

# CHALMERS



## Interface for a Railway Control System

Master of Science Thesis in the Programme Intelligent Systems Design

Monireh Sanaei

Akbar Abdi Azandaryani

**Report No. 2011:078**

**ISSN: 1651-4769**

Chalmers University of Technology

University of Gothenburg

Department of Applied Information Technology  
Göteborg, Sweden, December 2011





**CHALMERS**



**BOMBARDIER**  
*TRANSPORT*

The Authors grant to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Authors warrant that they are the authors to the Work, and warrant that the Work does not contain text, pictures or other material that violates copyright law.

The Authors shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Authors have signed a copyright agreement with a third party regarding the Work, the Authors warrant hereby that they have obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Interface for a Railway Control System

Monireh Sanaei, Akbar Abdi Azandaryani

© Monireh Sanaei, Akbar Abdi Azandaryani, December 2011.

Examiner: Claes Strannegård

Chalmers University of Technology  
University of Gothenburg  
Department of Computer Science and Engineering  
SE-412 96 Göteborg  
Sweden  
Telephone + 46 (0)31-772 6036

Department of Applied Information Technology  
Göteborg, Sweden December 2011

Chalmers University of Technology

University of Gothenburg

Department of Applied Information Technology  
Göteborg, Sweden, December 2011

# Abstract

Centralized Traffic Control, commonly named CTC, is the most common way nowadays of controlling train traffic. It is based on the principle of centralizing all information concerning railroad tracks over a given area and to control trains running on it by the mean of a single person, the train dispatcher. This technology needs efficient telecommunications as the employee who dispatches trains is not physically present at a station but more commonly at a remote place, namely the CTC office. The CTC retrieves railway signals and gives useful information to the train dispatcher.

Although it was a completely hardware device, CTC is now most of the times a software which summarizes signals, headways and trains to allow the train dispatcher taking good decisions, particularly when some trains stop or have delay. Some of CTC are even partially automatic and can take routine decisions by themselves, letting only critical decisions for human control. It also most of the times prevent human errors by avoiding decisions which could result in train collisions.

This thesis objective is to design and development of sub-system CTC in the software system InterFlow for Bombardier Transportation incorporation. In this context, the CTC is the run time user interface and alarm handling facility for the stationary part of the InterFlow system, subsystem for Centralized Traffic Control.

This thesis describes the process of design and implement a new interface to TCC server in order to use XML formatted messages over a TCP/IP socket, and design and implementation of new graphical user interface and integrate it with existing graphical component which has been developed in Adobe Flex in another thesis project.

This report has been divided into five chapters. The first chapter provides the background and motivation of the thesis. The second chapter describes the evaluation of candidate frameworks considering the requirements in order to choose the most appropriate framework for this application. The third chapter explains the design process of different parts of the application. The fourth chapter explains the implementation. At the end of each section within the chapters the results has been explained. The final chapter provides the conclusion.

# Acknowledgements

We would like to thank all the people who support us during this thesis, Martin Karlsson our supervisor in Bombardier transportation and Olof Torgersson our supervisor for Interaction Design and Claes Strannegård our examiner professor in Chalmers University. Also we would like to thank our families and friends for their kind supports.

# Table of Contents

<b>Abstract.....</b>	<b>8</b>
<b>Acknowledgements.....</b>	<b>9</b>
<b>1 Introduction .....</b>	<b>1</b>
1.1 Bombardier Transport .....	1
1.2 Context and motivation.....	1
1.3 Objectives .....	2
1.4 System Architecture .....	2
<b>2 Framework Evaluation .....</b>	<b>4</b>
2.1 Overview.....	4
2.2 Target Frameworks .....	4
2.2.1 Java.....	4
2.2.2 C#.NET.....	6
2.2.3 Power Builder .....	9
2.3 Requirements Analysis.....	11
2.3.1 Requirements Specification .....	11
2.3.2 Support for Local Languages .....	15
2.3.3 Socket Communication .....	16
2.3.4 XML Parsing.....	17
2.3.5 Database Connectivity.....	18
2.3.6 Integration with Adobe Flex .....	19
2.3.7 Portability.....	21
2.3.8 Ease of Use for Developer .....	21
2.3.9 Widely Common in Industry .....	22
2.3.10 Maturity .....	22
2.3.11 Maintainability in Future .....	22
2.4 Conclusion.....	23
<b>3 Application Design .....</b>	<b>27</b>
3.1 Overview.....	27
3.2 Model-View-Controller Design Pattern.....	27
3.2.1 Model.....	28

3.2.2	View .....	28
3.2.3	Controller .....	28
3.3	TCC Interface.....	29
3.3.1	Communication Protocol.....	30
3.3.2	Method.....	32
3.3.3	Results .....	38
3.4	Application Logic.....	39
3.5	Graphical User Interface .....	40
3.5.1	Design Theories and Methodologies.....	40
3.5.2	Method.....	43
3.5.3	Results .....	45
<b>4</b>	<b>Implementation.....</b>	<b>46</b>
4.1	Overview.....	46
4.2	TCC Interface.....	46
4.2.1	Protocol Handling .....	47
4.3	Application Logic.....	47
4.3.1	Model.....	47
4.4	Graphical User Interface .....	49
4.4.1	Functional Requirements.....	49
4.4.2	Results .....	50
<b>5</b>	<b>Conclusion.....</b>	<b>63</b>
	<b>Terminology.....</b>	<b>65</b>
	<b>References.....</b>	<b>66</b>
	<b>Appendix .....</b>	<b>70</b>

# 1 Introduction

This chapter includes an overview of the thesis, background, context at the company and objectives of the thesis as well as the method and limitation.

## 1.1 Bombardier Transport

Bombardier is a global transportation company with engineering and producing sites in 23 countries and worldwide service sites. This company is providing worldwide leadership in Aerospace and Rail Transportation. Bombardier Transportation (BT) is a worldwide leading supplier of equipment to railways. The Rail Control Solutions (RCS) division within BT develops, installs and maintains system solutions for rail traffic control and supervise, i.e. signaling systems. The Gothenburg office of BT/RCS is specialized in systems based on radio communication, a leading edge technology within the rail industry which reduces the need for physical installations substantially compared to traditional systems.

## 1.2 Context and motivation

Bombardier's solution contains many components, one of them is CMI, Controller Machine Interface which is the interface to the system for dispatcher to control and supervise the traffic. Its main feature is a graphic representation of the track layout, in which positions of trains and states of objects are continuously updated. The dispatcher can get access to command menus by clicking relevant objects in the layout.

The CMI is often subject to specific customer requirements, tradition and/or system environment imposes different solutions, which may be provided by the customer, another unit within Bombardier, or a third party. But there also exist a locally developed alternative, branded EBIScreen 1400, which can be supplied to customers as a fall back solution to keep the system operational in case of failure of the main CMI. The EBIScreen 1400 has not been part of any of our projects for several years, and is in many respects based on obsolete technology.

The object of the thesis project is to transfer this product to a modern environment, using state-of-the-art technologies. This transfer requires design and implements new interface to Train Control Center or TCC and also new graphical user interface which can be integrated with available graphical component representing the Track Layout.



## 1.3 Objectives

The first objective of this thesis is to evaluate candidate technologies considering the requirements to observe which development environment fits better with the requirements.

The second objective is to design the application, including design of the new interface to TCC according to the available communication protocol using XML formatted messages, as well as design of new graphical user interface considering the available graphical user interface with the aim of improving it in order to make it more efficient and user friendly. Also design the rest of the application's logic which was the data and logical parts to handle the logic of the application in connection with the two interfaces, TCC interface and graphical user interface.

The third objective is to implement different parts of the application including logic, TCC interface and graphical user interface using the suitable framework.

## 1.4 System Architecture

The CTC is the run time user interface and alarm handling facility for the stationary part of the InterFlow system. CTC provides the train dispatcher with means of controlling and supervising the system.

CTC communicates with RBC for all train control functions. Alarms are read from and manipulated in the database, where they may be placed by any subsystem.

The CTC subsystem consists of two main functions, Run time MMI for the InterFlow stationary system and Event logging facility. The MMI is a client process, using the RBC subsystem as a server.

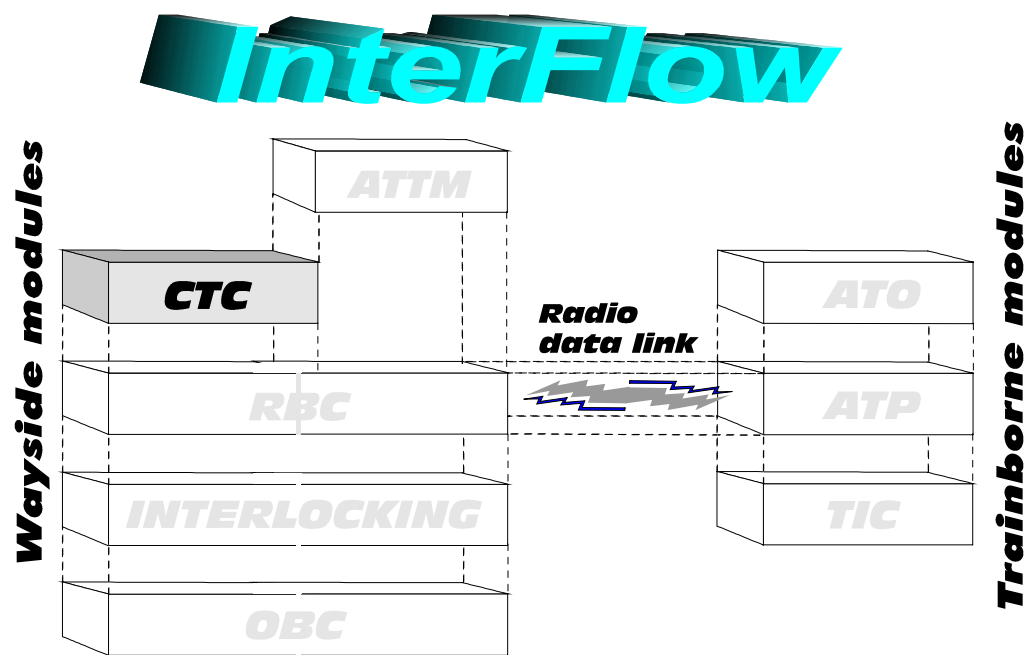


Figure1. System Architecture

## 2 Framework Evaluation

### 2.1 Overview

This chapter as explained earlier in the report is dedicated to the evaluation of frameworks. According to the first objective of the thesis there was an evaluation on different candidates for frameworks. The requirements to be considered includes capability to meet the application requirements specification, support for local languages, support for socket communication, support for XML parsing, support for database connectivity (MYSQL), integration with available graphical component, portability, ease of use for developer, being widely common in industry, maturity and finally maintainability in future.

### 2.2 Target Frameworks

Main candidates for frameworks were Java and C sharp, besides the upgraded version of PowerBuilder should be considered as well. In order to do the evaluation, first we provide an overall description for each of these candidate frameworks considering their advantages and drawbacks, and then provide analysis of the each framework remarking the framework requirement for this thesis.

#### 2.2.1 Java

Java platform and language has been started in December 1990 by Sun Microsystems and for the first time it was released in 1994 and the current version is Java SE7 released in July 2011. The target was as an alternative to available programming languages such as C and C++ languages, while being object oriented language, get rid of memory managing issues and being portable cross different platforms. <sup>1</sup>

Java programming language is a general-purpose language and it gives the possibility to write code on wide range of areas from a piece of code for mobile phones to enterprise computer network applications, “expecting low level code which deals directly with underlying hardware.” <sup>2</sup>By the way it provides some mechanisms to perform platform-dependent tasks, although Java is not made for this purpose. <sup>2</sup>

There are several advantages which make Java one of the most common used programming technologies in industry which some of the most important one are explained here.

It is an object oriented language in which application components are treated as objects. The developers need to understand the concept of object that is quite straight forward. Then can start coding by creating objects and manipulating them and reuse the objects in different pieces of code. Regarding the fact that The Java core libraries provide programmers a well-designed and intuitive set of APIs containing well-known set of classes with proper set of methods to manipulate common objects and perform common tasks on them.

The other major advantage of Java is the portability. Put another way, Java is on a higher level of abstraction in compare to many other languages in order to make it portable regardless of underlying platform. This property enables the developers to create software application once and run it on several different environments regardless of the platform. The portability has been possible using software called Java Virtual Machine or JVM which intermediates between Java program and underlying platform. It comes up with some disadvantages as well for instance lower speed of execution although it has been improved by introducing Just-in-time compiler later on. <sup>2</sup>

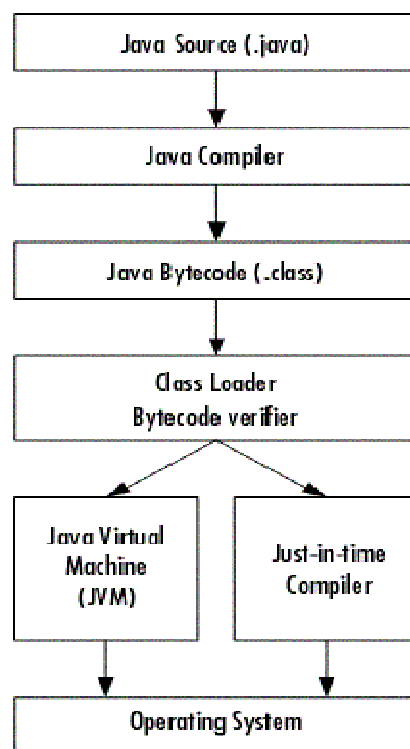


Figure2. Java Code Cycle <sup>5</sup>

The other major facility providing by Java for the developers is automatic memory management using the Garbage Collector. It frees the memory from the created object when they are no longer used which is a tough and error-prone task for developers in some other languages like C or C++. Java is secure and highly reliable in the sense of error detection providing in compile time and also try-catch facility in run time to help the developer to point out the faults. These properties make Java an easy and elegant language for programmers to be more efficient and productive, compare to other programming languages.<sup>2</sup>

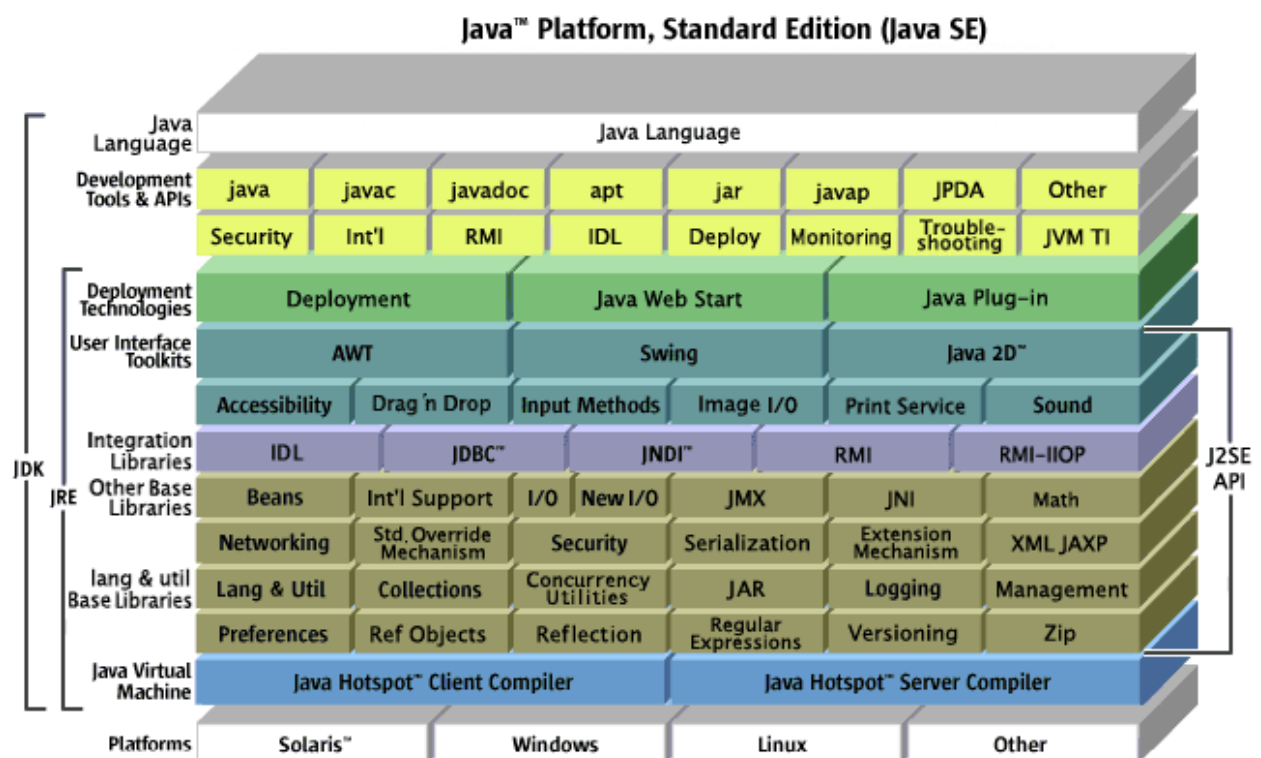


Figure3. Java Architecture <sup>5</sup>

## 2.2.2 C#.NET

C Sharp or C# programming language had been started in January 1999 by Microsoft for .NET platform and for the first time published in 2000. The current version is C# 4.0 version released in April 2010.

C# is common to be a multi-paradigm programming language, covering the concepts of object oriented, functional, component oriented paradigms and etc. C# is general-purpose language which suited for developing a wide variety of robust applications for .NET platform from Windows-based to Web-based applications, particularly suits to develop software piece to use in distributed systems. <sup>4</sup>

.NET Framework is a software component running on Microsoft Windows. This framework includes two main components; a virtual execution system called the Common Language Runtime (CLR) and a set of class libraries. Common Language Runtime is in charge of handling the code execution issues such as memory management, threading, exception handing and type safety. And the class libraries which are organized in to namespaces are containing a comprehensive set of predefined useful classes that enable developers to develop all kinds of applications ranging from Consol to Web application. <sup>3</sup>

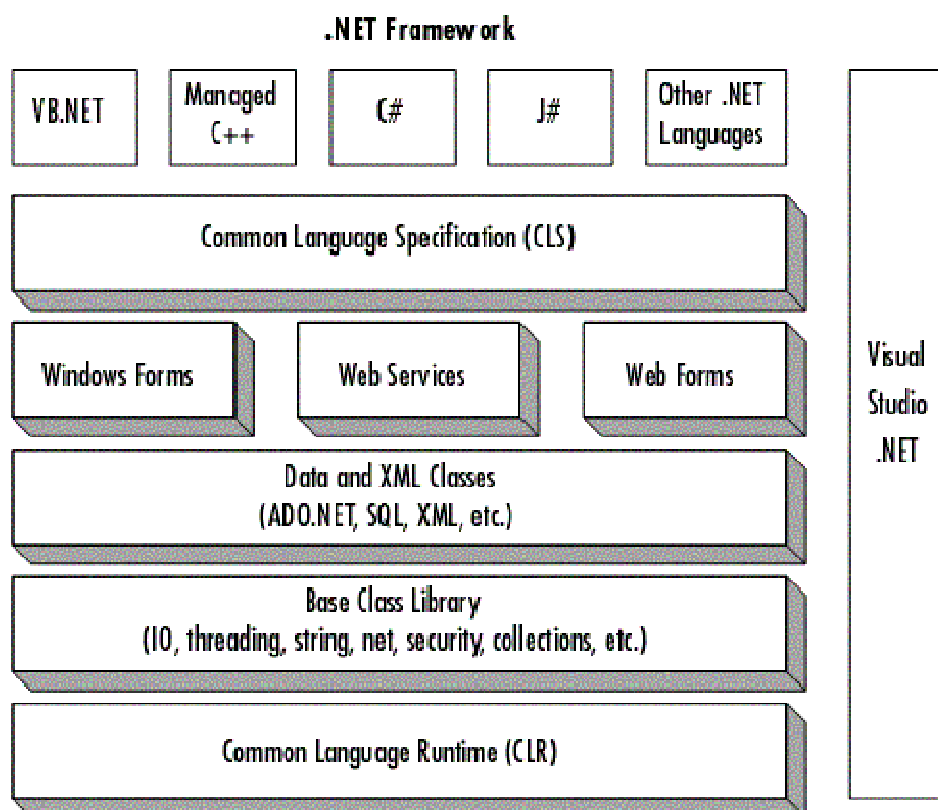
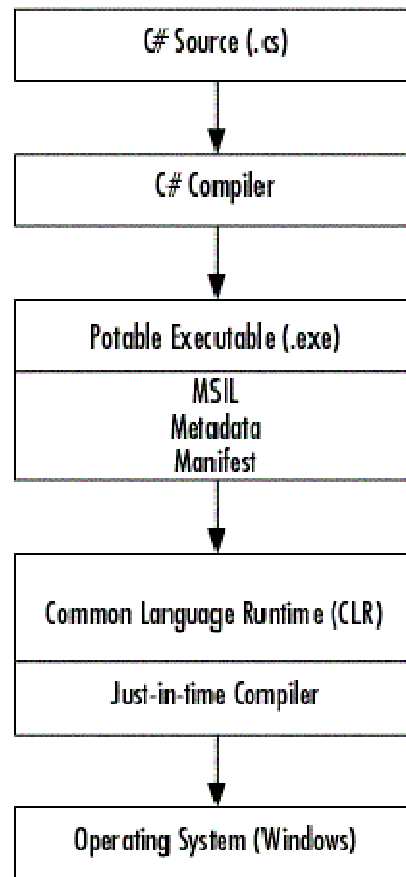


Figure4. .NET Framework platform Architecture <sup>5</sup>

C# is a modern programming language which among the other programming languages is mostly compared with Java. The reason of this comparison is for having many common features similar to Java programming language. But since C# has been invented later than java, the creators have inspired from both the strengths and weaknesses of Java.

C# is object oriented programming language with strong type checking. Therefore developers can take advantage of creating reusable codes. One of the important design goals of C# is to support internationalization. It supports automatic memory management using Garbage Collector as well. <sup>6</sup>

Despite the fact that C# source code is converted to an intermediate code called Microsoft Intermediate Language or MSIL which in runtime will be converted to system-purposed code, but this intermediate code currently only runs on few operating systems which the main one is Microsoft Windows platform. Therefore in the sense of portability Java is preferred than C#.NET since it supported on more operating systems.<sup>4</sup>



Figurer5. C# Code Cycle <sup>5</sup>

### 2.2.3 Power Builder

Power Builder was originally created by Powersoft in 1991. The first version was released in July 1991. Powersoft later on acquired by Sybase. The current version is 12.5 but the major upgrade happened in version 12.0 released on April 2010. <sup>7</sup>

PowerBuilder encompasses a scripting language called PowerScript, which is used for application event handling. PowerBuilder includes a Foundation Class library (PFC) which is based on object oriented design and contains a set of PowerBuilder libraries (PBLs) including objects written in PowerBuilder. These libraries enable the developers to create class libraries or applications by customizing them. <sup>8</sup>



### Original Application Architecture |

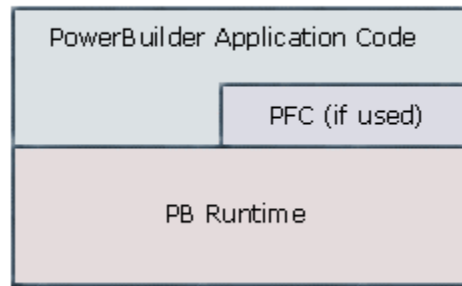


Figure6. PowerBuilder Application Architecture <sup>9</sup>

PowerBuilder is an object oriented enable developer to develop reusable, robust and maintainable codes. PowerBuilder is also event driven which events correspond to user actions, such as clicking on components of the user interface, for instance buttons or menu items. PowerBuilder can be used to develop desktop, Web, Rich Internet Applications (RIA) or Windows Presentation Foundation (WPF) applications, particularly suitable for developing business applications handling large amount of data or including Graphical User Interface. <sup>11</sup>

The most key component of PowerBuilder programming language is DataWindow object which enables developers to specify and control user interface appearance and behavior. This object also decreases the complexity of data access and manipulation by facilitating the programmers to create, edit and display data from the database. <sup>11</sup>

PowerBuilder is known to use Rapid application development (RAD) software development methodology which consumes minimum time on planning in order to starts developing the software rapidly. So it is quick and easy to develop. PowerBuilder also has a high level of abstraction hiding programming complexities. Since PowerBuilder version 12.0, in order to keep it in line among other available competitors, there have been substantial efforts to make it back to the market. As a major example of these efforts, one can point to provide the strategy to make PowerScript compliant with common language specification (CLS) of .NET framework which started in a version 9.0 and continued until current version. This facility enables developers to develop applications that performs on several different platforms provided by .NET Framework, including ASP.NET Web Forms, Windows Forms, or Windows Presentation Foundation or WPF. <sup>11</sup>

Since PowerBuilder 12.0, this strategy can make PowerScript language comparable with C#, Java or VB.NET, in the sense of additional programming features supported by that

such as Arrays, Delegates, Parameterized Constructors, User-defined Enumerations and Generics. In the current version PowerBuilder 12.5 adds support for some more new features such as Multithreading. It has less flexibility, performance and more limitations comparing for instance Java, by the way in current version of PB they overcame the extensibility issue somehow.<sup>10</sup>

## **2.3 Requirements Analysis**

According to programming experiences, it is difficult to conclude which language is the best and which one is the worst, because it pretty much depends on type of the target application. For instance some experiences show that in the case of Graphical User Interface development, C# Windows Forms are a lot better than Java Swing/AWT, because it is faster to develop, the applications run faster and look better while for developing Web application Java is great particularly on Linux servers.

In the following section this report takes a look at the requirements of this thesis and then provides analysis of candidate frameworks' capability to fulfill the requirements.

### **2.3.1 Requirements Specification**

The requirements specification for the CMI application was the old version that has been used for the earlier versions of the application. This specification mostly proceeds with the CMI graphical user interface requirements and provides very detailed description of interface components. After precise studies and considering the limited time, some parts of it ignored through some revision for this thesis.

Graphical User Interface of the CMI is expected to be designed as Multiple Document Interface or MDI. MDI is a Microsoft Windows programming user interface which enables user to work with multiple documents at the same time. MDI encompasses a parent window used as back or desktop window which can embed arbitrary number of child windows, secondary windows can be added to provide additional information as well. Through each window user can access different data through either menu bar or tool bar or both of them and viewing some status regarding window via status bar.

Almost all graphical user interface APIs provide some components proper to develop and manipulate MDI user interface. In the following part we provide an analysis of available facility provided by the candidate frameworks to fulfill the requirement specification.

Java Swing toolkit provides some components to implement multiple-document interfaces, including `JDesktopPane` component proper to be used as the parent window, `JInternalFrame` class as child windows and `JDialog` component as secondary windows. Classes `JDesktopPane` and `JInternalFrame` provide many methods for handling child windows.

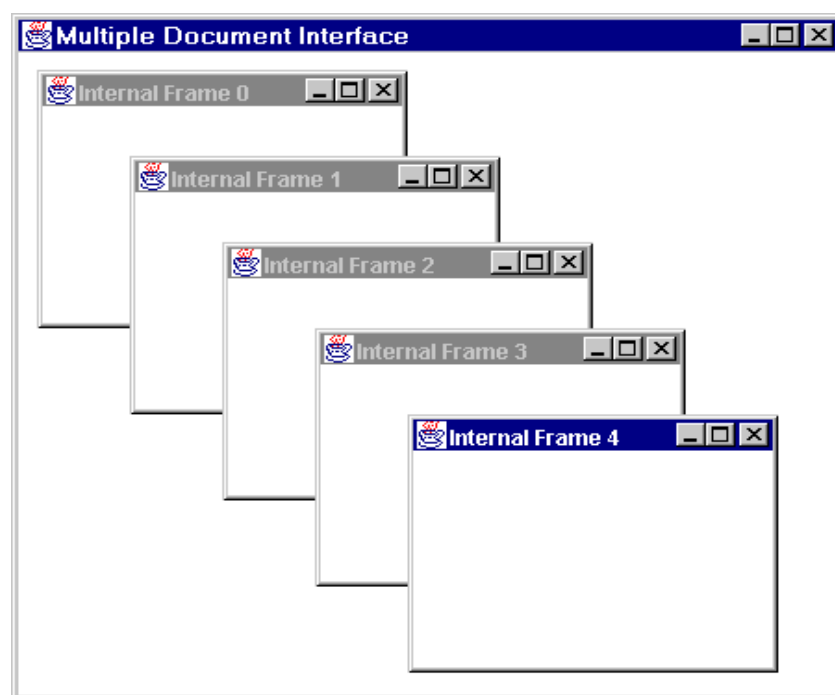


Figure7. Java Multiple Document Interface Design <sup>12</sup>

Here we explain a simple piece of Java code to indicate syntax for implementing multiple-document interface.

To create MDI Parent window;

```
JFrame parentWindow = new JFrame ();  
JDesktopPane desktop = new JDesktopPane ();  
parentWindow.add(desktop);
```

And To create MDI Child windows;

```
JInternalFrame childWindow = new JInternalFrame ();  
parentWindow.add(childWindow); 12
```

C#.NET programming language provides several properties through its graphical interface component, Form, to create and manipulate multiple-document interfaces. As we see in the following piece of code the C# solution is even shorter and straighter forward than Java.

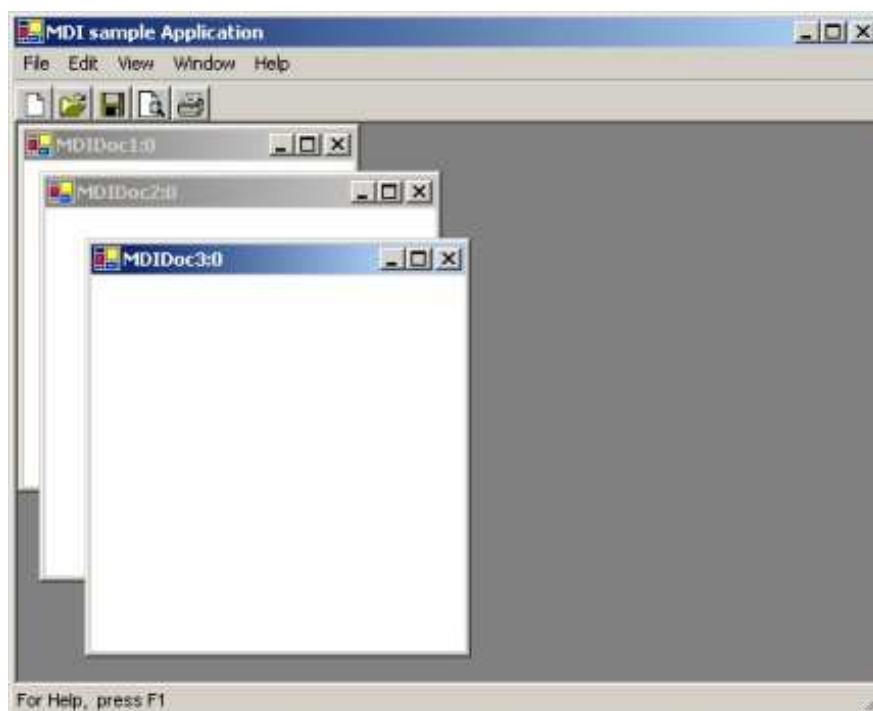


Figure8. C# Multiple Document Interface Design <sup>14</sup>

To create MDI Parent window;

```
Form parentWindow = new Form ();  
parentWindow.IsMDIContainer = true;
```

And To create MDI Child windows;

```
Form childWindow = new Form();
```

```
childWindow.MDIParent      =  
parentWindow;
```

And manipulating child windows for instance keep them focused, is performed using ActiveForm property of an MDI Form.<sup>13, 14</sup>

In PowerBuilder, to build multiple-document interfaces, either MDI Frame or MDI Frame with Microhelp component are used as main or parent window.

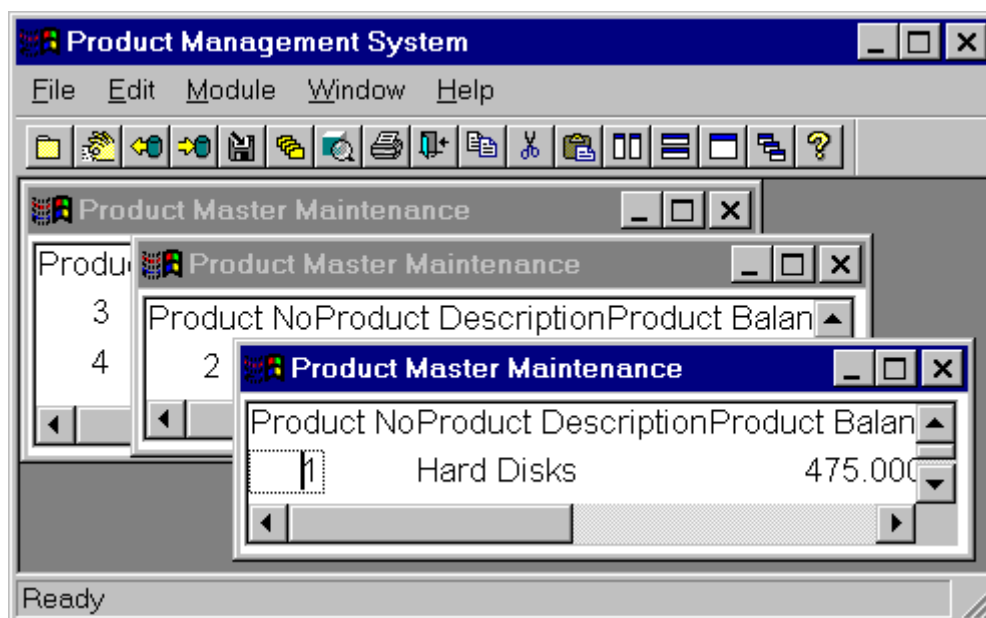


Figure9. PowerBuilder MDI Application <sup>15</sup>

The MDI frame window consists of three parts, Frame, Client Area and Sheets. The Frame consists of components including menu bar, toolbar, window title and the status bar to display Microhelp, a brief description of the current menu item or current activity. Client Area is the area between MDI Frame title bar and the Microhelp status bar in which the sheets are embedded. Sheets correspond to the child windows to perform different user activities. These child windows are opened within the client area, using `OpenSheet()` and `OpenSheetWithParm()` functions provided by PowerBuilder. <sup>15</sup>

### **2.3.2 Support for Local Languages**

The CMI application is often subject to specific customer requirements. One of these requirements is the capability to adapt the application to different languages according to customers around the world.

This process of adapting application is called Internationalization. In order to internationalize an application, it required to be designed in such a way that enables the user to change the language according to region, easily and quickly without software engineering changes. The process of customizing the application to specific locale is called Localization which is accomplished using locale and culture specific constituents such as translated text, fonts, dates and currencies. Since the software applications which only support English are getting old-fashioned, state-of-the-art software technologies support internationalization.

Java SE platform fully supports internationalization within its libraries facilitating language or culture-specific functionalities. A Java internationalization facility enables developers to fast and easy development of multi-lingual applications. For instance in `java.text` package, `MessageFormat` class provides local specific languages, or `SimpleDateFormat` class supports calendar specific eras and date formats for calendar systems different than Gregorian. <sup>16</sup>

In .NET framework this language adaptation capability is called Globalization of which both concept of Internationalization and Localization as explained above, are considered as two aspects.

In C# `System.Globalization` namespace is dedicated to Globalization using `CultureInfo` class. `CultureInfo` is the main class providing a set of different properties and methods to customize the application according to specific cultures. <sup>17</sup>

PowerBuilder internationalization support started with a translation toolkit in version 6.5. The Translation Toolkit is set of tools made by Sybase to localize PowerBuilder

applications. It supports language and culture specific data display for several different languages such as French, German, Italian, Spanish, Dutch, Danish, Norwegian, and Swedish using localized runtime files. After that by PowerBuilder 10 more complete set of multi-lingual features has been added through Unicode support enabling developers to build internationalized application.<sup>18</sup>

### **2.3.3 Socket Communication**

One of the main purposes of the CMI application is to connect to TCC server in order to exchange data over TCP/IP socket. TCP/IP is complement suite of the Transmission Control Protocol and Internet Protocol, providing a reliable ordered delivery of data between computer applications on internet. It brings the requirement for the target framework to support TCP/IP socket, application end-point for the communication over internet using TCP/IP protocols.

Today in a world of connected computers is becoming more and more difficult to avoid the need of communication in software applications. Therefore it is necessary to every common programming language provides an API for socket communication.

Java facilitates socket communication through java.net package in which ServerSocket and Socket classes are providing properties and methods to access and manipulate socket for server and client applications, respectively. ServerSocket functionality is to wait for client request over the network. Upon receiving such a request will process the request by performing required operations and providing responses. The Socket functionality is to try to initialize the communication by sending connection request to server and then receives its response and so on.<sup>19</sup>

C# programming language provides the socket communication through System.Net and System.Net.Sockets namespaces, Socket class provides methods to create client and server sides' applications. Server program is listening to clients request using an instance of TCPListener class and client program connects to server using an instance of TCPClient class.<sup>20</sup>

PowerBuilder provides socket communication through PowerSocket library, TCP/IP Toolkit for PowerBuilder. The available source of this library is old and for 12 years ago, so there is the probability that is no longer used. There is other solution, SocketWrench, a software tool for network programming on Microsoft Windows platforms which can be used with a range of programming languages including PowerBuilder.<sup>21, 22</sup>

### 2.3.4 XML Parsing

As explained earlier, one of the objectives of this thesis is to design and implement the communication protocol between CMI application and TCC server. The communication protocol is an XML-based protocol or using XML-formatted messages. It requires that both sides of the communication be able to do XML parsing.

Java provides several different libraries for applications in order to process XML documents or messages, such as SAX; the Simple API for XML, DOM; the Document Object Model API from W3C, XSLT; the XML Style Sheet Language Transformations from W3C and etc. The most common of them are SAX and DOM. DOM parser creates a tree structure of the XML source document in memory which enables random access to the arbitrary nodes of the DOM tree. SAX parser works with event-driven fashion, tokens of XML document are caught and thrown by the parser as events to be handled.<sup>23</sup>

The .NET framework provides a wide variety of API options for reading and writing XML documents which may offer different efficiency and productivity. To process XML documents, C#.NET programming interface provides System.Xml namespace which is built on key XML industry standards, such as DOM, XPath, XSLT, XML Schemas (XSD) and etc. According to .NET framework, processing XML document is including several steps or layers, as shown in the following figure.<sup>24</sup>

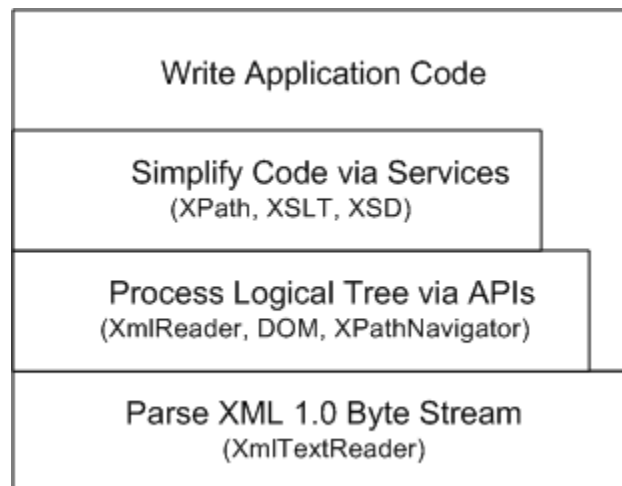


Figure10. XML Processing Layers in .NET<sup>24</sup>



XmlTextReader provides XML 1.0 byte stream functionality which is the most basic step in processing XML documents. After this step, the higher layers are presenting XML APIs choices in order to process the XML documents by treating them as logical tree structures. The APIs performs with two main strategies, streaming and traversal-oriented. The most common API for streaming strategy is SAX, but because of the difficulties with SAX-based XML processing, it has been alternated by Microsoft with simpler and more intuitive streaming API through the XmlReader class library. XML schema and DTD validation facility is also provided through XmlValidatingReader class, which can be used with XmlReader implementation including XmlTextReader. Provided APIs for traversal-oriented strategy by .NET frameworks are including Document Object Model or DOM and XPathNavigator. DOM is the most common traversal-oriented API and is provided through the XmlNode class hierarchy. XPathNavigator API gives the possibility to traverse XML logical tree using a cursor model.<sup>24</sup>

The API provided by PowerBuilder to process XML documents is called PBDOM. This API is quite similar to DOM library by the World Wide Web Consortium or W3C and also to Java DOM or JDOM. It treats XML document based on tree model which can be traversed and manipulated through provided methods within PowerScript code.<sup>25</sup>

### **2.3.5 Database Connectivity**

Besides providing the user interface, another facility of the CMI application is Alarm Handling. Alarms are read from and manipulated in the database, where they may be placed by any subsystem. In order to accomplish this facility the CMI application target framework is required to support database connectivity for today common databases particularly MySQL.

In Java platform, database connection facility is provided by Java Database Connectivity or JDBC API through java.sql and javax.sql packages. JDBC makes it possible for Java programs to virtually connect to any database such as MySQL, Oracle, postgresql, JavaDB and etc, and then manipulates data using SQL queries. The only challenge left is to connect with appropriate connection string.<sup>26</sup>

In C# applications, database connectivity is performed by a set of components called ADO.NET provided by Microsoft.NET framework. ADO.NET classes are contained in System.data namespace, including System.Data.SqlClient, System.Data.Odbc, System.Data.OleDb and System.Data.Oracle which are used to connect to different databases. System.Data.Odbc class is used to communicate with the Sql Server database, System.Data.SqlClient class is used to connect and manipulate MySQL

databases, System.Data.OleDb class is used to perform operations on the Access Database and System.Data.Oracle class is used to perform operations on the Oracle database. Another requirement for C# applications to connect to mysql database is a small program called mysql connector .net which can be found in MySQL official website.

27

PowerBuilder can connect to most of major databases such as Sybase, IBM, Microsoft and Oracle using its native driver which makes it faster and more efficient compares using external drivers for instance Open Database Connectivity or ODBC. By the way it also supports connectivity using ODBC, Java Database Connectivity (JDBC), Object Linking and Embedding, Database (OLE-DB) and ActiveX Data Object for .NET (ADO.NET) within Enterprise applications. <sup>11</sup>

### **2.3.6 Integration with Adobe Flex**

As mentioned before, one of the major parts of the graphical user interface which presents the Track Layout has been developed in another project using Adobe Flex technology. In this thesis is has to be integrated with rest of the CMI application user interface, therefore it requires to consider the possibility and circumstances of integration of Adobe Flex with the candidate frameworks. According to the structure of graphical user interface of CMI application, integration should be carried out in such a manner that embeds the Track Layout within the MDI interface. To consider the feasibility and quality of this integration, first of all we provide an over view of Adobe Flex technology and then proceed with the possible solutions to integrate it with each candidate framework.

Adobe Flex is a free and open-source Software Development Kit or SDK by Adobe Systems. The framework consists of a library of ActionScript classes and components including wide variety of user interface controls enabling developers to build applications containing user interface. It provides the option to execute graphical user interface on the browser for Web applications using Adobe Flash Player, or on the desktop using Adobe AIR.

The most common applications that are taking advantage of integrating Java with Adobe Flex are Java server-side applications using Java Server Pages or JSP integrating with Adobe Flex (desktop) as client side. The reason can be found in difficulties of developing Java client-side applications that can be solved through Adobe Flex. For instance the process of GUI development for client-side using Adobe Flex is more flexible and less

complicated than Swing development and would lead to more interactive and appealing user interface.

Applications integrating Adobe Flex with Java can be a great solution for Rich Internet Application or RIA while taking advantage of strong object-oriented principles of Java such as abstraction, polymorphism, inheritance and etc, brings a highly interactive and appealing graphical user interface for Java server applications. Today this solution seems becoming more common, particularly with the help of available tools providing easier way for development of Java and Flex together, such as the Eclipse integrated development environment plug-in for Flex development.<sup>28</sup> None of these solutions can be used to embed Flex component in Java component. There are some libraries and projects that are using this solution, such as The DJ Project, EasyJCom and JFlashPlayer libraries. The NativeSwing library of DJ Project “allows an easy integration of some native components into Swing applications, and provides some native utilities to enhance Swing's APIs”.<sup>29</sup>

EasyJCom and JFlashPlayer libraries enable the developers to embed swf into JFrame. The problem with these libraries is that none of them are cross-platforms. EasyJCom enables Java Swing to access COM/ActiveX components by embedding them. It requires to built the Flash COM object (Flash.ocx)'s JNI DLL with EZ JCom using the package name flashswf.<sup>30</sup> JFlashPlayer is a Flash Player API Java package that enables Java applications to play and interact with Adobe™ Flash Player movies. Using this API enables to Call Java methods from Flash and ActionScript functions from Java.<sup>31</sup>

Available solutions in order to integrate .NET and Flex frameworks are based on the communication between the libraries of the two frameworks. Put another way, according to the integration answer for this application we considered the solutions to embed the Flex component within Windows Form C# application.

There is a library called External API within ActionScript language which enables the developers to interact with the container application. The container can be a web browser or a desktop application, such as C# Windows Form. A prerequisite is to install Flash Player to run the ActionScript compiled code or SWF file. Then the host application only requires embedding SWF within Windows Form application and then can sending and receiving data using the available methods within the API.<sup>32</sup> There is another solution to integrate C# and Flex which is also based on the communication between the frameworks libraries. This solution is used for client-server integration, through remoting APIs, WebORB in .NET server side communicating with RemoteObject API in ActionScript client side.<sup>33</sup>

PowerBuilder enables the developers to embed Internet Explorer control within windows in the applications. This may be probably solution in the case that Internet Explorer is able to load Flex executable or SWF files.<sup>34</sup>

### **2.3.7 Portability**

As mentioned earlier, one of the most significant advantages of Java is portability, provided by its virtual machine which takes care of the complexities between Java applications and underlying operating system regarding to the type of operating system. C# source code converts to a common intermediate language, CIL or MSIL which can be performed on platforms supporting Common Language Infrastructure, such as the .NET runtime on Windows, or the cross- platform using Mono runtime software.

PowerBuilder runs on different versions of Microsoft Windows platforms. According to PowerBuilder 12.0 which is the latest stable version of PowerBuilder, it supports Windows 7 Professional 32-bit platform, Windows XP (SP 3), Windows XP Tablet PC (SP 3), Windows Server 2003 (SP 2), and Windows Vista (SP 2). PowerBuilder 12.0 maintains support for deployment to Windows Server 2008 (SP 2). It also still supports for deployment to Windows Server 2008 (SP 2), but no longer supports deployment to Windows 2000.<sup>35</sup>

### **2.3.8 Ease of Use for Developer**

Providing facilities such as object oriented features, automatic memory management, intuitive set of well-designed libraries, reusable codes, high level of abstraction hiding the platform-dependency complexities besides plenty of available resources makes Java programming language an elegant and easy target for developers in order to be more efficient and productive.

C# programming language is an object-oriented language which compasses a bit more complex features than Java. It is facilitating automatic memory management, high level of abstraction, intuitive and comprehensive set of libraries, very expressive syntax which makes it easy to learn and improves the developers' productivity.<sup>36</sup>

PowerBuilder is following object oriented paradigm with high level of abstraction. It is taking advantage of Rapid application development (RAD) software development methodology provides a quick and easy to develop applications.

### **2.3.9 Widely Common in Industry**

According to the ranking provided by The TIOBE Programming Community index in October 2011, Java is the most popular programming language with more than 17% usage and C# with more than 6% is in the fifth rank among 20 first most popular languages and Power Builder doesn't exist even among first 50 programming languages. It is remarkable that this ranking is based on the number of skilled engineers worldwide, courses and third party vendors.<sup>36</sup>

We pointed out the efforts to make PowerBuilder back to the market competing among other available programming languages. One of the most major efforts was the effort to make PowerScript compliant with common language specification (CLS) of .NET framework and also enhancing PowerScript by adding new programming features such as arrays, delegates and etc to make it comparable with Java or C#. By all these efforts, according to the programming languages popularity ranking, PowerBuilder is not even among the first 50 programming languages.<sup>35</sup>

### **2.3.10 Maturity**

There are some attributes that are predicated for a programming language as a mature programming language. For instance the programming language should be in public domain, it should be supported by a community such as forums. It should be used by several groups to develop enterprise successful projects, and there should be set of stable libraries or APIs and several other attributes which according to these attributes all of the candidate frameworks are mature.<sup>37</sup>

### **2.3.11 Maintainability in Future**

Software maintenance is group of efforts to improve the software functionality, such as debugging, improving performance, modifying current functionalities, adding new functionalities and so on. Maintainability is not a standard and can be defined by relatively different factors. Some of common factors counted as properties of maintainable code are; clarity or readability of the code, modularity, being easy to understand, less complexity, flexible in front of new changes and etc.<sup>38</sup>

Considering the factors that improves Software maintainability and Java attributes, such as object oriented paradigm which improves the modularity and understanding the code, Garbage Collector which decreases the complexity and probable defect and etc, we can conclude that Java is one of good option to have maintainable application.

The conditions are pretty much verifiable for C# programming language, even better by having a very expressive syntax which improves readability of the code significantly.

PowerBuilder besides providing facilities to reuse codes, by minimizing the designing time and maximizing the development time decreases the maintenance costs compared to Java or C++.

Maintainability of an application developed in a specific programming language is affected by popularity of the programming language in industry. In one hand as explained earlier, there have been significant efforts in upgrading PowerBuilder in order to keep it in track versus other popular technologies. In the other hand there are lots of efforts encouraging industries to migrate their PowerBuilder applications to state-of-the-art and reliable environment such as Java, VB.NET and ASP.NET, to decrease the risk of getting pushed out the market completely by modern technologies.

The result of programming languages popularity ranking regarding PowerBuilder shows that less people know or learn about this language which may cause trouble to maintainability issues for PB applications in future.

## **2.4 Conclusion**

The requirement analysis provided in this chapter enable us to conclude with one of these candidates as the target framework. To accomplish the final evaluation we make a summary of the result for each candidate framework as shown in table1.

Java framework fulfills the requirements specification. Java is fully supporting application internationalization and localization facilities. Socket communication is supported through Socket and Server Socket class hierarchies in Java API. It is providing several APIs to process XML documents including Java DOM, SAX and StAX interfaces. Java Database Connectivity or JDBC programming interface enables java applications to connect to any database including MySQL. Integration with Adobe Flex component is

possible through several solutions which the appropriate one for the case of this application is to embed compiled or SWF file of Flex component within Java component. Java is portable, maintainable, mature and widely used in industry. Java providing object oriented features, high level of abstraction and etc is an easy choice to for developers.

C#.NET meets the requirements specification. .NET framework provides internationalization and localization facility through Globalization API. It supports socket communication through System.Net and System.Net.Socket libraries. It provides comprehensive set of API for XML processing through System.XML namespace. Connectivity to MySQL database in .NET framework is offered through System.Data.SqlClient class hierarchy. The integration with Adobe Flex component is possible by embedding SWF within Windows Form application as well. C# applications can be cross- platform using Mono runtime software. It is maintainable, mature and widely used in software industry projects.

Power builder meets the requirement specifications as well. PowerBuilder facilitates internationalization and localization through translation toolkit and Unicode support. It supports socket communication through SocketWrench software component. PBDOM application programming interface provides XML processing for PowerBuilder applications. PB applications can connect to MYSQL database using ODBC connection. It is may be possible to integrate Flex component with PB through loading Flex executable or SWF file in Internet Explorer control embedded within PB window. Applications developed in PowerBuilder are only supported by Microsoft Windows platforms. It has not been commonly involved in industrial projects recently. It is mature programming language but since it has not been state-of-the-art in recent years, it may affect the maintainability.

The final conclusion we could come up with, considering all the requirements analysis, PowerBuilder is not a suitable choice for the target framework since it doesn't fulfill some of the requirements. It is not cross-platform and widely used in industry, the maintainability is with difficulties and the integration with Flex is not guaranteed. After that, between Java and C#.NET frameworks, both of them seem appropriate for our purpose. Considering and comparing more precisely indicates that portability and connection to MySQL database for C#.NET is not as straight forward as Java which made us to pick Java SE as the most suitable framework.

Requirements	Java	C#.NET	Power Builder
--------------	------	--------	---------------

Meet Requirements Specification	Yes	Yes	Yes
Support for Local Languages	Yes, Using Java Internationalization Java.Text.Message-Format Library	Yes, Using System.Globalization.CultureInfo Library	Yes, It has been available since PowerBuilder v.6.5 and fully supported since PowerBuilder v.10
Support for Socket Communication	Yes, Using java.net.Socket and java.net.ServerSocket class hierarchies	Yes, Using System.Net.Socket class hierarchy	Yes, Using SocketWrench software component
XML Parsing	Yes, Using Java DOM SAX, StAX libraries	Yes, Using System.XML namespace	Yes, Using PBDOM library
Database Connectivity (MYSQL)	Yes, Using JDBC library	Yes, Using System.Data.SqlClient class hierarchy provided by ADO.NET	Yes, Using ODBC library
Integration with Adobe Flex	Yes, Possible by embedding SWF file of Flex component within Java Swing	Yes, Using ExternalInterface API provided by ActionScript and through embedding SWF within C# Windows Form	Maybe, Using loading Flex executable file in Internet Explorer control embedded within PowerBuilder window
Portability	Yes	Yes, Using Mono runtime software	No, Microsoft Windows dependent
Ease of Use for Developer	Yes, providing object oriented features and high-level of abstraction	Yes, providing object oriented features and high-level of abstraction	Yes, providing object oriented features and high-level of abstraction
Widely Common in Industry	Yes	Yes	No, has not been state-of-the-art in recent years



Maturity	Yes	Yes	Yes
Maintainability Future	in Yes	Yes	Problematic, less people who learn and know about it

Table1. Summary of Framework Evaluation

## 3 Application Design

### 3.1 Overview

This chapter describes the process of the CMI application design according to the second objective of the thesis within four different sections. The first part is dedicated to overall design of the CMI application, considering different modules. The second part describes the design process of the new interface of CMI to TCC server based on the latest version of the communication protocol using XML formatted messages. The third part describes the design quality of the CMI application's logic including the modules handling the data and logical objects. The final section explains the design of new graphical user interface considering the available version of the graphical user interface.

### 3.2 Model-View-Controller Design Pattern

Before proceeding with designing of individual modules, it is required to make an overall view of the application to choose an overall strategy or pattern to design the whole application. We can view the CMI application as an application encompasses graphical user interface as one of its main facilities and logical modules to handle application data as well. User interface and logical modules require interacting with each other within the application but in an independent way. For this purpose, Model-View-Controller or MVC design pattern has been used to decouple the data access and logic handling from data presentation and user interaction of the application. Using MVC pattern adds little more complexity to the code but provides some other benefits such as improving the modularizing, usability and maintainability of the application. MVC pattern decouples the application in to three different modules including model, view and controller.<sup>39</sup>

The pattern performs in such a way that whenever a user interacts with the application, according to the interaction a proper event via the view is transferred to the controller. Then, the controller will call proper method on the model to change its state (data). As soon as the model changes its state will notify the concerning view (s) in order to get updated state from the model and display it to the user.

### **3.2.1 Model**

The model holds the data and logic of the application, so includes some objects with set of public functions either to change the states of the models or get information about the states. A model should be able to register view(s) and to notify them as well when its state changes.

### **3.2.2 View**

The view holds for data presentation and user interactions which includes all the graphical user interfaces. A view would display and update the data by querying the model which is displaying data for.

### **3.2.3 Controller**

The controller acts as an intermediate between model and view modules. A controller translates and transfers the user interactions such as button clicks, menu selection and etc that happen in the views to actions that the models perform to update their states according to the user interactions.

There are two types of MVC patterns. First the passive type in which the controller informs the model to change its state and then informs the view of the change as well. Second is the active type in which the controller only informs the model to change its state, after is the model itself which notifies the view about change. In applications that there is more than one source to change the data the active MVC should be used. In our case, since the application shall run on different nodes of the network regarding the fact that there is only one dispatcher in the network. The nodes connect to the dispatcher which sends them information of the track layout objects and trains periodically and they nodes can request changes on the data by sending messages to the dispatcher.<sup>39</sup>

Whenever the user interacts with the application interface (view) the following sequence occur, as the figure shows.

View realizes the interaction using the action listener registered for the interaction. For instance every time that user pushes a button, action listener registered for the button is being called.

View transfers the interaction to the Controller, through action listener that calls appropriate method in Controller.

The Controller calls the appropriate method on model to update itself if required according to the interaction.

As soon as the Model updates, notifies all its registered listeners of the change which can be one or several views to get the latest changes in the data.

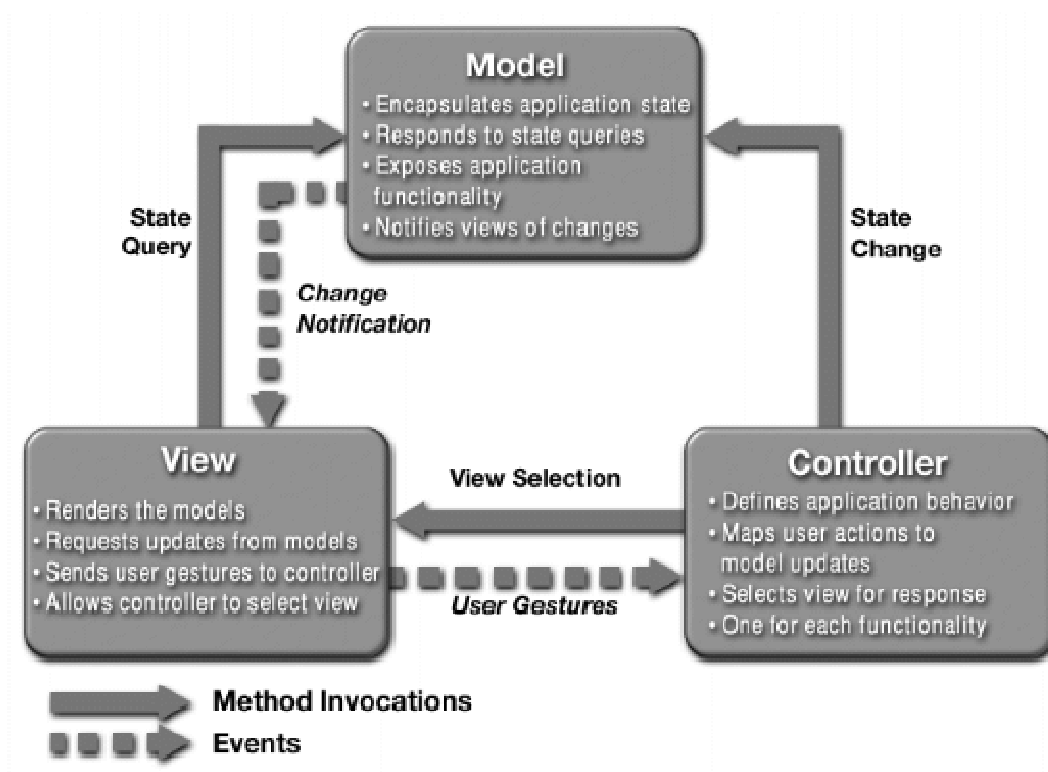


Figure1. MVC Design Pattern <sup>40</sup>

### 3.3 TCC Interface

This part describes the process of designing the new interface between the CMI application and the TCC server using the XML-based communication protocol. The first step to design the interface was studying the communication protocol to understand the content and structure of the protocol and message hierarchies. The next step was to

study Extensible Markup Language or XML technology considering available XML facilities within the target framework to choose the most proper XML processing API. The purpose of these studies is to find a way that leads us to the best facility which provides the most simple and understandable implementation.

### **3.3.1 Communication Protocol**

The communication protocol between CMI application and TCC server is an extensive protocol with a hierarchical structure based on a specific general message format. The general message format consists of three different sections, message head, message data and message error check, as indicated in figure14. According to the protocol, every message consists of message components which are either elements containing other components or variables containing a value.

Message head consists of five components including receiver ID, sender ID, message type, and time stamp and sequence number which are variables holding values of receiver ID, sender ID, message type, time stamp and sequence number, respectively.

Message data consists of the message to be sent which is an element contains one of the main message types out of the nine main categories of messages using in the protocol. The main types of message is including ConnectionMessage, ControlMessage, AlarmMessage, Indication, SystemCtrlMessage, SystemStatusMessage, CtrlResponse, RefreshReady and AlarmAcknowledged messages.

Message error check consists of cyclic redundancy check or CRC which is an error detecting code calculated for the message to detect accidental changes that may occur in data while transferring through network. CRC32 is one of the most common variations of CRC that has been used in this protocol.

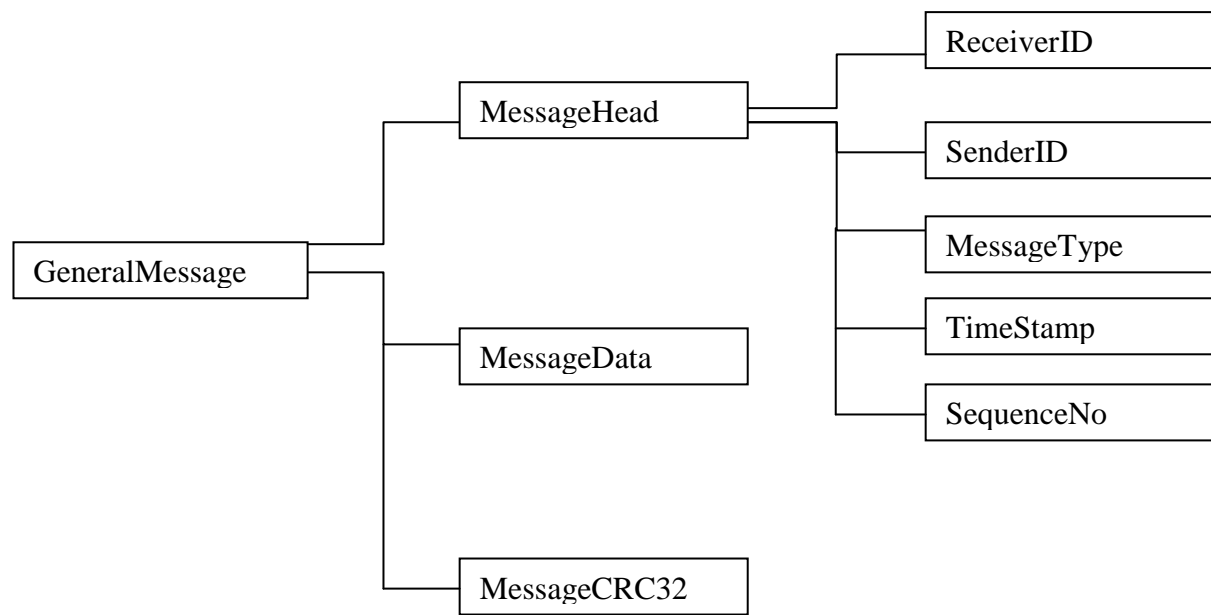


Figure14. General Message Format

### 3.3.1.1 ConnectionMessage

ConnectionMessage contains seven sub categories of messages in order to handle all the functionalities regarding connection procedure between the CMI and TCC applications. The types of ConnectionMessage are including AcknowledgeConnection, ReqConnection, Disconnect, Login, Logout, ContactRequest and ConfirmStartup messages.

### 3.3.1.2 ControlMessage

ControlMessage contains seven sub categories as well to send different commands from CTC application to TCC server. The types of ControlMessage are including ObjectCtrl, RouteCtrl, TrainCtrl, TSRCtrl, ESACtrl, AcknowledgeRequest and CommandAcknowledge messages. These commands are used to handle different types of objects including derailer, point, track and location objects through ObjectCtrl message, setting and cancelation of the route objects through RouteCtrl message, different operations of the train objects through TrainCtrl message, to handle TSRs through TSRCtrl command or to handle emergency stop areas through ESACtrl message in the Track Layout.

### **3.3.1.3 Indication**

Indication messages are sent from TCC to CTC spontaneously when changes in the objects state occur in order to update the CTC information. This message can consists of arbitrary number of three different components including Object, Train and TSR. Object element can contain the information of different object including Derailer, Detector, ESA, Point or Track in the Track Layout. Train and TSR elements respectively contain information of Trains and Temporary Speed Restriction objects in the Track Layout.

The remaining messages are small messages which don't contain more sub components. These messages are including AlarmMessage which is sent from TCC to CTC to indicate an alarm, or a status change of a previous alarm, SystemCtrlMessage to control individual computers in the system, SystemStatusMessage to indicate status of individual computers in the system, CtrlResponse to use as response to messages, and RefreshReady to indicate that a requested refresh is ready. And AlarmAcknowledged message is sent from CTC to TCC when operator has acknowledged an alarm, which is identified by an AlarmInstance number.

## **3.3.2 Method**

### **3.3.2.1 Extensible Markup Language**

Extensible Markup Language or XML is an open and text formatted language made by W3C organization which is an organization devoted for developing the Web and standardizing protocols. Ease of use over internet, get supported by wide range of applications, being human readable, Being easy to create and quick to design and being easy to handle XML document through programs are some of the most important considerations in design of XML. <sup>41</sup>

This language has been originally intended for publishing large amount of electronic data on the Web. Today XML is being used to exchange various types of data on the Web and also communication between application programs. XML is a generic markup programming language drawn from SGML. It is extensible knowing the fact that it is possible to define different namespaces, namely different languages with their own vocabulary. XML is easily recognizable by the usage of "<" and ">" signs framing markups. The main objective of XML is to facilitate interoperability of heterogeneous information systems. <sup>41</sup>

XML 1.0 was published on February 10th 1998 by the W3C (World Wide Web Consortium). This international organization, founded in 1994 and still headed by Tim Berners-Lee, the recognized inventor of Web was created in order to standardize and promote compatibility between internet technologies such as HTML, RDF or CSS. XML was originally a project intended to “enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML” <sup>41</sup>. This objective has been completed as XML is now widely used to transfer information between different languages, thanks to an easy implementation and standardized syntax and tools.

The difference with HTML, which is also a markup language, is that XML can theoretically use as many and as different markups as necessary. When HTML has limited markups intended for a particular purpose (mainly orientated for document presentation), XML can be considered as a meta language which allows inventing new markups regarding which information the user wants to isolate and/or use. <sup>41</sup>

However, XML differs from HTML in its usage, even if it is possible to present documents with XML, it is mostly used in order to permit communication between languages, to share information. XML is what we can consider as a simplification of SGML, the first language for interoperability in the World Wide Web, way too difficult for a standard user. Anyway, XML shouldn't be considered as a language such as C or Java and is only intended to structure data in a standard way. The strength of this language is to have been able to federate the web community. Now plenty of languages gravitate around XML which allow to manipulate XML documents (DOM, SAX), to describe them (DTD, XML Schema), to format them (CSS), and so on. It is also now a standard document for office tools like MS Office, OpenOffice or Apple's iWork and is understood by most of scripts and programming languages. <sup>42</sup>

### **3.3.2.2 JAXP Technology**

Java API for XML processing provides various libraries to process such as parse transform or validate XML documents for Java applications. It consists of three different XML processing interfaces, Document Object Model or DOM, Simple API for XML or SAX and Streaming API for XML or StAX interfaces. Within the following section, we provide an overview of all of these interfaces considering the properties offered by them.

### **3.3.2.3 DOM interface**



Document Object Model or DOM interface builds an object representation of XML data. DOM is the easiest XML API to understand use, since it represents XML data in familiar tree structure. It is also more proper for interactive applications as the DOM parser when parsing the data loads the whole data in memory that provides easy traversing and manipulation.<sup>43</sup>

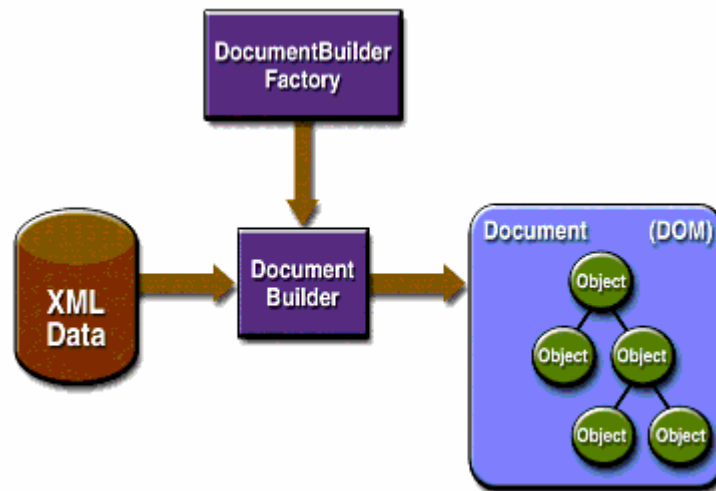


Figure11.Java DOM Architecture<sup>43</sup>

Java DOM API provides proper classes and methods to create and parse XML files. It treats XML files as objects in memory.

#### 3.3.2.4 Create DOM Objects

As mentioned before, DOM interface represents XML data in a tree structure in such a way that there is a single root node at the top and different other nodes can be appended to the root as its children. Attributes, text and comments can be added to any node (element) as well.

In order to create XML documents, first of all is required to create a blank DOM object. DocumentBuilderFactory class is used to create an instance of DocumentBuilder object. Using the instance Document object can be created which is the DOM tree. After creating the blank DOM object, using provided methods by Document interface mostly createElement(String tagName) and appendChild(Node newChild) methods, different elements and attributes can be created and get appended to the other nodes.<sup>43</sup>

### 3.3.2.5 Parse XML Documents

The result of parsing will be Document object which can be traversed using provided methods of the parsing API to arbitrary access the elements and their attributes. In this application the methods that have been used the most are `getElementsByTagName(String tagname)` to access specific elements by their names and `getChildNodes()` to get child nodes of an specific element. <sup>43</sup>

### 3.3.2.6 DOM Schema Validation

Using DOM schema validation, XML data after parsing and converting to Document objects, can be validated according to a specific XML schema. In order to do the validation, first of all is required to make an instance of Schema object to load the schema in to it. Then multiple DOM objects can be validated using that. <sup>43</sup>

Package	Description
org.w3c.dom	Defines the DOM programming interfaces for XML documents, as specified by the W3C.
javax.xml.parsers	Defines the DocumentBuilderFactory class and the DocumentBuilder class, which returns an object that implements the W3C Document interface.
javax.xml.validation	Defines the Schema classes.

Table2. Java DOM Packages <sup>43</sup>

### 3.3.2.7 SAX interface

SAX interface is called SAX parser. It provides a simple API for XML documents. It parses the XML data as a stream of events and doesn't make a representation of data on memory, so it uses less memory and is faster. SAX API writes and reads the XML data to and from a data repository or the web, and it provides serial access to the data. <sup>44</sup>

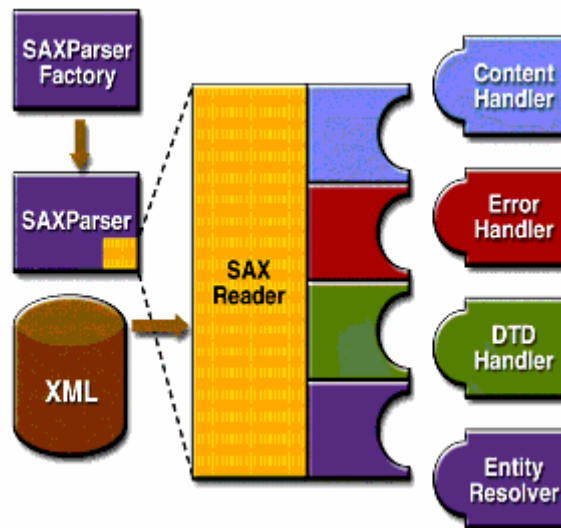


Figure13. Java SAX Architecture <sup>44</sup>

In order to process XML documents using SAX, first it requires creating an instance of SAXParser using SAXParserFactory class. SAXParser includes a SAXReader which can be configured using getXMLReader() method. The next step is to implement some callback methods to be called when the parser fire the events. Then the SAX parser is ready to parse the XML documents using a ContentHandler passed on it. ContentHandler constitutes the parser to handle the document content including the methods startDocument(), endDocument(), startElement(), and endElement() and etc. ErrorHandler can be provided to the parser to handle parsing errors. EntityResolver is used to located XML document identified by a URI or a public identifier. When a XML document passed through a SAX parser, based on document features encountered it will generate proper type of event such as element nodes, text nodes, comments and etc. <sup>44</sup>

Package	Description
org.xml.sax	Defines the SAX programming interface.
org.xml.sax.ext	Contains classes to handle more complex SAX processing.
org.xml.sax.helpers	Defines some helper classes.
javax.xml.parsers	Includes the SAXParserFactory class and also classes for exception handling.

Table3. Java SAX Packages <sup>44</sup>

### 3.3.2.8 StAX interface

StAX interface is the most recent XML API by JAXP as an alternative strategy to DOM and SAX with the aim of improving their drawbacks such as complexity of SAX and memory management in DOM. It implements the Streaming API for XML standard. It provides a pull parsing API for streaming XML documents. StAX programming interface facilitates two different strategies for XML processing including cursor strategy and iterative strategies.<sup>45</sup>

### 3.3.2.9 Cursor Interface

The cursor-based API provides the developer to moving forward through XML documents using a cursor, through XMLStreamReader and XMLStreamWriter class hierarchies. XMLStreamReader interface enable the user to read XML documents as streams, one by one from the beginning forwards to the end. It provides a set of appropriate methods to retrieve all required information from XML document elements. XMLStreamWriter interface enable the users to write to a XML stream.<sup>45</sup>

### 3.3.2.10 Iterator Interface

The Iterator API provides developers a representation of XML document as a set of events, which are generated by the parser as they are read from the source document and pulled by the application. The base library of the API is XMLEvent to handle the event objects which contains other class hierarchies to handle different event types similar to SAX, for instance StartDocument, EndDocument, StartElement, EndElement and etc. The Iterator API offers XMLEventReader and XMLEventWriter respectively to read and write events. XMLEventReader iterates over XML stream to allocate event objects and can be extended to read customized event objects as well. XMLEventWriter interface enables the developers to write events iteratively to an XML stream without letting modification after writing them.<sup>45</sup>

Package	Description
javax.xml.stream	Defines the StAX programming interface.
javax.xml.transform.stax	Contains classes for handling StAX transformations.

Table4. Java StAX Packages<sup>45</sup>

Considering features of different XML parsing interfaces as shown in table5, DOM interface provides all kinds of processing on XML documents using simple and intuitive methods avoiding complexity of the two other interfaces.

Feature	StAX	SAX	DOM
API Type	Pull, streaming	Push, streaming	In memory tree
Ease of Use	High	Medium	High
XPath Capability	No	No	Yes
CPU and Memory Efficiency	Good	Good	Varies
Forward Only	Yes	Yes	No
Read XML	Yes	Yes	Yes
Write XML	Yes	No	Yes
Create, Read, Update, Delete	No	No	Yes

Table5. XML Parser API Feature summary <sup>45</sup>

### 3.3.3 Results

Studying the communication protocol and relevant methods led us to design the `tccInterface` package, contains classes to handle the interaction between CMI and TCC applications. The interaction can be including compose XML messages with all data included and send them through the connection as well as receiving the messages and parsing them.

Besides `tccInterface`, it was required to manipulate different types of XML messages with all required information in such a way that respects the hierarchy of them as in the communication protocol. This purpose was achieved by designing a set of protocol handler packages. In order to make the design intuitive and easy to understand and maintain, the class design hierarchy in these packages is following the hierarchical structure of the messages in the communication protocol.

Protocol handling packages includes `protocolHandler` containing main types of messages, and other protocol handler packages are dedicated to each main message

type which is containing other sub-categories. `protocolHanler.connectionMessage` package is dedicated to manage all sub-categories of `ConnectionMessage` type. `protocolHandler.controlMessage` has been designed to handle all types of `ControlMessage`. And `protocolHandler.objectCtrl` and `protocolHandler.trainCtrl` are respectively designed to manage different types of `ObjectCtrl` and `TrainCtrl` messages.

### 3.4 Application Logic

This part of the CTC application contains the modules handling the logic and data used within the application. According to Model-View-Controller pattern, modules regarding the application model are included in the application logic. In fact the other modules that manipulate data including `tccInterface` and protocol handling modules are included in application logic but based on thesis objectives and their functionality has been discussed in earlier chapter.

In order to design model modules, classes have been designed to handle objects with relevant functionalities based on object oriented design. For instance a set of classes has been considered to handle logic of MDI parent window and its main components such as menu bar, tool bar and so on, which are included in a package called `model`. The classes are related using aggregation in such a way that MDI parent window is aggregates of the classes regarding its components.

Furthermore, it was required to design classes to handle the data regarding the child windows, the data which are accessed and manipulated through the child windows. For instance `Location` class has been designed to provide appropriate properties and methods to preserve the information regarding location objects and handle them in the Track Layout. All these classes are included in `model.MDIChildrenObjects` package.

The other major part of application data is regarding the Track Layout. The Track Layout contains several different objects such as Tracks, Points, Trains and etc, which need to be considered as separate objects within the application. These objects share some common properties considering that a general type of object called `TrackLayoutObject` has been defined to hold the common properties. All other track layout object including `Derailer`, `Detector`, `ESA`, `Point` and `Track` are inheriting this general type included in `model.trackLayoutObjects` package.

## **3.5 Graphical User Interface**

This section is dedicated to the Graphical User Interface design process according to the requirements specification and considering the available version of the user interface. The requirements specification has been considered during the first phase of the thesis in order to accomplish the framework evaluation. The available version of the user interface should have been considered as an example of CMI user interface which could be inspiring for design of the new user interface.

### **3.5.1 Design Theories and Methodologies**

As we mentioned earlier, at the beginning we had to consider the requirement specifications which the previous version was designed based on. Later on some modifications have been applied in the specifications to design the new version.

We did not have a finalized design for this project and one of our objectives was to design a new graphical user interface (GUI) that can improve the interaction between the users and the system in the sense of being user friendly. The steps involved within the design process we followed to achieve this goal are explained here as indicated in figure 14.

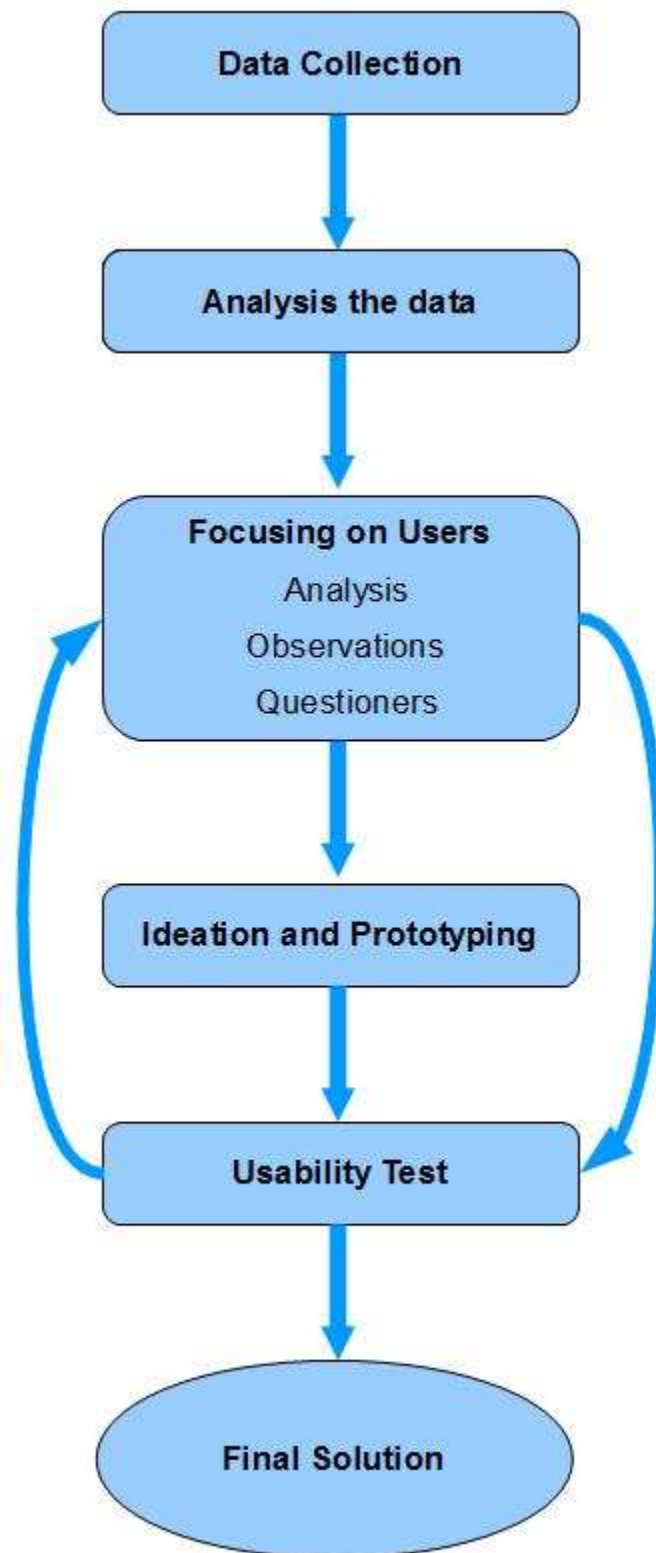


Figure14. Design Process



### **3.5.1.1 Data Collection**

As we mentioned earlier at the beginning of this project the only thing that we had was a specification which the old version of the project was designed by and there were some changes on that to design the new one. Then we decided to categorize the data in different groups such as Logic requirements, protocols and the graphical user interface requirements.

### **3.5.1.2 Analysis of the data**

After the data collection phase we had been able to define the related data for each part and analyze the relations between them and gather the data for the new designs.

### **3.5.1.3 Focusing on users**

The application must be understandable for all these different kind of users, the main focus was on Controllers and as we did not have access to them. We did the observations with the test engineers in the company who are testing the systems in different manners. This phase had been held in a circle of analysis, observations and questioners in order to define the things that we needed to avoid and the functions that we needed to focus more.

### **3.5.1.4 Using Ideation and prototyping**

When we finished doing the previous phase, it was time to come up with some solutions through the brainstorming. And the next step is to design prototypes. Designing different prototypes would lead us to get more close to the final solution. We started to first design wire frames and again turning back to the latest phase which was focusing on user groups. In this phase we came up with two different alternative designs.

The first alternative design was most related to the access procedure that user could have to the system. At the two first steps, Login and MDI Parent Window, the toolbar is invisible. Instead of opening the latest window, when the operator passes the login page then in the MDI Parent Window, four thumbnails will be shown, which are the four latest areas that the operator was involved with and will be sorted by date and time of termination. Three menus considered for this design which are File, View and Help. The items in tools and window (perspective) menu are accessible from two sub menus inside of View menu and also all zoom and Tools items are accessible in the Toolbar. The name of "Window" is changed to the "Perspective".

The other alternative that we had was having multiple windows in tabbed system.

Instead of having different child window in the MDI we considered a tab system which designed the most important child windows as some tabs that users can switch to them by using arrow keys and other shortcut keys.

#### **3.5.1.5 Usability Test**

After designing the alternatives it was the time to have more observation that how the users interact with the new designs and this phase could influence the designs in addition and give us a better and closer image to the final solution. This phase again had been done with mostly questioners about the features of the design and the user experiences.

#### **3.5.1.6 Final Solution**

At the end we came up with one solution that was optimum comparing to the old version of the application and many changes to the design standards which could satisfy the client. In the bottom we explain the differences and changes that we have done in the graphical user interface.

According to the requirements specification, the user interface should be following Multiple Document Interface or MDI structure standard. MDI structure encompasses a main or parent window which starts over after running the interface. The parent window normally contains a menu bar and a tool bar at the top, a desktop in the middle and a status bar at the bottom. The menu bar and tool bar enables the user to access the application functionalities and manipulate the components within the application. MDI parent window can contain arbitrary number of child windows which reside within the desktop container. The child windows perform independently and can have their own child windows or popup windows. By the way, all implementations provide some ways for the main window to control or keep track of them.

The main advantage of Multiple Document Interface which has been considered for this application as well is that user can access and manipulate data through several windows simultaneously. This property fulfils the needs for CMI interface users which in many cases may need to have several windows open at the same time to process data on them, and it was the reason why MDI interface beat the tabbed interface alternative.

### **3.5.2 Method**

Abstract Window Toolkit or AWT and Swing libraries of Java programming language have been used to design and implement the graphical user interface.

### 3.5.2.1 Java AWT and Swing

AWT is the core foundation of Java libraries that support graphical user interface development. The API provides event driven user interface, including graphics, imaging, fundamental user interface components including windows, buttons, and layout managing and etc.

Swing is a platform independent API built on top of several Java desktop technologies such as AWT, Java 2D and Java Internationalization. The framework contains more comprehensive and flexible set of graphical user interface components which enable the developer to build desktop and internet applications. Swing provides additional feature for applications called “Look and Feel” which can be native to make the application look as the underlying platform or pluggable to provide different look than the user current platform. Swing architecture allows the user to override the default implementation of the components in order to get their own customized features. Swing library has been designed using Model-view-Controller software design pattern which is for decoupling the data sections of the application from the interfaces. For instance the Swing component for table called JTable has a model TableModel to manage the tabular data. <sup>46</sup>

### 3.5.2.2 Swing GUI Building

Netbeans IDE provides a GUI building feature that enables user to create graphical user interface by dragging and positioning the components from a palette containing AWT/Swing components on to a canvas. Indeed using GUI Builder components with the default setting are added to the interface and the user can easily edit the properties later on. Besides GUI Builder takes care of the alignments and spacing of the components which is complicated to set by manually coding. <sup>47</sup>

The feature has been used to implement the new version of graphical user interface of the application. There is such a feature in Eclipse IDE as well, but it was not as complete and easily used as the one provided by Netbeans. The advantage of using this tool is that the effort to arrange the components next to each other with proper properties such as place, size, font and etc is not comparable with the way to do it by customizing them from scratch. There are some disadvantages as well. The first disadvantages is that since the code is added automatically by the IDE, is quite huge bulk of code which doesn't look human friendly, so it requires to get edited and kind of pruned in order to make it easier to read and understand. The other disadvantage is that, the GUI Builder in fact just takes care of the design and not the generated code and it doesn't make any classification on the components or objects, so every component that is added in a canvas they are part of the same class. For instance when we create the menu bar, all the menu items added to it

belonged to the menu bar class which is not object-oriented designed anymore. Suppose the case that later on they need to add one more menu item to the menu bar, in such a case they have to make edition in the menu bar class which doesn't seem good idea at all. Thus the solution was to decouple the different components and dedicate a different class to each of them to be able to treat them as separate and independent objects later on. So in case of needing for changes like the example we explained, they can just add one more menu item object to the existing set of classes.

### **3.5.3 Results**

Graphical user interface classes, according to Model-View-Controller design pattern are included in view module which are divided in to three packages, view, view.MDIChildWindows, and view.secondaryWindows.

View package contains the components of the MDI parent window, including classes that manipulating the MDI window, menu bar and its sub-components including file menu, view menu, tools menu and help menu as well as the tool bar, status bar and also the user login window.

The package view.MDIChildWindows is including the classes to handle MDI child windows interfaces, that the user can have access to them through the view menu item of the menu bar. The view.secondaryWindows package has been designed to manage secondary windows which provide the user further information on different data or objects. The user is able to update some of the information through them, as well. These windows are accessible from the child windows which belong to.

# 4 Implementation

## 4.1 Overview

This chapter is dedicated to implementation results of CMI application, including interface to TCC, application logic and user interface.

## 4.2 TCC Interface

As explained in the design process, the `tccInterface` package contains classes to manage interaction between CMI and TCC applications, such as send or receive messages as well as processing the messages. These classes include `CRC32Checksum`, `FormConnectionMessage`, `FormControlMessage`, `FormObjectCtrl`, `FormTrainCtrl`, `MessageFormer`, `MessageHandler`, `MessageParser`, `Startup`, `XMLInputStream`, `XMLOutputStream`, `XMLSender`, `XMLReceiver` and `LifeSign`.

The user interface interacts with `tccInterface` package within the application in order to send and receive messages. The interaction is performed in this way that, according to the user interactions the application requires to send appropriate XML messages to the TCC; therefore it has to go through `tccInterface` classes to form the message using `MessageFormer` class and then send it using `XMLSender` class.

The application creates the corresponding type of the message with given data and then pass it to `MessageFormer` class as an object of general type of `Message` using the polymorphism principle. `MessageFormer` reveals type of the message and build the general format of the message and then based on the type passes it to a more specific message former class such as `FormConnectionMessage`, `FormControlMessage` and etc to complete body of the message. `MessageFormer` has been implemented using Java DOM libraries and creates XML document objects messages based on the general message format in the communication protocol.

Receiving the messages has a bit different scenario. After startup the connection, the application is always waiting for messages through `XMLReceiver` object. Whenever it receives an XML message, it will pass it to the `MessageParser` class in order to parse the message according to its type, using DOM implementation and extract the included information. After extracting the message's information, the user interface gets updated if required. For instance if the CTC receives indication message, the Track Layout or child windows gets updated according to the new information of objects.

### 4.2.1 Protocol Handling

In the top level of protocolHandler class hierarchy implementation, there is a very general class called Message from which all main categories of messages are inherited. It only includes an enumeration of different message types and default constructor. In next level of the design hierarchy there are the main types corresponding the nine categories of messages in the protocol including ConnectionMessage, ControlMessage, AlarmMessage, Indication, SystemCtrlMessage, SystemStatusMessage, CtrlResponse, RefreshReady and AlarmAcknowledged, which all inherits the type Message. The next level contains classes corresponding sub categories of the main message types included in protocolHandler.connectionMessage, protocolHandler.controlMessage, protocolHandler.objectCtrl and protocolHandler.trainCtrl packages.

## 4.3 Application Logic

As described in the design process, some of the main parts of CMI application, according to Model-View-Controller design pattern are included in model packages.

### 4.3.1 Model

This package includes classes to handle the logic of the classes in view package which are the main interface objects such as LoginView, MDIParentView class and its components. This package includes LoginModel, MDIParentModel, MenuBarModel, ToolBarModel, StatusBarModel, FileMenuModel, ViewMenuModel, HelpMenuModel and ToolsMenuModel.

In the top level of the class hierarchy there exist LoginModel and MDIParentModel classes. In the next, there are MenuBar, ToolBar and StatusBar which aggregate which MDIParentModel is aggregate of them. At the bottom of the hierarchy there are FileMenuModel, ViewMenuModel, ToolsMenuModel and HelpMenuModel aggregate in to MenuBar.

Each class contains appropriate properties in form of variable or data structure to keep track of the data relevant to the concerning view class, and proper methods as well to perform required actions regarding the data held by the objects.

#### **4.3.1.1 model.MDIChildrenObjects**

This package contains classes handling the logical functionality of MDI child windows including Alarm, Location, TrackLayoutModel, TrainControlModel, TrainListData, TSRListData and User classes.

Alarm class is used to keep track of the alarms within the application. Whenever an alarm is recognized by CTC it instantiates this class using the alarm properties. The alarm instances are saved in an appropriate sort of data structure and are displayed in AcknowledgeAlarmView window upon the user request.

Location class is used to handling location objects representing available locations in the track layout. TrackLayoutModel class handles the logic of track layout object. TrainControlModel class is used to perform provided actions on the trains when a train is selected such as view the train information and unregister the train or view train error.

TrainListData class is used to keep track of required information for trains within the application in order to display in case of user request. This information is including train name, status and destination in fact are collected from the list of train objects tracked by CTCApplicationData.

TSRListData class is used to keep track of required information for available temporary speed restrictions within the application. This information contains start track, end track and speed of TSRs which are collected as well from list of TSRs tracked by CTCApplicationData in order to display them upon the request.

User class is used to handle the users of the CTC application. This object keeps track of user information including username, password and access level. When user is about to login to the application, CTC makes an instance of the object with the user information and send it to TCC for the validity check.

#### **4.3.1.2 model.trackLayoutObjects**

This package contains classes handling the logical functionality of Track Layout objects including, Derailer, Detector, ESA, Point, Track, Train, TSR and their components including TrainExtension, RouteExtension, Extension, TrainConfiguration, TrainIdentity, TrainStaticData and TrainDynamicData.

Derailer, Detector, ESA, Point, Track, Train and TSR classes are used to manipulate data regarding derailer, detector, ESA, point, track, train and temporary speed restriction objects in the track layout respectively.

Train object is aggregation of three other objects including TrainIdentity, TrainStaticData and TrainDynamicData. TrainIdentity class contains engine identity and train number. TrainStaticData consists of static information of train object including train configuration object, train category, length and etc. TrainDynamicData contains dynamic information of train object including TrainExtension, RouteExtension objects, speed, mode and etc.

### **4.4 Graphical User Interface**

#### **4.4.1 Functional Requirements**

Beside the technical point of view and requirements that have been discussed in chapter 2, from the practical point of view the graphical user interface shall fulfill the following functional requirements.

- The application shall enable the user to logs in or out the application.
- The functionalities shall be enabled for the users according to their access level.
- The application shall enable the user to view his last open viewed windows.



- The user shall be enabled to view the track layout with all available objects in it, such as locations, trains and so on, and also to manipulate the objects as he is privileged.
- The application shall enable the user to view unacknowledged alarms in the database and acknowledge any of them, to view list of locations in the track layout, to view list of trains in the track layout, to view list of temporary speed restrictions, and update their information or remove them.

## **4.4.2 Results**

### **4.4.2.1 View Module**

According to the MVC pattern all objects (classes) regarding the graphical user interface reside in view modules (packages), categorized in three different packages, view, view.MDIChildWindows and view.secondaryWindows which are sub-packages of the view package. The main view package contains classes regarding the MDI parent window and login window as well, MDIChildWindows includes the MDI child windows classes and secondaryWindows includes secondary windows classes.

### **4.4.2.2 View**

This package contains the components of the MDI parent window, including classes that manipulating the MDI window, menu bar and its sub-components including file menu, view menu, tools menu and help menu as well as the tool bar, status bar and also the user login window.

View package is containing LoginView, MDIParentView, MenuBarView, ToolBarView, StatusBarView, FileMenuView, ToolsMenuView, ViewMenuView, HelpMenuView and TimeDisplay classes.

### **4.4.2.3 User Authorization**

In the old version the user authorization window was appearing inside the MDI parent window while is not necessary to view the MDI parent window before log on to the system.

The other drawback about the old design was that the User Authorization window was not always on top and it might be hiding behind the MDI parent window and could confuse the user.

In the new version, user authorization window will appear asking for the username, password and access level as user information and gives the server name. The user information will be sent by Login message as explained in communication protocol to the TCC server, and then the validation will be checked in the database in TCC.

Several access levels have been defined to access the functionalities within the CTC application including Controller, Signal Engineer, Track Maintenance, Staff, Administration and Personnel. The application is supposed to allow different privileges according to the access levels.



Figure15. User Authorization

#### 4.4.2.4 MDI Parent Window

When user authorization is done, MDI parent window will appear and is supposed to display the latest child windows open by the user by preserving their size and position. MDI parent window provides user with a menu bar, tool bar, desktop and a status bar.

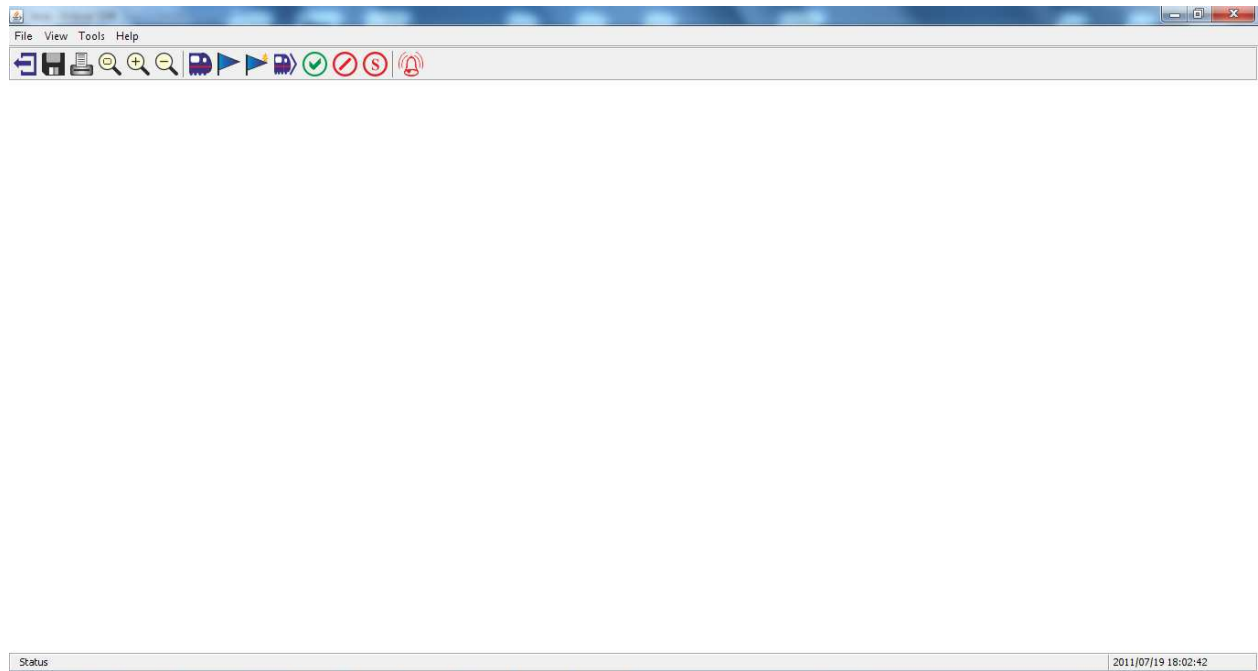


Figure16. MDI Parent Window

Menu bar control placed at the top of the MDI parent window enables. It the user to manipulate commands regarding file saving and printing, different child windows mainly the track layout and finally the help options, through four menu items, File, View, Tools and Help.

By analyzing the old version of the application and comparing with the designing standards we discovered weaknesses and tried to avoid them in the new design. For instance, in the old version of the menu bar, there are many different menu items for each menu with corresponding button in the toolbar which lead to information overloading or complexity.

Unsorted items are the other drawback in the old version. In menu designing the items should be sorted in logical groups. For instance, it is clear that the item for terminating the application should not be the first item in the menu.

The other problem with the menus was that many different items could be placed in sub menus. Using sub menus would save the application's graphical screen while viewing the menu items by the user.

Using file menu, the user will be able to save the run time data, printing, logout and exit from the system.

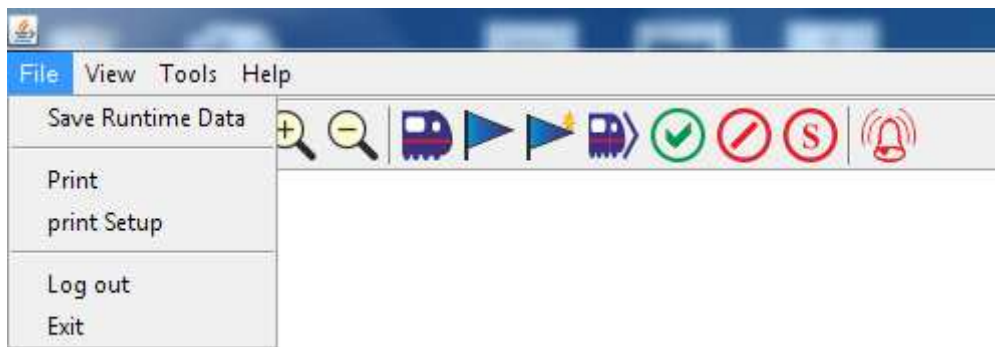


Figure17. File Menu

The view menu enables the user to view different child windows including the track layout, Acknowledge Alarm, location list, train list, Temporary speed restriction and system status, as well as the identities of available trains on the track layout and finally the zoom functionality.

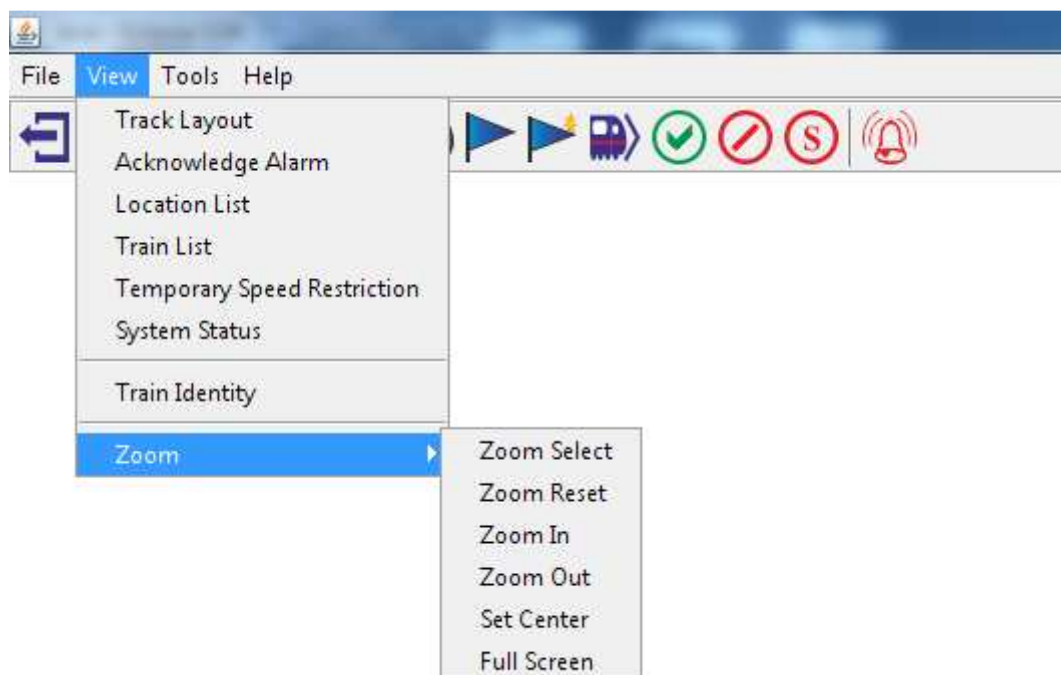


Figure18. View Menu

The tools menu enables the user to display the tool bar and run some commands on the track layout including train destination commands and several other commands to manipulate the track layout such as set temporary speed restriction.

Train destination commands include, setting a new destination for specific train, to change the destination of specific train, and to shunt specific train. Then, to run the command, user is able to confirm the request or cancel it at all.

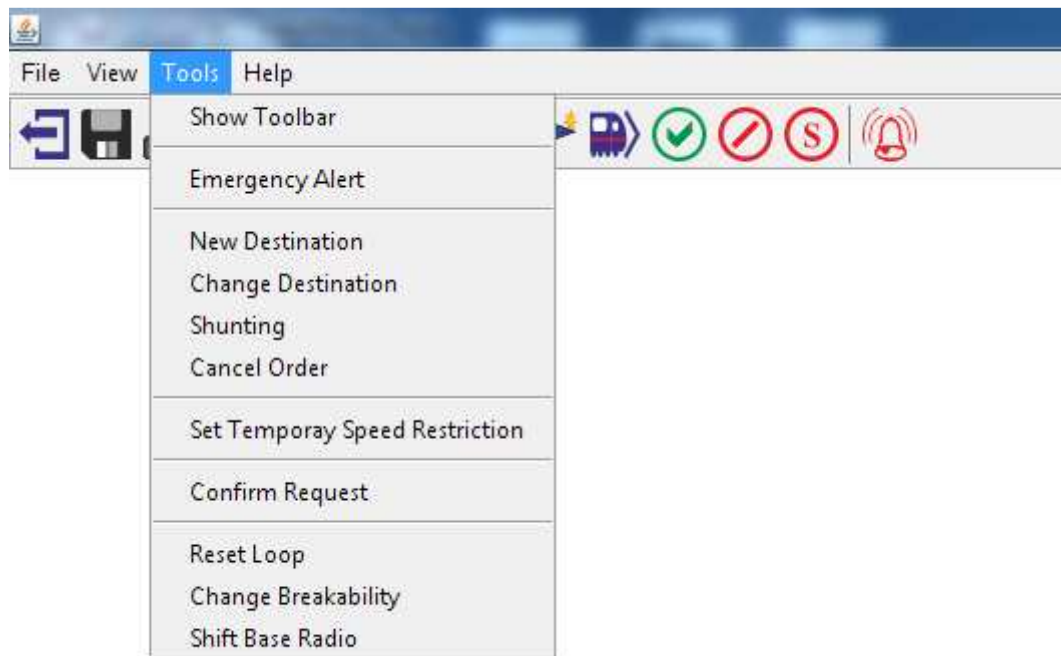


Figure19. Tools Menu

The help menu provides a help content which enable user to look up through the available subject and look up through web as well using the about menu item.

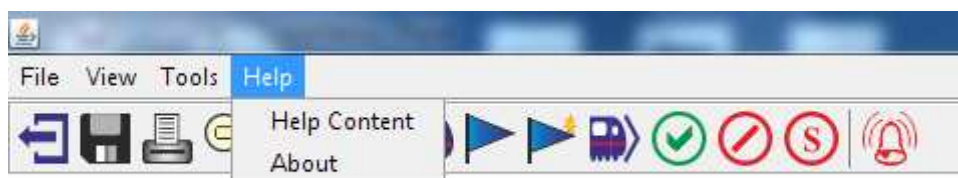


Figure20. Help Menu

The tool bar control placed below the menu bar in the MDI parent window. The tool bar contains some command buttons to enable the user to run commonly used functions.

The toolbar is supposed to provide a quick and convenient access to the set of frequently used commands or options. The typical toolbar contains iconic buttons but in some cases it contains other components.

In the old version the icons were hard to remember using complex images for small icons in the toolbar. In the new version we have replaced the icons by new icons. The icons were not sorted properly; some of them were even irrelevant to the commands (wrong Icons like alarm and logout).



Figure21. Tool Bar

The status bar control, placed across the bottom of the MDI parent window is to display the application status information and status messages.



Figure22. Status Bar

#### 4.4.2.5 MDI Child Windows

MDI child windows are the internal windows that the user can have access to them through the view menu item of the menu bar. The child windows will provide available data of the CTC application which has been received from TCC server and is supposed to get updated regularly.

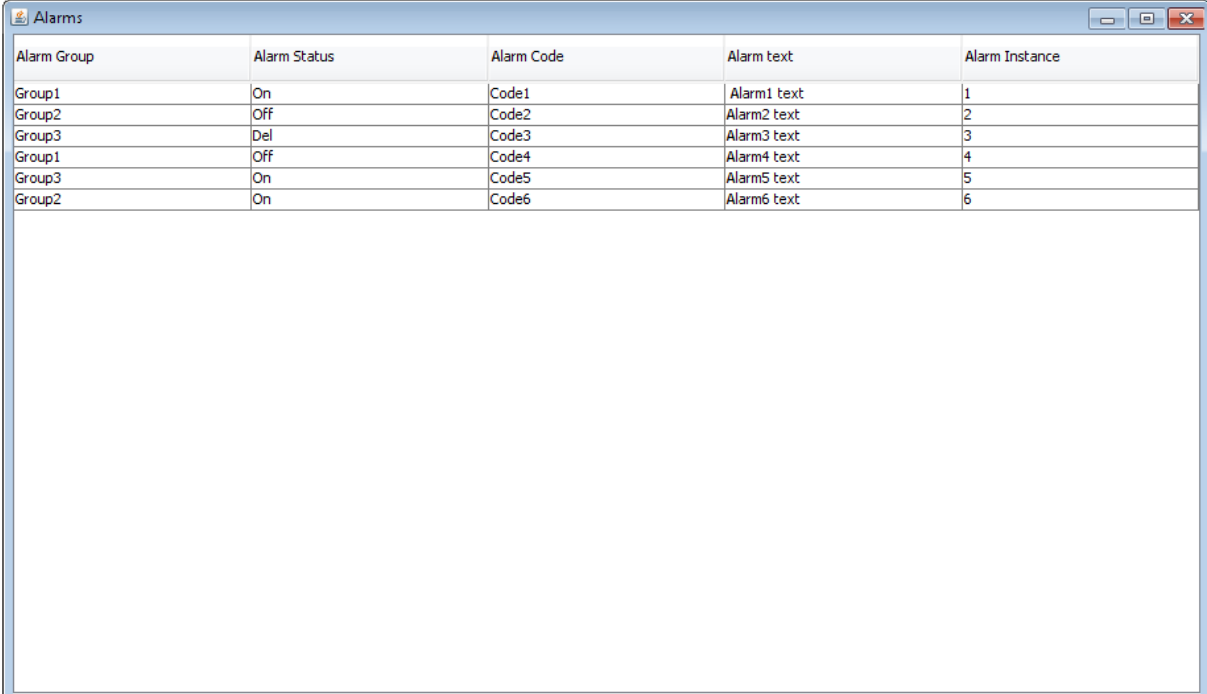
The package view.MDIChildWindows is containing classes handling MDI child windows interface, including AcknowledgeAlarmView, LocationListView, TrainListView, TSRLListView and TrackLayoutView.

#### 4.4.2.6 Track Layout

This window is supposed to represent graphic image of the actual track layout. The components of track layout are including derailleurs, detectors, ESAs, points, tracks, locations and trains.

#### 4.4.2.7 Acknowledge Alarm

This window displays a list of unacknowledged alarms with their properties, as stored in the database. User will be able to acknowledge any alarm that selects.



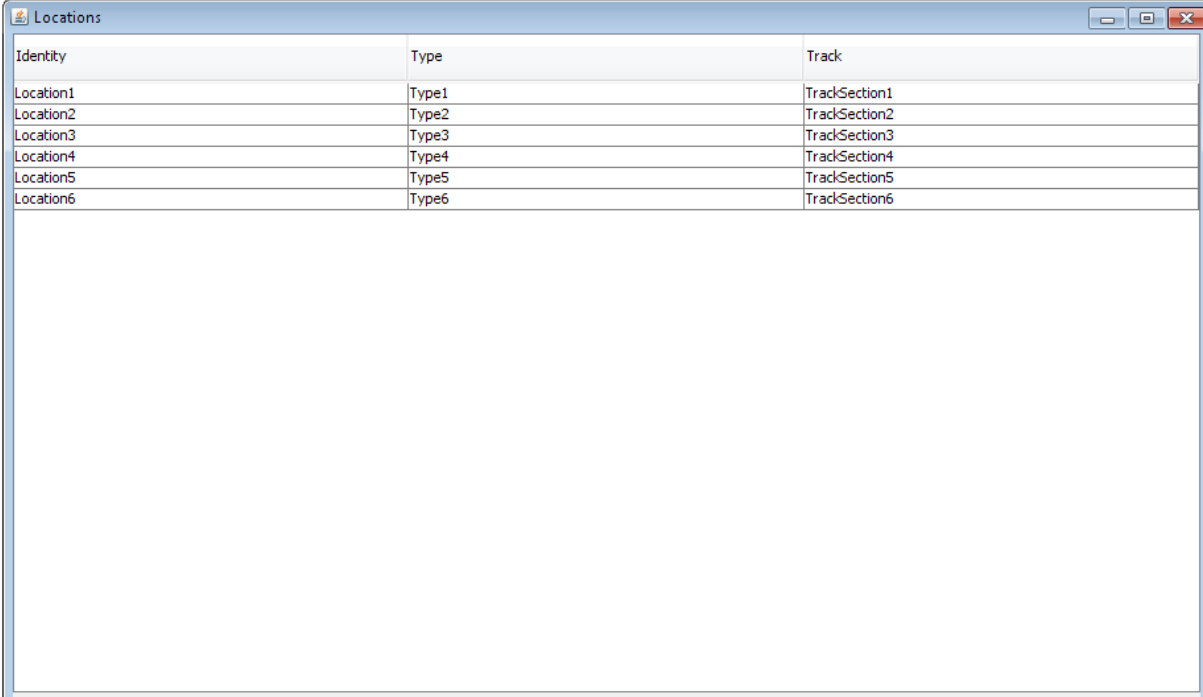
The screenshot shows a window titled "Alarms" with a table containing the following data:

Alarm Group	Alarm Status	Alarm Code	Alarm text	Alarm Instance
Group1	On	Code1	Alarm1 text	1
Group2	Off	Code2	Alarm2 text	2
Group3	Del	Code3	Alarm3 text	3
Group1	Off	Code4	Alarm4 text	4
Group3	On	Code5	Alarm5 text	5
Group2	On	Code6	Alarm6 text	6

Figure23. Acknowledge Alarm

#### 4.4.2.8 Location List

The location list window displays a list of all location with their properties, as available in the track layout. The locations are supposed to be read from CTC configuration file.



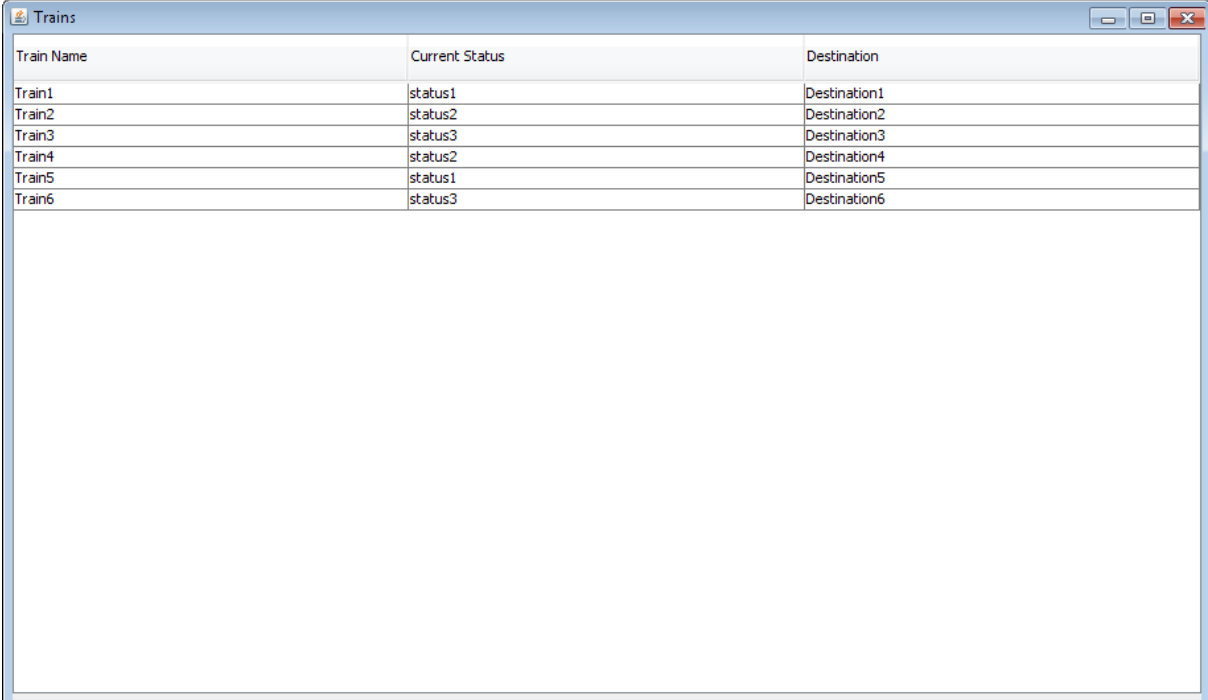
Identity	Type	Track
Location1	Type1	TrackSection1
Location2	Type2	TrackSection2
Location3	Type3	TrackSection3
Location4	Type4	TrackSection4
Location5	Type5	TrackSection5
Location6	Type6	TrackSection6

Figure24. Location List

#### 4.4.2.9 Train List

Train list window displays a list of registered trains with their information including name or identity, status and destination. CTC receives this information from TCC through indication messages.



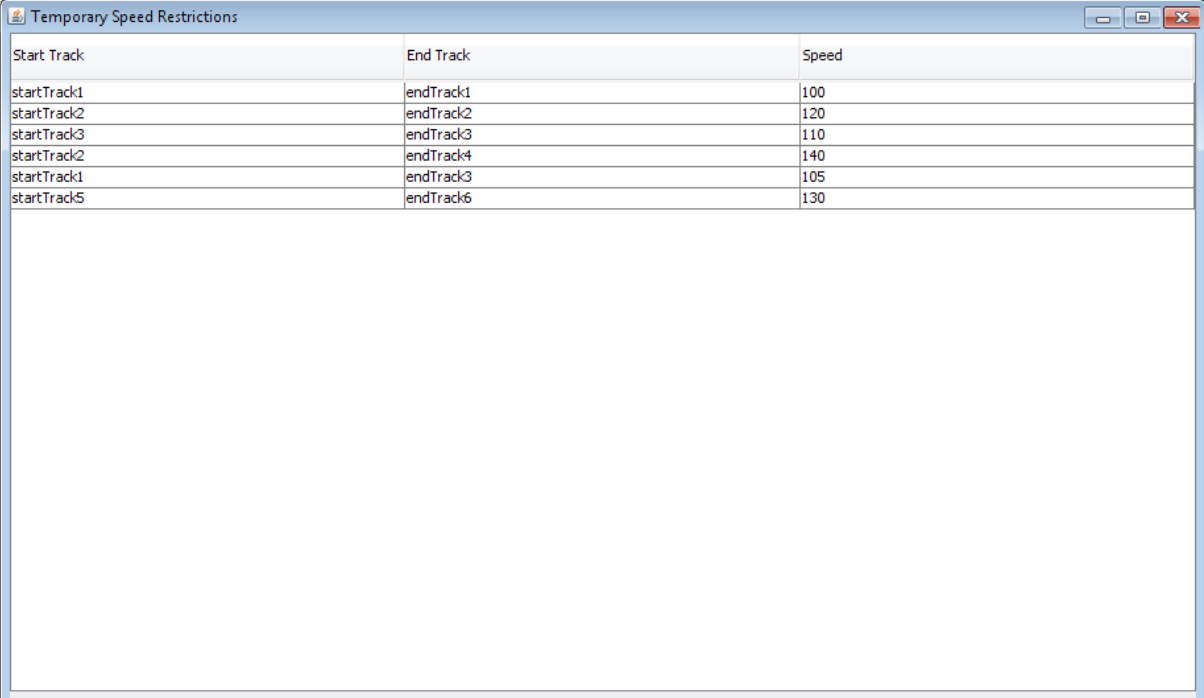


Train Name	Current Status	Destination
Train1	status1	Destination1
Train2	status2	Destination2
Train3	status3	Destination3
Train4	status2	Destination4
Train5	status1	Destination5
Train6	status3	Destination6

Figure25. Train List

#### 4.4.2.10 Temporary Speed Restriction

The Temporary Speed Restriction or TSR list window displays a list of all available TSRs with their properties, as they have been set in the application. CTC receives this information from TCC through indication messages.



Start Track	End Track	Speed
startTrack1	endTrack1	100
startTrack2	endTrack2	120
startTrack3	endTrack3	110
startTrack2	endTrack4	140
startTrack1	endTrack3	105
startTrack5	endTrack6	130

Figure26. Temporary Speed Restriction

#### 4.4.2.11 Secondary Windows

The secondary windows technically belong to MDI child windows which provide the user further information on different data or objects. The user is able to update some of the information as well. These windows are accessible from the child windows which belong to.

The package `view.secondaryWindows` is containing classes handling the secondary windows interfaces, including `LocationInformation`, `TrackInformation`, `TrainControl`, `TrainErrorIndication` and `TrainInformationView`.

#### 4.4.2.12 Location Information

This window is called from Location List window and provides additional information on any location which is selected by the user on the list.

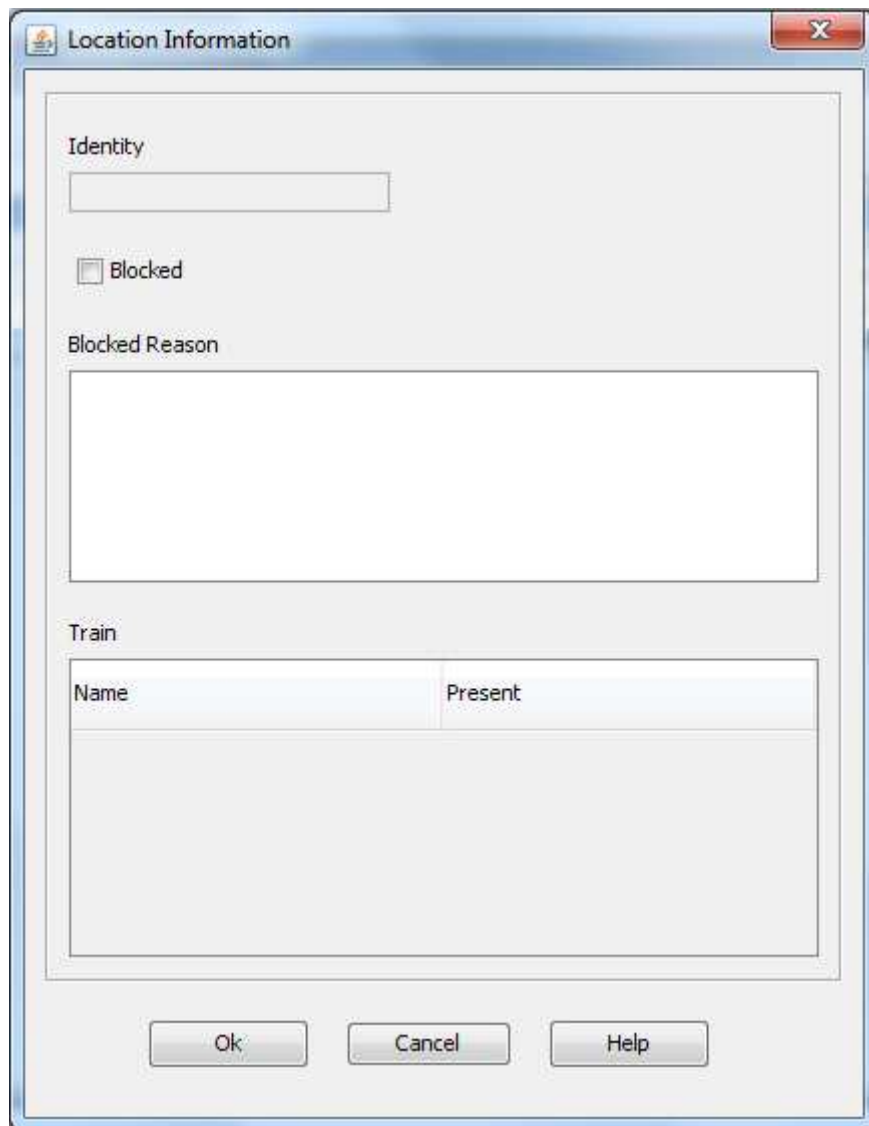


Figure27. Location Information Window

#### 4.4.2.13 Track Information

This window is called from Track Layout window and provides information on any selected track in the track layout.

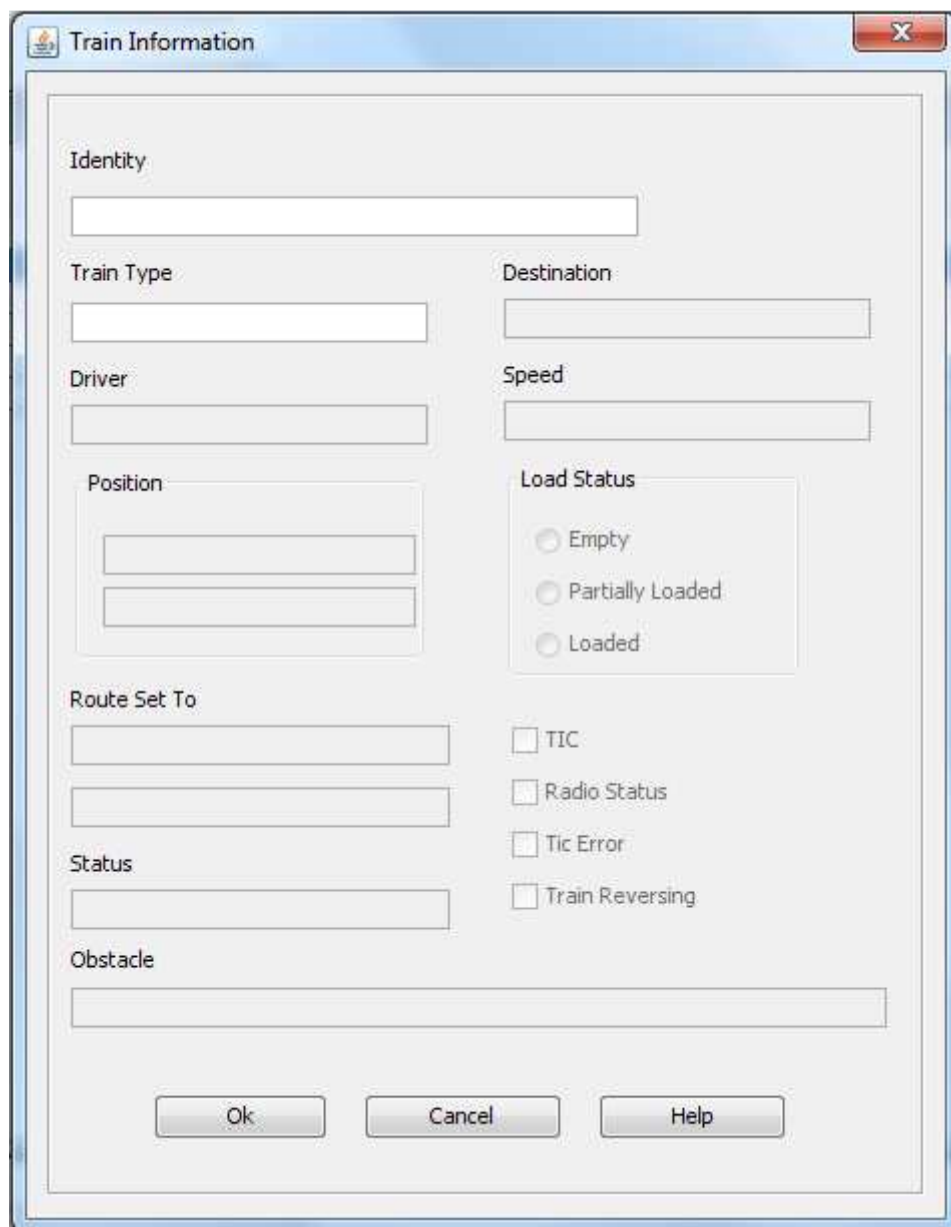
#### 4.4.2.14 Train Control

This window is called from Track Layout window and provides the user a menu with options to be able to view the train information window, unregister the train, view the

train error indication to set or reset error for the train, and view locomotive control window.

#### 4.4.2.15 Train Information

This window is called from Track Layout window and provides additional information on any train in the track layout which is selected by the user.



The screenshot shows a 'Train Information' dialog box with the following fields and controls:

- Identity:** A single-line text input field.
- Train Type:** A single-line text input field.
- Driver:** A single-line text input field.
- Position:** Two stacked single-line text input fields.
- Destination:** A single-line text input field.
- Speed:** A single-line text input field.
- Load Status:** A group box containing three radio buttons: 'Empty', 'Partially Loaded', and 'Loaded'.
- Route Set To:** Two stacked single-line text input fields.
- Status:** A single-line text input field.
- Obstacle:** A single-line text input field.
- Checkboxes:** Four checkboxes are located on the right side: 'TIC', 'Radio Status', 'Tic Error', and 'Train Reversing'.
- Buttons:** At the bottom are three buttons: 'Ok', 'Cancel', and 'Help'.

Figure28. Train Information Window

#### **4.4.2.16 Train Error Indication**

The Train Error Indication window is called from the Train Control window, providing information on the error status of the train.

#### **4.4.2.17 Locomotive Control**

The Locomotive Control window is called from the Train Control window and provides information on locomotive of any train on the track layout.

## 5 Conclusion

We have implemented the CMI application using Java framework as the most suitable target framework. Considering all the requirements analysis provided in chapter 2, PowerBuilder was not a suitable choice for the target framework since it doesn't fulfill some of the requirements, such as portability and being common in industry. Furthermore, more precise considering and comparing between Java and C#.NET frameworks indicated that portability and connection to MySQL database using Java is pretty straighter forward than using C#.NET framework.

To design the CMI application, we considered it as an application encompasses graphical user interface as one of its main facilities and logical modules to handle application data as well. Considering that, Model-View-Controller or MVC design pattern has been used to decouple the data access and logic handling from data presentation and user interaction of the application. Using MVC pattern we decoupled the application in to three different modules including model, view and controller. Furthermore within each module there have been some divisions based on the functionality and class hierarchies have been designed according to Object-Oriented design principles such as polymorphism, inheritance and etc.

For the implementation we faced some issues. One of the main issues was integrating different parts of the graphical user interface that we required in order to complete CMI application. The Track Layout which already has been done as prototype version using Adobe Flex technology and the rest of the user interface developed in Java Swing. The SDK version used to develop track layout is the third version of Flex SDK which in fact has been integrated with new Adobe product to support AIR or Adobe's desktop application runtime. The track layout has been developed using air libraries.

According to the solution for this project to embed Flex component within Java Swing interface, we tried the available libraries. We have performed some tests using the DJ Project through its NativeSwing Library. This library enabled us to embed swf or compiled file of Flex component within Java Swing interface and invoke Flex code methods. We tried to execute the Track Layout project swf file in our Java Swing interface but it didn't work. According to our studies on this subject, the only possibility to embed Adobe Flex within Java Swing is in the case of using Flex Web API, which is not the case here. The solution can be converting the existing code from air version to web. We tried to do it but because of lack of good knowledge on Flex libraries we ran out of the time.

The Remaining issues within the application are mostly related to the integration problem, for instance the functionality of menu bar or tool bar on the track layout objects, such as trains, locations and etc. Also some left issues are regarding the TCC interface module within the application, caused by lack of a real server to test the TCC interface with. On the other hand, there was not enough time to complete the functionality of the TCC server we have created. For instance, the message validating using the suitable XML schema is not performing in this version of the application.

# Terminology

CTC	Subsystem for Centralized Traffic Control.
InterFlow	The name of the complete integrated train control system, both InterFlow stationary and train borne part.
RBC	Radio Block Central. Sub-system which handles e.g. train separation and route search.
TCC	The Train Control Centre, which implements Interlocking, RBC and limited TMS functionality.
CMI	The Controller Machine Interface, the systems main user interface.



# References

1. "Java Programming Language", *CBS Interactive*, October 2005, [http://findarticles.com/p/articles/mi\\_hb3234/is\\_5\\_35/ai\\_n29240910/](http://findarticles.com/p/articles/mi_hb3234/is_5_35/ai_n29240910/)
2. D. Harms, "Industry Trends: How Important Is Java", *Clarion Magazine*, September 1999, <http://www.clarionmag.com/cmag/v1/v1n8understandingjava.html>
3. ".NET Framework Conceptual Overview", Microsoft, 2011, <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>
4. K. Radeck "C# and Java: Comparing Programming Languages", Microsoft, October 2003, <http://msdn.microsoft.com/en-us/library/ms836794.aspx>
5. J. N. Kostaras, "Java vs C#" February 2008, [http://jkost.ergoway.gr/jnkjavaconnection/java\\_vs\\_csharp.html](http://jkost.ergoway.gr/jnkjavaconnection/java_vs_csharp.html)
6. "Introduction to the C# Language and the .NET Framework", Microsoft, 2011, <http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx>
7. P. Lannigan "PowerBuilder History, Powersoft History", fall 2004 [http://www.lannigan.org/powersoft\\_powerbuilder\\_history.htm](http://www.lannigan.org/powersoft_powerbuilder_history.htm)
8. "PowerBuilder Foundation Class Library User's Guide", *Sybase*, March 2003 <http://download.sybase.com/pdftdocs/pbg0900e/pfcug.pdf>
9. "Technical Overview", Techné Knowledge Systems Inc., 2003, <http://www.techne.ca/whitepaper.htm>
10. B. Armstrong, "Say Hello to PowerBuilder 12.5", *PowerBuilder Journal*, August 2011, <http://pbdj.sys-con.com/node/1955021>
11. "PowerBuilder", *Sybase Inc.*, 2011, <http://www.sybase.com/products/modelingdevelopment/powerbuilder>
12. M. Hall, "Internal Frames", 1999, <http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/Swing-Tutorial-JInternalFrame.html>
13. G. Gnana Arun Ganesh, "Developing MDI Application in C#", January 2002, <http://www.csharpcorner.com/UploadFile/ggaganesh/DevelopingMDIApplicationsinCSharp11272005225843PM/DevelopingMDIApplicationsinCSharp.aspx>
14. S. Rodriguez, "C# SDI/MDI Application Wizards", November 2003, <http://www.codeproject.com/KB/macros/sdimdiwizards.aspx>

15. "MDI Applications in PowerBuilder .NET", *Sybase*, April 2010, <http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc01261.1200/doc/html/kwi1244748243372.html>
16. "Java Internationalisation", *Oracle Corporation*, 2010, <http://java.sun.com/javase/technologies/core/basic/intl/>
17. Kumar, Ravikant, "Globalization, Internationalization, and Localization using C# and .NET 2.0", November 2009, [http://www.codeproject.com/KB/locale/Internationalization\\_I18N.aspx](http://www.codeproject.com/KB/locale/Internationalization_I18N.aspx)
18. "Building Internationalized Applications with Sybase PowerBuilder", *Sybase*, 2005, [http://www.sybase.com/content/1036154/L02684\\_PB\\_InternationalizedApps\\_WP.pdf](http://www.sybase.com/content/1036154/L02684_PB_InternationalizedApps_WP.pdf)
19. A.P.Rajshekhar, "Socket Programming in Java", April 2007, <http://www.devarticles.com/c/a/Java/Socket-Programming-in-Java/>
20. A. Dhar, "Socket Programming in C#", July 2003, <http://www.devarticles.com/c/a/C-Sharp/Socket-Programming-in-C-Part-I/>
21. "The PowerSocket Library", January 1999, <http://www.level5software.net/documents/Pslib21.htm>
22. "SocketWrench Freeware Edition", *Catalyst Development Corporation*, 2011, <http://www.catalyst.com/products/socketwrench/freeware/index.html>
23. T. Violleau, "Java Technology and XML", *Oracle Corporation*, November 2001, <http://java.sun.com/developer/technicalArticles/xml/JavaTechandXML/>
24. A. Skonnard, ".NET XML Best Practices", *SoftArtisans, Inc.*, July 2003, [http://support.softartisans.com/kbview\\_673.aspx](http://support.softartisans.com/kbview_673.aspx)
25. "Using PowerBuilder XML Services", *Sybase Inc.*, 2007, [http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.dc37774\\_1100/html/apptech/BABJBFGG.htm](http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.dc37774_1100/html/apptech/BABJBFGG.htm)
26. "JDBC Overview", *Oracle*, <http://www.oracle.com/technetwork/java/overview-141217.html>
27. "Accessing Data with C#", *Exforsys Inc.*, March 2005, <http://www.exforsys.com/tutorials/asp.net/accessing-data-with-csharp.html>
28. M. Pastore, "Java Developers Finding a Home at Adobe Flex", *QuinStreet Inc.*, November 2009 <http://www.devx.com/HotList/HotList-Adobe/Article/42707>
29. C. Deckers, "The DJ Project", <http://djproject.sourceforge.net/main/index.html>
30. "Easy Java COM Connectivity", *University Blvd USA*, <http://www.ezjcom.com/>

31. "JFlashPlayer", *VersaEdge Software, LLC*, <http://www.jpackages.com/jflashplayer/>
32. G. Wishine, "Fun with C# and Flash Player 8 External API", October 2005, <http://blog.another-d-mention.ro/programming/communicate-between-c-and-an-embedded-flash-application/>
33. M. Piller, "Introduction to Flex 4 and .NET Integration", *Adobe Systems Inc.*, March 2010, [http://cookbooks.adobe.com/post\\_Introduction\\_to\\_Flex\\_4\\_and\\_.NET\\_Integration\\_16930.html](http://cookbooks.adobe.com/post_Introduction_to_Flex_4_and_.NET_Integration_16930.html)
34. "PowerBuilder-Internet Explorer OLE to display Flash charts", November 2010, <http://anvil-of-time.com/wordpress/powerbuilder/powerbuilder-%E2%80%93-internet-explorer-ole-to-display-flash-charts/>
35. "PowerBuilder12.0 New Features", *Sybase Inc.*, 2010, <http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc00357.1200/html/newfeat/BABICBGC.htm>
36. "TIOBE Programming Community Index for November 2011", *TIOBE SOFTWARE*, 2011, <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>
37. M. Kosyakov, "Long way to Maturity", <http://www.worksforweb.com/publications/PHP-mature-platform/>
38. B. Cohen, "How to Write Maintainable Code", March 2001, <http://www.advogato.org/article/258.html>
39. "Model-View-Controller", *Microsoft Inc.*, 2011, <http://msdn.microsoft.com/en-us/library/ff649643.aspx>
40. R. Eckstein, "Java SE Application Design with MVC", *Oracle Inc.*, March 2007, <http://www.oracle.com/technetwork/articles/javase/mvc-136693.html>
41. T. Bray, J. Paoli, C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0", *W3C Recommendation*, February 1998, <http://www.w3.org/TR/1998/REC-xml-19980210>
42. B. Bos, "XML in 10 Points", *W3C*, March 1999, <http://www.w3.org/XML/1999/XML-in-10-points.html.en>
43. "Document Object Model APIs", *Oracle and/or its affiliates*, <http://download.oracle.com/javase/tutorial/jaxp/intro/dom.html>
44. "Simple API for XML APIs", *Oracle and/or its affiliates*, <http://download.oracle.com/javase/tutorial/jaxp/intro/simple.html>

45. “Steaming API for XML APIs”, *Oracle and/or its affiliates*, <http://download.oracle.com/javase/tutorial/jaxp/intro/streaming.html>
46. “Java SE Desktop Overview”, *Oracle Corporation and/or its affiliates*, 2010, <http://java.sun.com/javase/technologies/desktop/>
47. “Swing GUI Builder (formerly *Project Matisse*)”, *Oracle Corporation and/or its affiliates*, 2011, <http://netbeans.org/features/java/swing.html>

# Appendix

## Old version Design

1. I think that I would like to use the window menu frequently

Strongly agree

Strongly disagree

1	2	3	4	5

2. I found the system unnecessarily complex

1	2	3	4	5

3. I thought the system was easy to use

1	2	3	4	5

4. I think that I would need the support of a technical person to be able to use this system

1	2	3	4	5

5. the zoom options are easy to find in this design.

1	2	3	4	5

6. I thought there was too much inconsistency in this system

1	2	3	4	5

7. It's more easy to find the tool options in this design.

1	2	3	4	5

8. I found the system very cumbersome to use

1	2	3	4	5

9. The opening window is fair enough

1	2	3	4	5

10. I needed to learn a lot of things before I could get going with this system

1	2	3	4	5

# Specification Design

1. I think that I would like to use the window menu frequently

Strongly agree

Strongly disagree

1	2	3	4	5

2. I found the system unnecessarily complex

1	2	3	4	5

3. I thought the system was easy to use

1	2	3	4	5

4. I think that I would need the support of a technical person to be able to use this system

1	2	3	4	5

5. the zoom options are easy to find in this design.

1	2	3	4	5

6. I thought there was too much inconsistency in this system

1	2	3	4	5

7. It's more easy to find the tool options in this design.

1	2	3	4	5

8. I found the system very cumbersome to use

1	2	3	4	5

9. The opening window is fair enough

1	2	3	4	5

10. I needed to learn a lot of things before I could get going with this system

1	2	3	4	5

## Alternative Design

1. I think that I would like to use the window menu frequently

Strongly agree

Strongly disagree

1	2	3	4	5

2. I found the system unnecessarily complex

1	2	3	4	5

3. I thought the system was easy to use

1	2	3	4	5

4. I think that I would need the support of a technical person to be able to use this system

1	2	3	4	5

5. the zoom options are easy to find in this design.

1	2	3	4	5

6. I thought there was too much inconsistency in this system

1	2	3	4	5

7. It's more easy to find the tool options in this design.

1	2	3	4	5

8. I found the system very cumbersome to use

1	2	3	4	5

9. The opening window is fair enough

1	2	3	4	5

10. I needed to learn a lot of things before I could get going with this system

1	2	3	4	5

## Alternative Design 2

1. I think that I would like to use the window menu frequently

Strongly agree                      Strongly disagree

1	2	3	4	5

2. I found the system unnecessarily complex

1	2	3	4	5

3. I thought the system was easy to use

1	2	3	4	5

4. I think that I would need the support of a technical person to be able to use this system

1	2	3	4	5

5. the zoom options are easy to find in this design.

1	2	3	4	5

6. I thought there was too much inconsistency in this system

1	2	3	4	5

7. It's more easy to find the tool options in this design.

1	2	3	4	5

8. I found the system very cumbersome to use

1	2	3	4	5

9. The opening window is fair enough

1	2	3	4	5

10. I needed to learn a lot of things before I could get going with this system

1	2	3	4	5