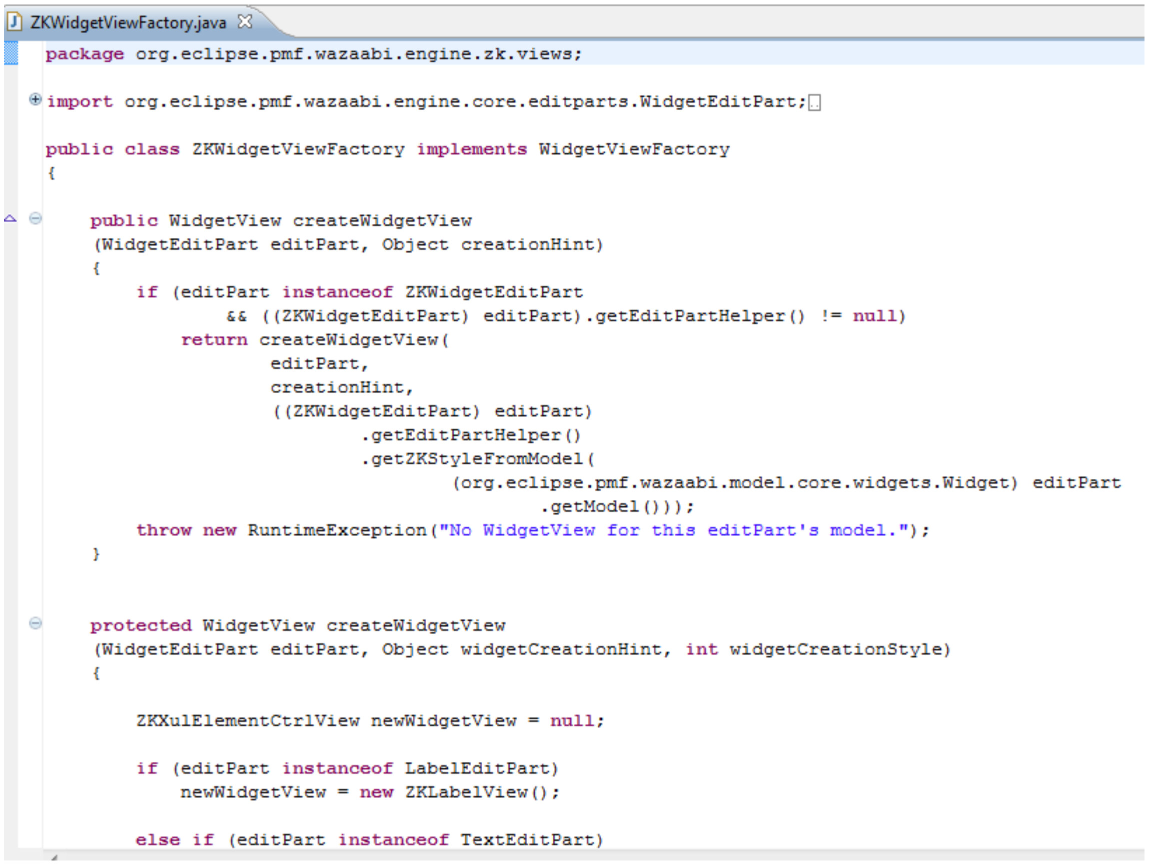
Figure A.8: Source Code of ZKXulElementCtrlView.java Part 7



ZKXulElementCtrlView.java

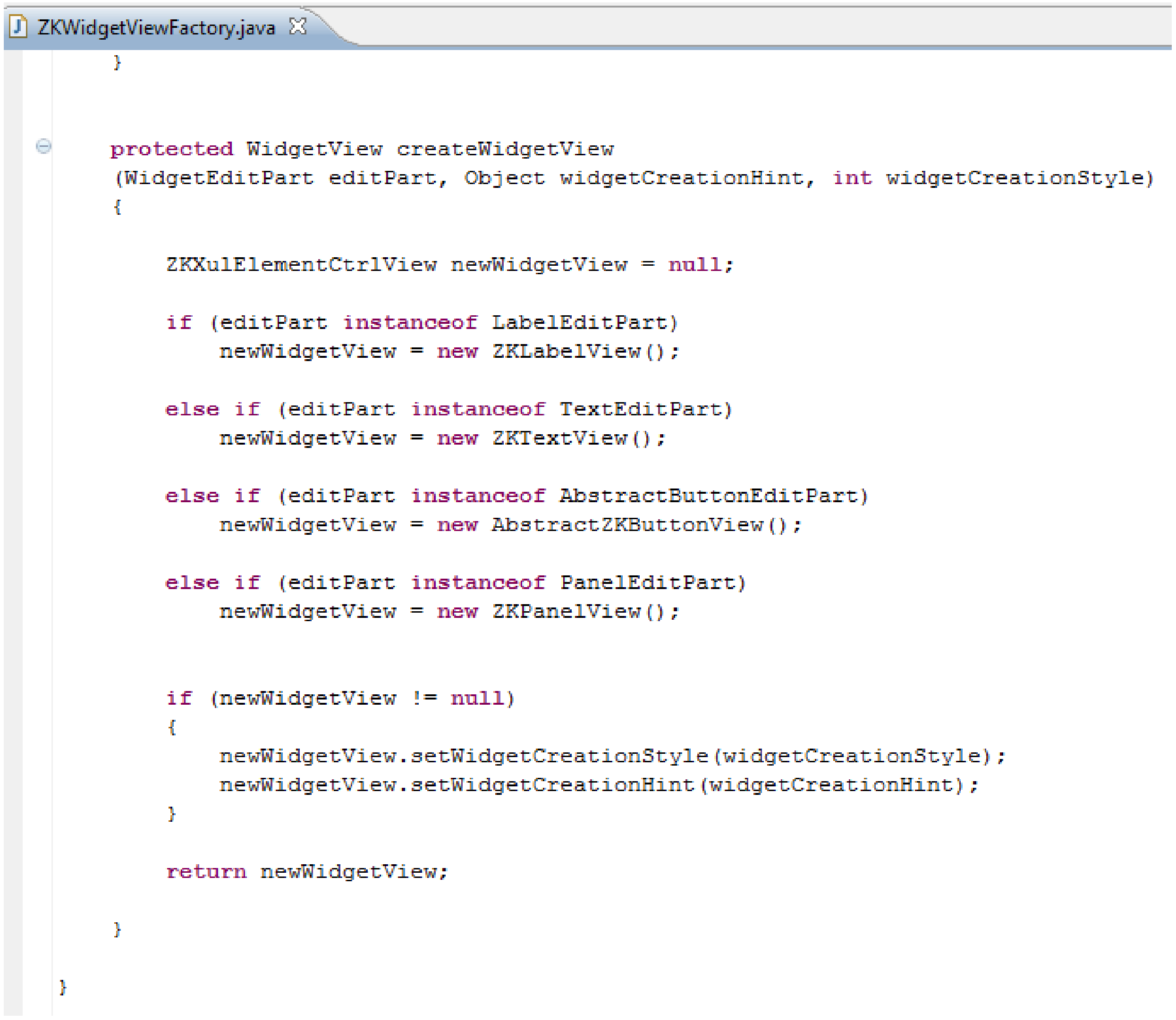
```
    public void invalidate() {
        setValid(false);
    }

    public boolean isValid() {
        return valid;
    }

    public void setValid(boolean value)
    {
        valid = value;
    }

    public void validate()
    {
        if (isValid())
            return;

        setValid(true);
        fireWidgetViewValidated();
    }

    public void setEnabled(boolean enabled)
    {
        getZKControl().setVisible(enabled);
    }

    public boolean isEnabled()
    {
        return getZKControl().isVisible();
    }

}
```

Figure A.9: Source Code of ZKXulElementCtrlView.java Part 8



```java
package org.eclipse.pmf.wazaabi.engine.zk.views;

import org.eclipse.pmf.wazaabi.engine.core.editparts.WidgetEditPart;

public class ZKWidgetViewFactory implements WidgetViewFactory
{

    public WidgetView createWidgetView
    (WidgetEditPart editPart, Object creationHint)
    {
        if (editPart instanceof ZKWidgetEditPart
                && ((ZKWidgetEditPart) editPart).getEditPartHelper() != null)
            return createWidgetView(
                    editPart,
                    creationHint,
                    ((ZKWidgetEditPart) editPart)
                            .getEditPartHelper()
                            .getZKStyleFromModel(
                                    (org.eclipse.pmf.wazaabi.model.core.widgets.Widget) editPart
                                            .getModel()));
        throw new RuntimeException("No WidgetView for this editPart's model.");
    }


    protected WidgetView createWidgetView
    (WidgetEditPart editPart, Object widgetCreationHint, int widgetCreationStyle)
    {

        ZKXulElementCtrlView newWidgetView = null;

        if (editPart instanceof LabelEditPart)
            newWidgetView = new ZKLabelView();

        else if (editPart instanceof TextEditPart)
```

Figure A.10: Source Code of ZKWidgetViewFactory.java Part 1

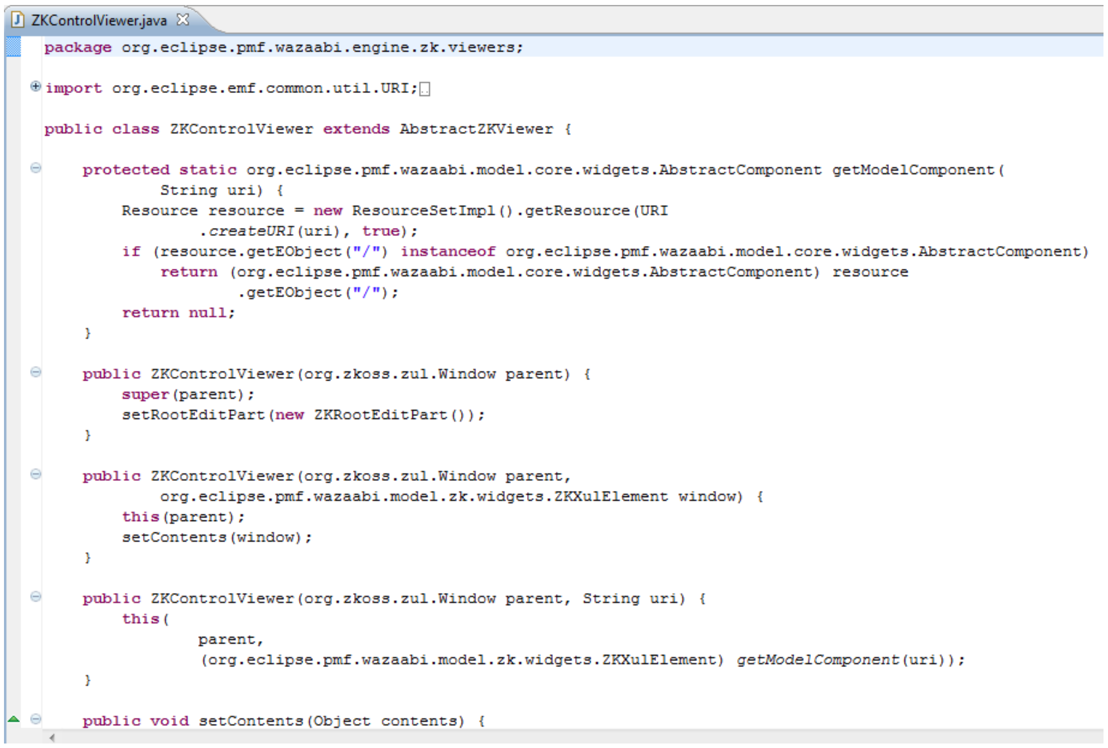


ZKWidgetViewFactory.java

```
    }


    protected WidgetView createWidgetView
    (WidgetEditPart editPart, Object widgetCreationHint, int widgetCreationStyle)
    {

        ZKXulElementCtrlView newWidgetView = null;

        if (editPart instanceof LabelEditPart)
            newWidgetView = new ZKLabelView();

        else if (editPart instanceof TextEditPart)
            newWidgetView = new ZKTextView();

        else if (editPart instanceof AbstractButtonEditPart)
            newWidgetView = new AbstractZKButtonView();

        else if (editPart instanceof PanelEditPart)
            newWidgetView = new ZKPanelView();


        if (newWidgetView != null)
        {
            newWidgetView.setWidgetCreationStyle(widgetCreationStyle);
            newWidgetView.setWidgetCreationHint(widgetCreationHint);
        }

        return newWidgetView;

    }

}
```

Figure A.11: Source Code of ZKWidgetViewFactory.java Part 2

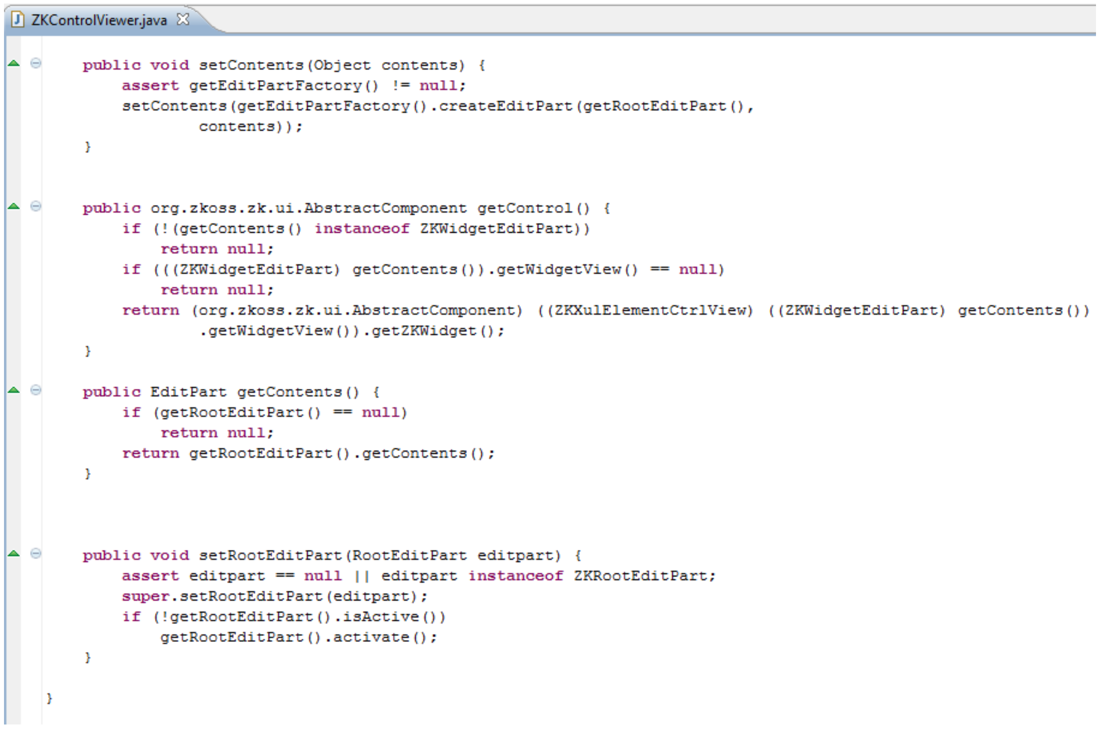
A.1.3 Viewers



ZKControlViewer.java

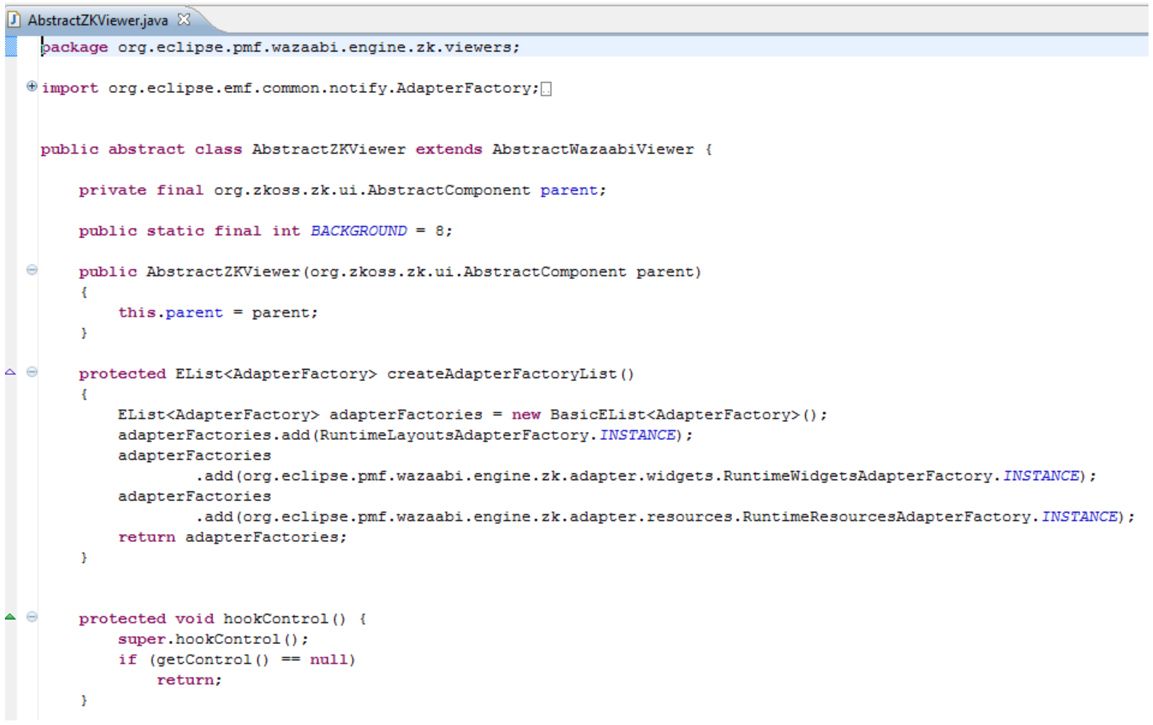
```java
package org.eclipse.pmf.wazaabi.engine.zk.viewers;

import org.eclipse.emf.common.util.URI;

public class ZKControlViewer extends AbstractZKViewer {

    protected static org.eclipse.pmf.wazaabi.model.core.widgets.AbstractComponent getModelComponent(
            String uri) {
        Resource resource = new ResourceSetImpl().getResource(URI
                .createURI(uri), true);
        if (resource.getEObject("/") instanceof org.eclipse.pmf.wazaabi.model.core.widgets.AbstractComponent)
            return (org.eclipse.pmf.wazaabi.model.core.widgets.AbstractComponent) resource
                    .getEObject("/");
        return null;
    }

    public ZKControlViewer(org.zkoss.zul.Window parent) {
        super(parent);
        setRootEditPart(new ZKRootEditPart());
    }

    public ZKControlViewer(org.zkoss.zul.Window parent,
            org.eclipse.pmf.wazaabi.model.zk.widgets.ZKXulElement window) {
        this(parent);
        setContents(window);
    }

    public ZKControlViewer(org.zkoss.zul.Window parent, String uri) {
        this(
                parent,
                (org.eclipse.pmf.wazaabi.model.zk.widgets.ZKXulElement) getModelComponent(uri));
    }

    public void setContents(Object contents) {
```

Figure A.12: Source Code of ZKControlViewer.java Part 1


ZKControlViewer.java

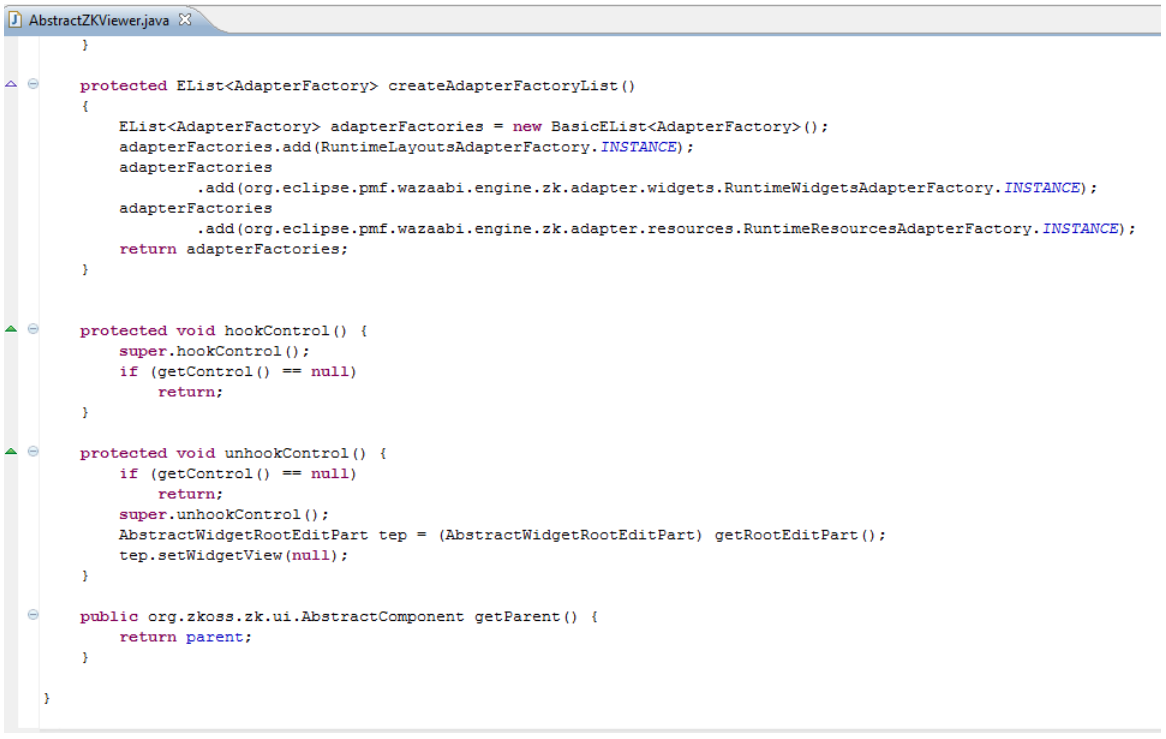
```
    public void setContents(Object contents) {
        assert getEditPartFactory() != null;
        setContents(getEditPartFactory().createEditPart(getRootEditPart(),
                contents));
    }


    public org.zkoss.zk.ui.AbstractComponent getControl() {
        if (!(getContents() instanceof ZKWidgetEditPart))
            return null;
        if (((ZKWidgetEditPart) getContents()).getWidgetView() == null)
            return null;
        return (org.zkoss.zk.ui.AbstractComponent) ((ZKXulElementCtrlView) ((ZKWidgetEditPart) getContents())
                .getWidgetView()).getZKWidget();
    }

    public EditPart getContents() {
        if (getRootEditPart() == null)
            return null;
        return getRootEditPart().getContents();
    }



    public void setRootEditPart(RootEditPart editpart) {
        assert editpart == null || editpart instanceof ZKRootEditPart;
        super.setRootEditPart(editpart);
        if (!getRootEditPart().isActive())
            getRootEditPart().activate();
    }

}
```

Figure A.13: Source Code of ZKControlViewer.java Part 2


AbstractZKViewer.java

```java
package org.eclipse.pmf.wazaabi.engine.zk.viewers;

import org.eclipse.emf.common.notify.AdapterFactory;

public abstract class AbstractZKViewer extends AbstractWazaabiViewer {

    private final org.zkoss.zk.ui.AbstractComponent parent;

    public static final int BACKGROUND = 8;

    public AbstractZKViewer(org.zkoss.zk.ui.AbstractComponent parent)
    {
        this.parent = parent;
    }

    protected EList<AdapterFactory> createAdapterFactoryList()
    {
        EList<AdapterFactory> adapterFactories = new BasicEList<AdapterFactory>();
        adapterFactories.add(RuntimeLayoutsAdapterFactory.INSTANCE);
        adapterFactories
                .add(org.eclipse.pmf.wazaabi.engine.zk.adapter.widgets.RuntimeWidgetsAdapterFactory.INSTANCE);
        adapterFactories
                .add(org.eclipse.pmf.wazaabi.engine.zk.adapter.resources.RuntimeResourcesAdapterFactory.INSTANCE);
        return adapterFactories;
    }


    protected void hookControl() {
        super.hookControl();
        if (getControl() == null)
            return;
    }
```

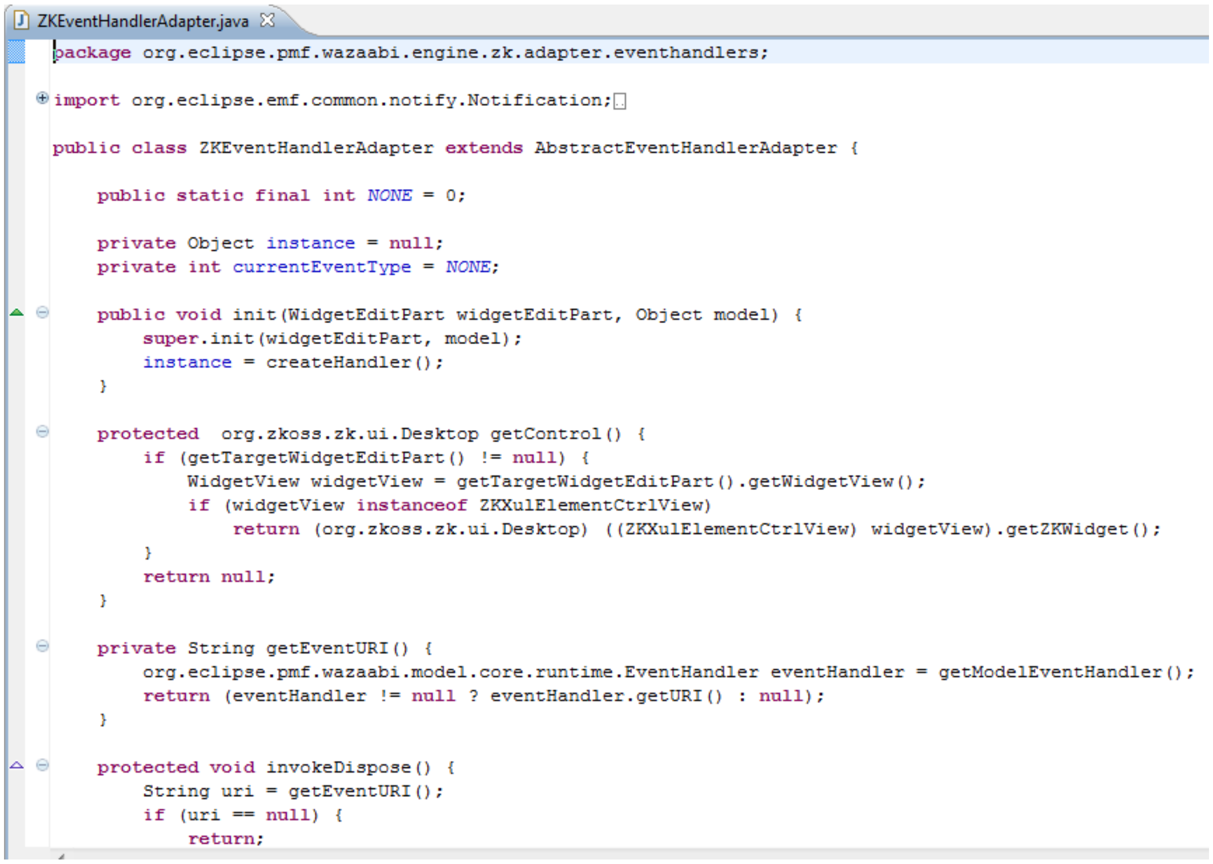
Figure A.14: Source Code of AbstractZKViewer Part 1


AbstractZKViewer.java

```java
    }

    protected EList<AdapterFactory> createAdapterFactoryList()
    {
        EList<AdapterFactory> adapterFactories = new BasicEList<AdapterFactory>();
        adapterFactories.add(RuntimeLayoutsAdapterFactory.INSTANCE);
        adapterFactories
                .add(org.eclipse.pmf.wazaabi.engine.zk.adapter.widgets.RuntimeWidgetsAdapterFactory.INSTANCE);
        adapterFactories
                .add(org.eclipse.pmf.wazaabi.engine.zk.adapter.resources.RuntimeResourcesAdapterFactory.INSTANCE);
        return adapterFactories;
    }


    protected void hookControl() {
        super.hookControl();
        if (getControl() == null)
            return;
    }

    protected void unhookControl() {
        if (getControl() == null)
            return;
        super.unhookControl();
        AbstractWidgetRootEditPart tep = (AbstractWidgetRootEditPart) getRootEditPart();
        tep.setWidgetView(null);
    }

    public org.zkoss.zk.ui.AbstractComponent getParent() {
        return parent;
    }

}
```

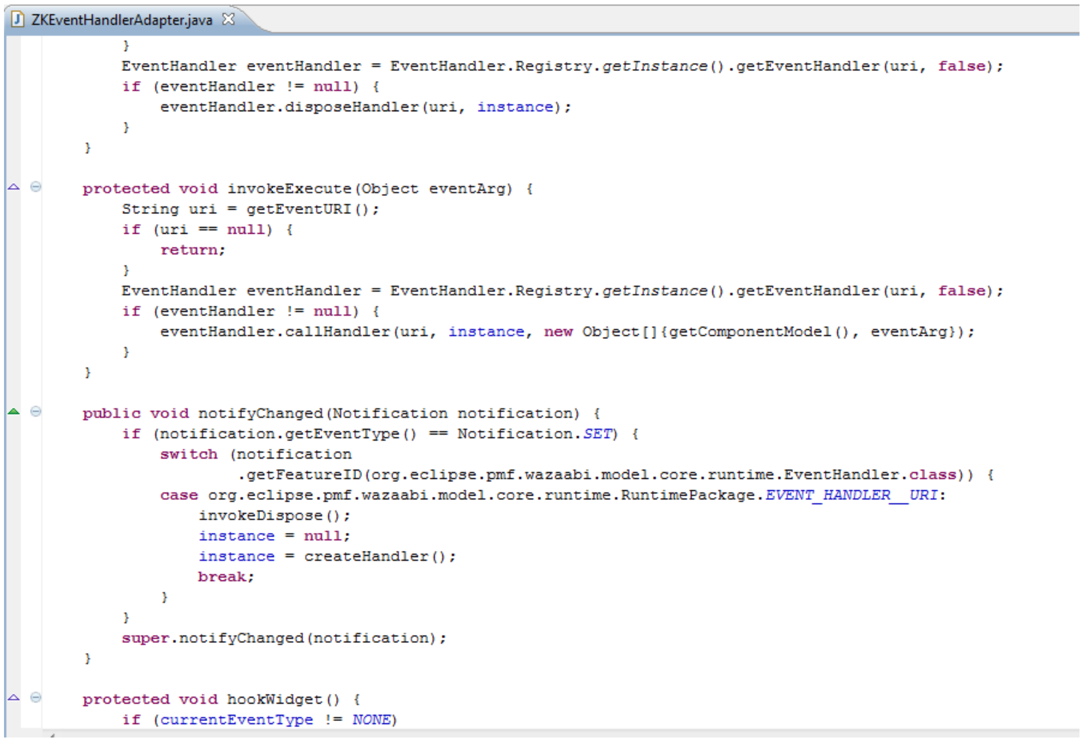
Figure A.15: Source Code of AbstractZKViewer Part 2

A.1.4 Adapters


ZKEventHandlerAdapter.java

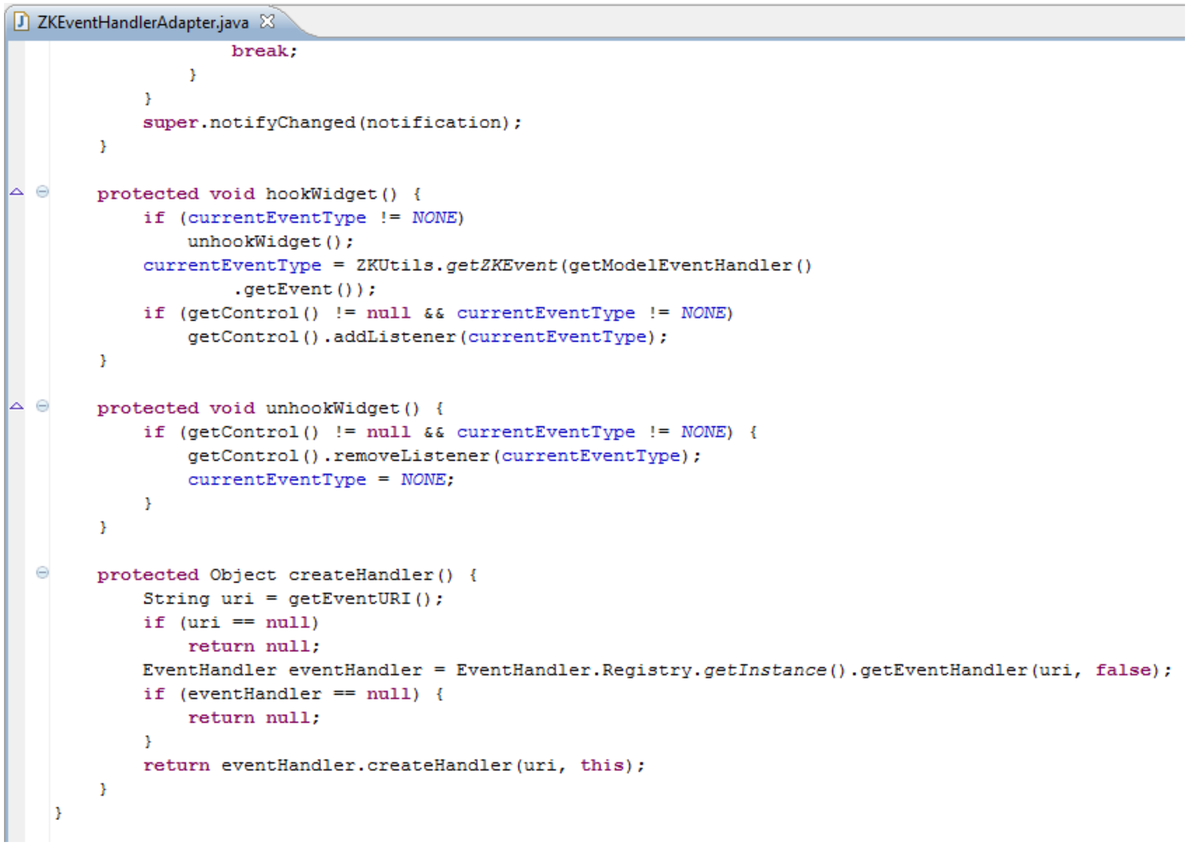
```
package org.eclipse.pmf.wazaabi.engine.zk.adapter.eventhandlers;

import org.eclipse.emf.common.notify.Notification;

public class ZKEventHandlerAdapter extends AbstractEventHandlerAdapter {

    public static final int NONE = 0;

    private Object instance = null;
    private int currentEventType = NONE;

    public void init(WidgetEditPart widgetEditPart, Object model) {
        super.init(widgetEditPart, model);
        instance = createHandler();
    }

    protected  org.zkoss.zk.ui.Desktop getControl() {
        if (getTargetWidgetEditPart() != null) {
            WidgetView widgetView = getTargetWidgetEditPart().getWidgetView();
            if (widgetView instanceof ZKXulElementCtrlView)
                return (org.zkoss.zk.ui.Desktop) ((ZKXulElementCtrlView) widgetView).getZKWidget();
        }
        return null;
    }

    private String getEventURI() {
        org.eclipse.pmf.wazaabi.model.core.runtime.EventHandler eventHandler = getModelEventHandler();
        return (eventHandler != null ? eventHandler.getURI() : null);
    }

    protected void invokeDispose() {
        String uri = getEventURI();
        if (uri == null) {
            return;
```

Figure A.16: Source Code of ZKEventHandlerAdapter.java Part 1



```
ZKEventHandlerAdapter.java
        }
        EventHandler eventHandler = EventHandler.Registry.getInstance().getEventHandler(uri, false);
        if (eventHandler != null) {
            eventHandler.disposeHandler(uri, instance);
        }
    }

    protected void invokeExecute(Object eventArg) {
        String uri = getEventURI();
        if (uri == null) {
            return;
        }
        EventHandler eventHandler = EventHandler.Registry.getInstance().getEventHandler(uri, false);
        if (eventHandler != null) {
            eventHandler.callHandler(uri, instance, new Object[]{getComponentModel(), eventArg});
        }
    }

    public void notifyChanged(Notification notification) {
        if (notification.getEventType() == Notification.SET) {
            switch (notification
                    .getFeatureID(org.eclipse.pmf.wazaabi.model.core.runtime.EventHandler.class)) {
            case org.eclipse.pmf.wazaabi.model.core.runtime.RuntimePackage.EVENT_HANDLER__URI:
                invokeDispose();
                instance = null;
                instance = createHandler();
                break;
            }
        }
        super.notifyChanged(notification);
    }

    protected void hookWidget() {
        if (currentEventType != NONE)
```

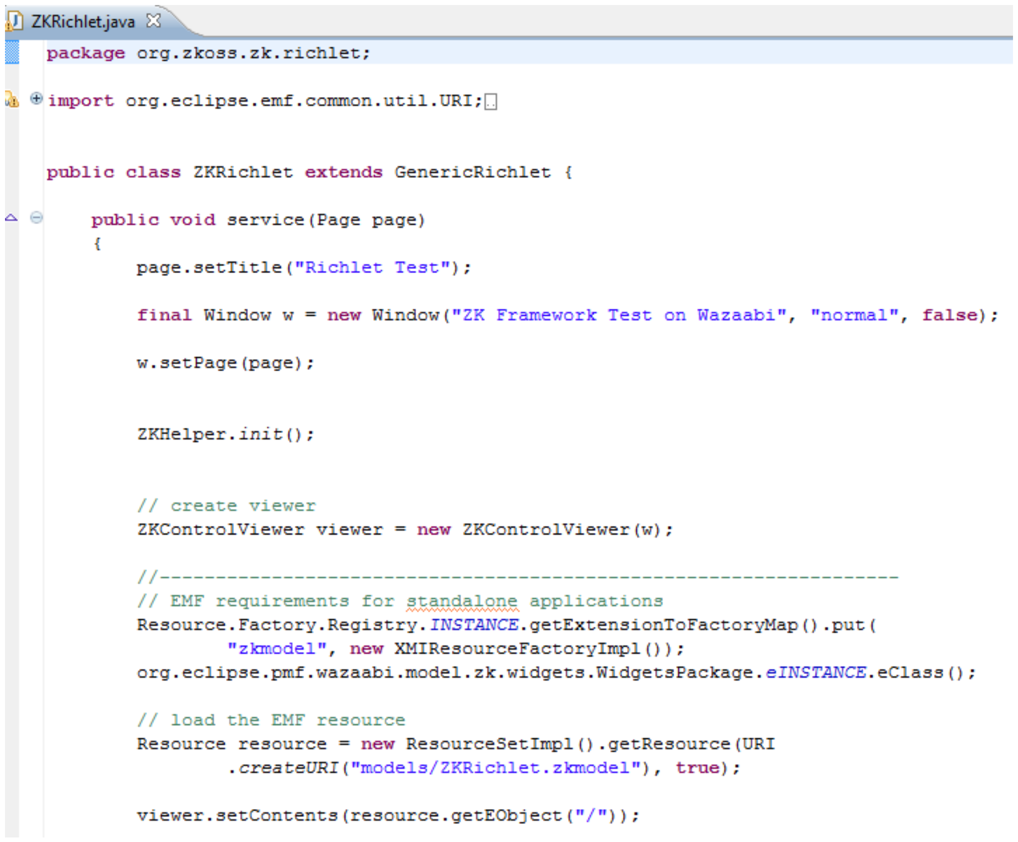
Figure A.17: Source Code of ZKEventHandlerAdapter.java Part 2



```java
                    break;
                }
            }
            super.notifyChanged(notification);
        }

        protected void hookWidget() {
            if (currentEventType != NONE)
                unhookWidget();
            currentEventType = ZKUtils.getZKEvent(getModelEventHandler()
                    .getEvent());
            if (getControl() != null && currentEventType != NONE)
                getControl().addListener(currentEventType);
        }

        protected void unhookWidget() {
            if (getControl() != null && currentEventType != NONE) {
                getControl().removeListener(currentEventType);
                currentEventType = NONE;
            }
        }

        protected Object createHandler() {
            String uri = getEventURI();
            if (uri == null)
                return null;
            EventHandler eventHandler = EventHandler.Registry.getInstance().getEventHandler(uri, false);
            if (eventHandler == null) {
                return null;
            }
            return eventHandler.createHandler(uri, this);
        }
    }
```

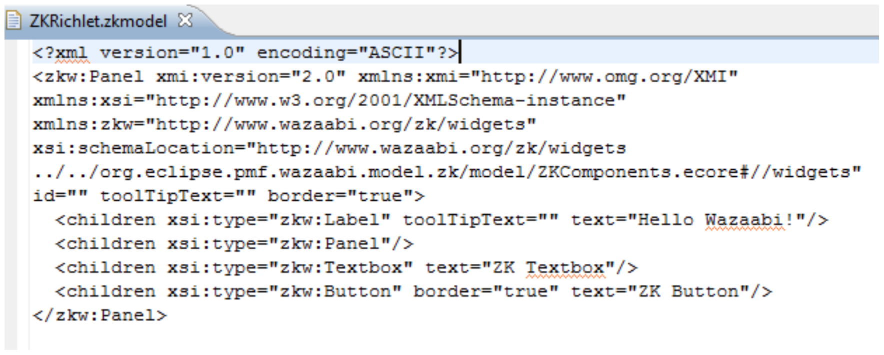
Figure A.18: Source Code of ZKEventHandlerAdapter.java Part 3

A.2 ZK Richlet


ZKRichlet.java

```java
package org.zkoss.zk.richlet;

import org.eclipse.emf.common.util.URI;

public class ZKRichlet extends GenericRichlet {

    public void service(Page page)
    {
        page.setTitle("Richlet Test");

        final Window w = new Window("ZK Framework Test on Wazaabi", "normal", false);

        w.setPage(page);


        ZKHelper.init();


        // create viewer
        ZKControlViewer viewer = new ZKControlViewer(w);

        //-------------------------------------------------------------------
        // EMF requirements for standalone applications
        Resource.Factory.Registry.INSTANCE.getExtensionToFactoryMap().put(
                "zkmodel", new XMIResourceFactoryImpl());
        org.eclipse.pmf.wazaabi.model.zk.widgets.WidgetsPackage.eINSTANCE.eClass();

        // load the EMF resource
        Resource resource = new ResourceSetImpl().getResource(URI
                .createURI("models/ZKRichlet.zkmodel"), true);

        viewer.setContents(resource.getEObject("/"));
```

Figure A.19: Source Code of ZKRichlet.java


ZKRichlet.zkmodel

```xml
<?xml version="1.0" encoding="ASCII"?>
<zkw:Panel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:zkw="http://www.wazaabi.org/zk/widgets"
xsi:schemaLocation="http://www.wazaabi.org/zk/widgets
../../org.eclipse.pmf.wazaabi.model.zk/model/ZKComponents.ecore#//widgets"
id="" toolTipText="" border="true">
  <children xsi:type="zkw:Label" toolTipText="" text="Hello Wazaabi!"/>
  <children xsi:type="zkw:Panel"/>
  <children xsi:type="zkw:Textbox" text="ZK Textbox"/>
  <children xsi:type="zkw:Button" border="true" text="ZK Button"/>
</zkw:Panel>
```

Figure A.20: Source Code of ZKRichlet.zkmodel

Appendix B

Project Partners and Contacts

Following are the project partners and their contacts that were involved in the development of this project.

- **Supervisor:** Hallvard Trætteberg, Associate Professor
  - Office: 115 IT-bygget
  - Tel: 735 93443 / 918 97 263
  - E-Mail: hal@idi.ntnu.no
- **Group Member:** Asim Cihan Erdemli, Master Student
  - E-Mail: erdemli@stud.ntnu.no
- **Group Member:** Onur Hazar, Master Student
  - E-Mail: hazar@stud.ntnu.no

Appendix C

Glossary

CSS

CSS is a cascading style sheet language which express the semantics of a document written in a markup language (HTML, XHTML, XML).

DOM

The Document Object Model is a cross-platform and language-independent tradition for representing and interacting with objects or so-called elements in HTML, XHTML and XML documents in order to address and manipulate them within the syntax of the programming language in use.

DOJO

Dojo is an open source JavaScript toolkit which contains specific JavaScript library for developing JavaScript and Ajax web sites and applications.

DWR

Direct Web Remoting, is a Java open source library which assists software developers to write web pages that include Ajax technology.

Eclipse IDE

Eclipse is a software development environment which has extensible plug-ins supporting different programming languages.

EMF

Eclipse Modeling Framework is a modeling framework for building various applications and tools based on Eclipse.

GWT

GWT is google's open source web toolkit which gives a set of tools that are needed to build different JavaScript applications.

HTML

HyperText Markup Language is a markup language used on websites.

Java

Java is a high level programming language developed by Sun Microsystems as a core component of their platform in 1995.

JavaScript

JavaScript is a dynamic scripting language based on object-oriented and functional programming.

JSON

JavaScript Object Notation is a kind of data interchange format which is a text based open standard designed for human readable data interchange.

LaTeX

LaTeX is a document markup language and document preparation system.

RAP

Rich Ajax Platform is a open source Eclipse project which uses Eclipse and its plugins to build rich internet applications based on Ajax.

RCP

Rich Client Platform is a software consisting of a core lifecycle manager, a standard bundling framework, a portable widget toolkit, file buffers, text handling, text editors, a workbench, a data binding and an update manager.

SCD

A System Context Diagram in software engineering and systems engineering is a diagram that represents the actors outside a system that could interact with that system.

SDK

A software development kit is typically a set of development tools that allows to create applications for a given software package, software framework, hardware platform, computer system, operating system, or similar platform.

SWT

Standard Widget Toolkit is a graphical widget toolkit developed by IBM for the Java platform.

SVN

Subversion (SVN) is a version control system initiated in 2000 by CollabNet Inc. It is used to maintain current and historical versions of files such as source code, web pages, and documentation.

Wazaabi

Wazaabi is a set of eclipse plugins with EMF based models for building parts of an application GUI.

WPD

WPD stands for the Widget Package Descriptor which is a file for describing the information of a package, such as its widget classes and external JavaScript files.

WYSIWYG

Acronym for "What You See Is What You Get". User interface principle which recommends that where the content shown while editing must be exactly the same as the final output.

XHTML

XHTML means eXtensible HyperText Markup Language which is an extended version of HTML for building web pages.

XUL

XUL is a user interface markup language based on XML developed by Mozilla.

XML

XML means Extensible Markup Language which contains rules decided by W3C for encoding documents into meaningful form for machines.

Vaadin

Vaading is an open source Ajax framework for building rich web applications.

ZK

ZK is an alternative open source Ajax framework written in Java that creates graphical user interfaces for rich web applications.