2. REVIEW OF LITERATURE

The literature review presents a general picture of the Rich Client Platform Developments context and shows which work has already been done in the field of designing projects based on it. It helps us understanding the topic and focusing on the relevant parts before conducting the case study. The literature review serves as a further introduction to the area, making them familiar with existing guidelines. The knowledge gained is crucial for designing this Earth Navigator using Rich Client Platform.
2. Review of Literature

2.1 THE INTERNET AND CLIENT SIDE APPLICATIONS

An unpredictable expansion in the use of Internet and in the features provided by internet came into picture in the mid-1990s. It has been seen widely that time. World Wide Web provides a way in which it is easy to use application and content on the personal computer which is connected to internet, is widespread adopted. In this scenario, there is requirement for an application which will make the work easier for the end user as well as boost the speed of internet also. At the same time ‘thin-client’ computing introduces in the world of Internet. This new model not only promised to lower the cost of developing but delivering applications to customers and business partners as well as end-user desktops also. Beside this it boost the range of application types that could be delivered. This ‘rich client’ model works on a very thin client based on HTML, and powerful application servers that composed and delivered ‘pages’ to web browsers dynamically.

The hold for programming embedded software still lags at the back the support for PCs. In meticulous, energetically typed languages are not yet widely available for embedded systems [10]. Side by side, there are some significant limitations and drawbacks, which we can get around the application interface’s richness. Beside this, a number of issues are there regarding content and media, and so the solutions that are built and delivered get complex. In many respects, web technology has offered ease of deployment to the traditional users in case of programming and user interface. In the late 1990s, architecture inherent imposes a lot of challenges to the web application development and deployment technology as Internet born those days.

Some other changes have been seen after 2002 in the internet like businesses and End-users were demanding more from their investments in Internet technology. In respect of, deliver true value to users, most of the companies had to look towards richer models for internet applications. Traditional desktop enriched with media-rich power and content-rich environment. For this, new models were developed to combine these changes to the web applications. At the same time, business companies are also eagerly waiting to see some growth in the use of software components which can be reused. These services were widely used that time over the network. Good functionality and data both are needed to distribute to the number of client devices through applications. There was clear evidence that there is need for rich client
applications in web technologies. So the new technology of rich clients as next-generation came into the picture in the industry.

2.2 RICH CLIENTS AND RICH INTERNET APPLICATIONS

2.2.1 The Need for Rich Desktop Clients
A rich client, also known as heavy, fat, or thick client, is a computer that typically provides rich functionality in client–server architecture or networks autonomous of the central server. Software developers would rather not have to study multiple interfaces and programming environments to effort with different vendors’ tools. This has encouraged the make use of the multivendor integrated development environment (IDE) [11]. Rich client or thick client as name implies do the most of their processes on the client side, in contrast of thin client, which is basically a computer heavily dependent on a server's applications.

A fat client as said above requires less or periodic connection to a central or network server, but is enrich of characteristics of the ability to perform a number of functions without that connection. In contrast, a thin client relies heavily on accessing the server each time the data needs to be validated or processed. Generally a thin client does as little processing as possible at the client side. Rich Desktop Applications/Rich Internet Desktop Applications (RDA/RIDA's) are the next generation of client web applications. It follows the evolution from thick desktop clients towards the thin client architectures. The preface of Rich Internet Applications (RIA) enhanced the user experience and controls, however, these web clients are still dependent on internet connectivity and browser functionality. Beside this all rich internet application have many other benefits like-

Real time interaction (dealing, monitoring)
Integration with OS (sound etc.)
100% Java counts
Ease of administration and distribution
Plain Swing maybe too plain
2.2.2 Crucial Aspects of Rich Client Technologies

Before going into details of the technical aspects of the Eclipse’s Rich Client environment, we should consider some of the crucial aspects of fat or rich client technologies. Rich client technologies should have the following things:

1. **Incorporate all the three main things into one familiar environment.**
   For the nice end user experience one is needed three things - content, communications, and application interfaces. In the Internet, the end-user experience can be fragmented into different types of works like they need the HTML browser for textual content, multiple communicating clients for performing messages, needs necessary application interfaces to provide ease to end user, and multiple media players for handling video, audio, and some other forms of media. Rich Client Application can provide all these types of interaction.

2. **Provide an high-performance, efficient runtime for executing code**
   We know that there are a number of performance related challenges in the end-user experience of web applications which are HTML based. These challenges include the need to dynamically generate a lot of text for transmission of simple data, the request-response page rendering model, the inability to easily invoke, the basic graphics model of HTML the lack of client-side data storage, and even the use of remote business logic. These all things must be improved.

3. **Re-use of components to enable rapid application development**
   Component-driven development should be supported by Rich clients to enable potential for the both third party and commercial developers and to accelerate development by using easily reusable visual components so that junior developers can access to complex functionality. For making the development easier, these components should be integrated seamlessly into the design time environment.

4. **Provide extensible and powerful object models for interactivity.**
   In supporting for interactivity, web browsers have progressed through the Document Object Model (DOM), DHTML and JavaScript, they still not able to build efficient web applications which can satisfy the end user. For this, there should be an object
model which must be integrated of system level services, communications and good user interface. So in this way Rich clients should provide a great object-based model which can develop events as well as applications.

4. **Embrace connected and disconnected clients.**

It happens that many times users need to be online if he wants to do some work in web browser. But still there are some applications which would provide the advantage from connected devices such as personal digital assistants (PDAs) and laptops and the ability to be used offline or occasionally connected. Likewise, many applications need two-way connection to support the notification-based communications. These types of connections are called persistent connections. These both types of applications can be easily built and deployed using Rich clients Applications.

5. **Enable the maximum use of web and data services provided by application servers.**

There is ability to include the user interfaces and cleanly separated presentation logic in the Rich client applications from the application logic hosted on the network. The model which is provided by Rich clients can be easily used by remote services. These remote services can be accessed as XML web services or hosted in an application server. Back-end components are responsible to provide these services.

6. **Enable easy deployment on multiple platforms and devices.**

Internet application does not bother that platform or device is using at client side whatever work is done by that application. So it is mandatory for Rich clients that they must support all the operating system which can be used at client side. This system may be PDAs, set-top boxes, smart phones, game consoles, or other internet appliances.

**Why there is need for a Rich Internet Application?**

HTML-based applications were using at that time as HTML was irrelevantly easy to learn and use, the structural design was simple, the cost of deployment was low. User interface improvements came into the picture as many users and developers are needed it for immediate access to new data and applications. The demand for desktop
computers was increasing. Some significant user interface functionality is benefited of being Web-based application [12].

Certain applications are sometimes not providing high-quality with respect of HTML. As sometimes user interaction is needed by complex applications for completion of transaction. This may lead to inadequately slow interaction. This is mandatory in some fields such as medicine and finance. This part must be a mean Consider a project management system. As it is needed on web application, can be handled as an HTML application, but some other certain works can make it better like manipulate hierarchies, schedules, and charts. For this, it is necessary that an HTML form could receive and send more-complex data structures in the form of XML documents. In addition, simple interactivity can require a lot of scripting to get the job done although HTML starts off straightforward. After all these things while an input form is carefully laid out and fully scripted, HTML forms can only send simple name/value pairs from the browser.

RIAs is known for its strong client-side rendering engines which is capable of presenting very dense, graphically, responsive rich user interfaces [13]. It offers a variety of controls like sliders, windows, date pickers, tabs, gauges, spinners, and a lot of others, so that it can allow you to construct graphics using Scalable Vector Graphics or some any other mechanism. Even full-motion animations are also offering by some RIA technologies.

Fewer round trips to the server are needed if RIA technology is using. It also provides more responsive user interface and data can be cached in the client which is not allowed in the HTML. The style is away from text-based Web clients and definitely toward rich clients for wireless and irregularly connected devices. [14]. Application can be designed to work offline which are running on laptops when connectivity is lost. For explaining form layouts and data transfer format XML is used. Access to an Oracle database is accomplished with Web service calls [15]. If it is required that data must be updated on server side in real time then the client can wait connected to the data source.

2.2.3 The Platform Runtime Core
The runtime engine is implemented by the platform runtime core. Then this runtime core starts the platform base and discovers plug-ins dynamically. The platform saves
information about the installed plug-ins along with the methods, which they provide, in its registry. The plug-in registry loads plug-ins dynamically. The end user should not misuse resources on plug-ins which are installed but never used. A plug-in can only be activated if and only if the function is provided by the plug-in is needed.

The extension mechanism mentioned above is the only way to extend the platform. Each plug-in in the Eclipse SDK is using this mechanism to contribute to the platform. The plug-ins uses no internal mechanism to achieve this.

2.2.4 Graphical user interface features

This section describes visual elements that ship with Eclipse. Responsive elements like progress bars are explained as well as key-binding support.

2.2.4.1 Look-and-Feel

The Eclipse platform ships with its own Look-and-Feel. A screenshot can be seen in figure 2.1. It consists of views and editors that include buttons for maximizing and restoring. Views and editors can be docked to the top, left, bottom or right side of the application and can even be stacked on top of each other. The fast view bar, whose size is reduced when there are no fast views, can be docked to the bottom, left or right. The platform even supports detached views on Windows and Linux GTK. The drag-and-drop feature gives feedback about items that are dragged.

The Welcome Page Eclipse supports welcome pages which show new features, provide tutorials and introduce the user to the product. Due to the reason that welcome pages are pluggable, a Rich Client application can define its own welcome pages.
2.2.4.2 Responsive User Interface

To give the user feedback about the current status of the platform, a couple of widgets have been introduced in Eclipse. For example support for running jobs in the background. The advantage of running jobs in the background is that they do not block the user interface [16].

2.2.4.3 Scalable User Interface

The platform allows Eclipse based products to scale down user interface elements to show a clearly arranged user interface. So-called capabilities make the filtering of currently unneeded functions possible and the OSGi runtime allows to dynamically adding user interface elements. It is even possible to conditionally contribute to menus and toolbars. Additionally, some menu or toolbar entries become only visible when the user selects a certain resource or element. The Run button for example is only visible when the user selects an executable Java file.
2.2.4.4 Editor Management

The editors maintain single and multiple tabs. The single tab option is especially useful when there are many files of the similar type open or the keyboard is used to navigate through the single editor instances. Editor pinning is used when the number of editors that can be open at once is limited and some editors should not be closed.

2.2.4.5 Key Binding Support

Additionally to the platform itself, key-bindings are also supported in dialogs. Multi-key shortcuts are also existing and used throughout the platform to improve the work practice. Plugins can contribute new key-bindings to the platform.

2.2.4.6 Themes

The Eclipse workbench has built-in support for themes. Although it is not fully customizable, the colors and fonts that are used in the application can be altered.

2.2.4.7 Other standard widget toolkit Specific Features

Eclipse comes with support for multiple monitors, non-rectangular windows, system tray icons, tiff image decoding, colored cursors, drag-and-drop support and colored table cells. These features make the power of Eclipse obvious.

2.2.5 Workspace, Workbench, Editors, Views and Perspectives

The *workspace* is responsible for collection of the user’s resources, which can be folders or files and which can be linked to a directory or file under the workspace directory of Eclipse. The workspace also holds the history of its files, so the user can track changes of his documents and revert to a previously saved state. It also informs registered listeners about resource changes [17].

The *workbench* is the graphical interface of Eclipse. It is built using standard widget toolkit and JFace, so it looks and behaves like a native application. It is structured into perspectives which contain views and editors. Views and editors are used to work on resources which are provided by the workspace. The user is allowed to navigate between the folders and projects by using workbench. The main purpose of the workbench is to provide navigation. Everything else is delegated to other plug-ins.
An editor is associated with a certain file type. So a different editor shows up for text files than for Java files. The Java editor for example provides syntax highlighting and other features, which are useful when working with java files.

Perspectives are basically used to arrange the views, which are provided by a certain feature, for example the JDT, in an ordered way. Each perspective is associated with a task, for example Java developing, debugging or fetching files from a Concurrent Version System (CVS) repository. All views which are useful in application for a particular task are arranged on the workbench.

2.2.6 Extensions, Extension-Points, Plug-ins and Fragments

Eclipse lets developers modify and extend user interface components (or plug-ins) via so-called “extension points,” a grouping of Java interfaces and XML markups that define component interfaces and facilitate their loose coupling [18]. An architectural pattern is represented by the Eclipse plug-ins for building an application from single components, which are objects that may be configured into a system at deployment time. For providing configuration and management support, a plug-in is embodied in other plug-in runtime class.

An Eclipse installation includes a plug-in directory, which contains all installed plug-ins. A manifest file named plugin.xml describes this plug-ins.

Fragments are another feature which provides additional functions to plug-ins. They are used to add functionality to a plug-in after it has been installed. When the platform finds a plug-in and a fragment it merges the extensions and extension-points of the fragment with the ones of the targeted plug-in. A simple fragment does not differ from the concept of a plug-in except for it cannot extend the import statements of the plug-in manifest file, which are used to declare the dependencies to other plug-ins. How and where a plug-ins may contribute to a plug-in is defined by Extension-points. It is important to use the appropriate extension-points when extending the platform. The plug-in manifest file describes the connections between the extension-points and their extensions. The extension point tag in a manifest file defines a new extension possibility; the extension tag describes a new extension to this extension-point.
2.2.7 Standard Widget Toolkit and JFace

The Eclipse IDE is built upon the foundation of standard widget toolkit, which delivers the richness and responsiveness of a native application. As it handles this in an operating system independent manner, it can be used across Unix™, Windows™ and other platforms.

Standard widget toolkit uses a procedural navigate layer to the operating systems (OS) graphics application programming interface, which enables standard widget toolkit to control the native widgets. With this feature, a minimum of native code is needed to reflect any changes to the look-and-feel feature of the operating system which is using for the running the application.

Drag-and-drop support is another big problem of Swing’s implementation. It was very buggy and did not work very consistent and easily. standard widget toolkit solves this problem because it works directly with the operating system’s drag-and-drop application programming interface and reflects therefore any improvements in the drag and drop behavior of the operating system immediately.

2.2.8 Eclipse Forms

The problem with layout managers and standard widget toolkit widgets is that they have been designed to solve either the situation when controls should appear in a dialog, which are for example preference pages or property pages, dialogs, message boxes, wizards, or when they should appear in an editor or a view. When displaying controls in a dialog, the Grid Layout is used because a dialog usually has a fixed size and the designer can layout the controls very precisely. These controls use the default fonts and colors which are basically provided by the operating system. There are controls like trees, tables and text areas which usually show up in the content area and can adjust themselves to the size of this area. But during the development of Eclipse, a third option appeared. It should be possible to use the content area to view dialog-like content. The reason for this requirement is that usually the content area is quite large and can therefore hold many elements on one page. One can even open more than one editor at the same time whereas dialogs are usually singletons. The use of editors allows the attachment of views that can be used to show for example the outline of the current settings in the form.
Using traditional layout managers and controls in a form is problematic for various reasons, because of their default dialog background; standard widget toolkit controls look strange in the content area. The Grid Layout does not scroll the composites, which are placed in one of its cells, so a composite that is larger than the provided space is clipped. Another point is that components like hyperlinks or text in an HTML form are wrapped when they are larger than the provided horizontal space. Custom programming is required to accomplish wrapping with existing standard widget toolkit components and layout managers.

This situation led to the development of Eclipse Forms. It provides a concept of a form that can be used in the content area. Eclipse Forms also handles the colors, fonts and hyperlink groups and has its own layout manager, which uses an algorithm that is similar to the one of HTML tables. Other features are custom controls and a multi-page editor; there each page is considered as a form.

In Java programming the most common layout manager is the Grid Layout. It is similar to the HTML table layout, which is often used to design complex websites where one needs precise layout of graphics and fonts. For example when a designer cuts a graphic into pieces and wants it to show up as a whole in HTML. The Grid Layout is very flexible and can be used in many situations, where other layout managers are not powerful enough to accomplish the specific task.

But when it comes to forms, the Grid Layout has various disadvantages. The Grid Layout asks the component that it should compute its width and layouts the component according to the result. But it does not pass the size of the client area that is currently available. Therefore the component has no possibility to calculate its width and height according to the available space.

It will always return the width that is needed to render in one line, and that may be more than is available at the moment. So a hyperlink component will not wrap, but force the user to scroll to see the whole content.

The solution to this problem is the newly introduced TableWrapControl, which is an alternative but the TableWrapLayout does not fill the vertical space like Grid Layout does. It works top down and when it places all the controls it is done. So there may be
some empty space left. Grid Layout tries to fill up this space by enlarging some of its controls, especially the ones with the GRAB flag set.

The ColumnLayout, the second layout manager that is introduced in Eclipse Forms, is similar to the in the Java world already existing RowLayout. The main purpose of this layout manager is to arrange the controls in a couple of columns. The amount of these columns is depending on the number of controls and the available space, so the number can increase or decrease depending on the width of the client area. Another purpose is to distribute the controls evenly on the available columns, so the last column should be completely filled too. All these issues are addressed by ColumnLayout but not by GridLayout, where the number of columns has to be chosen in advance, and not by RowLayout, where the last column would not be completely filled.

2.2.9 Wizards
Wizards automate repetitive and complex tasks through a user dialog that provides special assistance. In Eclipse, wizards can create, import, and export resources (files, folders, and projects). For example, the Java New Class or Interface wizards help one create class or interface files with the proper package importing, constructor declaration and inherited method definitions. The Eclipse platform contains many wizards, but it also makes it easy to write own customized wizards. Plug-ins can create and launch these customized wizards.

2.2.10 Help System
Help system based on an XML table of contents referencing HTML files is included in the Eclipse Platform. It provides online documentation and context-sensitive help. Moreover it allows adding one’s online documentation without any programming required.

2.3 USING ECLIPSE
Eclipse is open source so the developers typically want to work as part of a team and share their code with other members of their team. So Concurrent Versions System (CVS) is one of the important parts of the Eclipse which ships as part of Eclipse. It is very easy to create a Java Project and class and its execution, debugging and testing
2. Review of Literature

on the Eclipse platform [19]. Eclipse is not only a dominant integrated development environment (IDE) for Java but it allows development in other languages also for professional software developers. The complexity of Eclipse boundary and the associated computing environment can be a big deal for beginners as Eclipse is improperly suited for use in introductory computing education [20]. Eclipse is known as a fully featured professional Integrated Development Environment (IDE) for Java as it provides a lot of advanced features for development in this language. For those who have no programming background may be overwhelmed with these features, particularly [21].

2.3.1 How to work with Eclipse

As Eclipse provides a common User Interface model and considered the most popular Java IDE, it provides the both working tools as well as rapid development of modular features which are based on a plug-in component model. The main thing which is keep in mind at the time of Eclipse Foundation designing is that this platform should be platform independent means it should be capable of working with multiple operating systems, including Linux, Macintosh and Windows. Eclipse works as both as open, extensible application framework for building software as well as an open, extensible development environment upon which software can be built. Beside this, this platform should provide robust incorporation to rich clients. Again, these rich clients should support the Graphical user interface interactions like navigation, drag and drop, customization and cut and paste which are familiar to everyone and needed by everyone. And finally when this platform designed completely, it was providing all the features which are described here. It may be overwhelming to the novice programmer as it has developed with a lot of functionality. Nevertheless, there is no doubt that once programmers become recognizable with Eclipse environment in the manner of programming, they must find it empowering once and useful [22]. Eclipse can be thought as a “design centre”. We can ask a query at the time of developing our own software from a development team of 500 or more developers who developed this platform and supporting it. In fact, Eclipse is so powerful that NASA used Eclipse Rich Client Platform to build a software program for managing remote vehicles on space missions named as Maestro. Eclipse is not only used for creating Integrated Development Environment but it is also helpful for constructing general-purpose
applications also as Eclipse is designed as a universal tool-integration platform. It can be used for development of contact management systems, work how, help systems.

Although Eclipse is written in Java, it supports any language. The Eclipse platform can be considered as a hub of functionality to provide plug-in modules and development tools. Usually, in the platform’s workspace, a developer can mark a given tool as a separate plug-in that operates on files [23].

2.3.1.1 Getting Eclipse

As everybody knows that before using any platform, one has to download that particular platform, so as before using Eclipse, we need to download it. After downloading we install it and set it up for development.

The official web site for Eclipse is http://www.eclipse.org/. On-line resources like newsgroups, articles, mailing lists, bug tracking with the latest Eclipse news and links can be seen here on this web site.

Here on this website the latest version of Eclipse can also be downloaded from its download page. A number of mirror sites around the world are also available. So downloading Eclipse is not a big deal. There are some integration or nightly builds are also available but these are required when someone is involved in the development of Eclipse itself.

A large number of platforms are supported by Eclipse including Linux, Windows, HP, Solaris, Mac OSX and others. So there are various notes concerning that release and various links regarding each platform version are available at download page. For downloading the Eclipse choose the Eclipse SDK download link as per the platform. This is generally being a very large file near about more than 60 MB [24].

You might also want to download the Example plug-ins file corresponding to your platform. For the same, the download page has a number of other download links. You should ignore the rest of them unless there is not a specific need for one of the other downloads.

Java Runtime Environment
Although Eclipse is a Java program, and used for Java development but after that it does not include the Java Runtime Environment which is necessary to make it run. Any JRE 1.4 and above is sufficient for running Eclipse 3.0. As Eclipse is used by most Java developers, they already have a suitable JRE installed on their machines. This can be installed by downloading and installing one from [http://java.sun.com/](http://java.sun.com/).

2.3.1.2 Installation

After successfully downloading of Eclipse ZIP file, it should be unzipped on the hard drive. It does not matter where Eclipse is installed as Eclipse does not modify the Windows registry. If any of the Eclipse examples is downloaded, it should be unzipped at the same location.

2.3.2 The Eclipse Workbench

For working with Eclipse, we have to start Eclipse. This can be done by double-clicking on the eclipse.exe file in the C:\eclipse directory. It displays a dialog box at the time when Eclipse is firstly launched, and in between it completes its installation process. It creates an initial default workspace directory below the main Eclipse directory. After all this work finishes the main Eclipse workbench window appears on the screen. When first time Eclipse is launched, welcome page is opened automatically. Eclipse workbench window consists of a number of tiled panes which are known as views and editors with main menu bar and toolbar.

![Eclipse Workbench](image.png)

Figure 2.2 Eclipse Workbench
2.3.3 Perspectives, Views & Editors

A Perspective can be seen as a visual container for a set of Parts. Collectively, the various views like Outline, Navigator, and Tasks and editors which are used to work with various resources. And finally these various views which are visible within the Eclipse workbench, known as a perspective. Perspectives are mostly used by the Eclipse IDE to configure the menu and the toolbar and arranging the Parts for different development tasks. Perspectives help Eclipse to work in a managed way and perfect manner. Eclipse workbench is group of perspectives. In Eclipse, it is possible to open multiple perspectives at the same time and an icon for each perspective is visible in the vertical toolbar while various perspectives are opened. Each of them can be viewed on the left side of the workbench window. The title bar of the window shows the name of active perspective and its icon appears pressed. The main perspectives which are mostly used in the Eclipse IDE are the Debug perspective for debugging Java applications and the Java perspective for Java development.

Any changes made within a view are saved immediately. The main work of views is to modify properties of a resource and navigate resources. In contrast, to view a specific resource editors are used and this resource can be modified also. Basically the common open-save-close model is followed by Editors.

In a given perspective any number of editors of the same type can be opened at one time but only one instance of any view can be opened. So there is a set of views for each perspective but they can share some editors that are open editors which can be shared by all of the open perspectives. Currently active view or editor is highlighted by Title bar. Any global actions like cut, copy or paste is received by the view or editor. When Navigator view is clicked, its title bar becomes highlighted showing that it is active and the title bar of the welcome page turns white indicating that it is now inactive as we know that in windows the active window shows its title bar in different color.

Since Eclipse uses a different type of display that is tiled display for each of its panes, it makes one pane larger and makes its neighbors smaller and vice versa. We can also resize the Views and Editors panes by dragging the sizing border that appears on each side of the pane of Views and editors.
2. Review of Literature

By dragging the panes individual title bars, the panes can be moved around. It is also possible to drag a view on to another view, and in that case the two views will stack up in the manner to indicate both of the views. View can be bringing to top of the stack by selecting that particular view. The view occupies a portion of the available area if a view is dropped into the sizing area between views. In this way it inserts itself next to the view that previously occupied that space. For providing the space for accommodation of new view, the views that originally occupied that space shrink in size.

If a view is dropped into the left-hand tool bar area, the view docks as a fast view there. If there are some views which are not needed to be seen all of the time, then Fast views are good choice but they require a great size of screen area when they are visible on screen. Fast views expand to overlap most of the window area as Fast views remain docked to the tool bar till the icons clicked upon.

When the workbench first opens only a few views are initially visible, although there are many different views are defined within Eclipse. By selecting the Show View command from the Window menu, views can be added to the perspective. A list of all of the views which are defined in the system can be seen and chosen from the other… command.

2.3.3.1 Resource Perspective

Perspectives are most likely used while developing plug-ins. Here is review of various perspectives. Resource perspective is the initial perspective which has been shown in the workbench window. The Navigator view is the primary view with the Resource perspective. The Navigator presents a hierarchical view of the resources which are loaded in the workbench like files, folders and projects. Various viewing and filtering options are provided by the Navigator as it has its own tool bar and view menu.

2.3.3.2 Java Perspectives

Sometimes we need to look at the resources of the system, for this we are having Resource Perspective. For the development of the Java Code, Eclipse provides us two Java Perspectives for different uses. These are -

1. Java Perspective
2. Java Browsing perspective
First one is Java Perspective. We can use this perspective through selecting Java command from Open Perspective from windows menu. The view of the Java Perspective is something like Package Explorer. It provides best view for the Java files and resources. These all things can be viewed in well manner in Package Explorer. It provides completely different view which is provided by Navigator View, which just shows all java packages as nested folder. Rather than showing like this, Package Explorer shows hierarchy of each package considering them separate element.

Second Java Perspective is Java Browsing perspective. We can reach this perspective through by clicking on Java Browsing command from Open Perspective from Window menu. As explained in previous section, we can view the packages in the package view. But how could we see the list of view of project, for this we are having Java Browsing perspective. Selecting a type shows its members in the members view and selecting a package shows its types in the types view.

2.3.3.3 Debug Perspective

We are here discussing about perspectives, but till now we have talked about perspectives are useful for the purposes of editing resources and writing codes. But the most important thing in programming is debugging. Eclipse is also having the perspective for this work i. e. Debug Perspective. This perspective is used commonly. We can access this perspective through the Window -> Open Perspective -> Debug command. As the name implies, Debug Perspective is basically used to check java programs and helps in finding the errors in coding. Beside this code correction, Debug Perspective is used to inspect the values of variables which are associated with our code and also used to set breakpoints.

2.3.4 Actions

For representing the various commands or actions Eclipse includes a large number of menus and tool bar buttons beside views and editors which are usually uses the bulk of the display area.

2.3.4.1 Top-level Menus

There are eight top-level menus in the basic Eclipse menu bar. Depending on which perspectives and views are using and which add-on tools we have loaded, some additional menus can also be seen there. Those are –
1. File
2. Edit
3. Search
4. Navigate
5. Run
6. Project
7. Window
8. Help

1. File menu
The File Menu provides refresh resources relative to their associated disk files, actions to create new resources, some other features like save, close and print resources, inspect the proper-ties of resources, import and export resources, and exit from the workbench.

2. Edit menu
The Edit menu provides actions to work with the open resources which can be viewed in the editor area. Standard functions such as cut, copy and paste are included here as well as functions such as find and replace, delete, select all.

3. Search menu
The Search Menu provides access to search tools in whole workbench such as Java search, global file search, and plug-in search and help search.

4. Navigate menu
The Navigate Menu provides commands to drill down into resources and then navigate within them. It also provides actions to traverse the resources loaded in the workbench.

5. Run menu
It includes an External Tools option that allows running any random external tool on the resources within the workbench. The Run Menu contains perspective-specific items which help in running or debugging the Java applications.

6. Project menu
You can close any open projects, open any closed project, and manually build all of the projects in the workbench or either an individual project. The project menu provides actions to manipulate the projects loaded in the workbench.
7. Window menu
The window menu allows customizing the current perspective and access preferences for the entire workbench. It also includes items to open different perspectives, open additional workbench windows and add any view to the current perspective.

8. Help menu
The help menu provides general help on the entire environment access to various tips and tricks, information about the current workbench configuration with the software updates.

2.3.4.2 Context Menus
We have discussed all the menus in previous section. But there is one menu left, that has been opened at the time of right-clicking i.e. context menu. This popup menu can be viewed by clicking on any view or editor. The contents of the menu depend not only on the resources that were selected at the time but also on the view or editor that was clicked. For example, as if we select context menu from the navigate view we select nothing. Context menu will be different when a Java file is selected in the Package Explorer in the same way context menu will be different if we select contest menu when a file is selected from the Navigator. Beside this some options only appears in some particular views and under some specific circumstances, for example refactor Submenu.

2.3.4.3 Tool bars
When using the Resource perspective, the standard tool bar shows the items for creating new files, running external tools, saving and printing resources, navigating recently accessed resources in browser style and accessing the search functions, which are standard toolbar items.

When right-click on a view is done, context menu that appear and the tool bar items that appear are depend on which editor has been focused and which perspective is in use. Common and standard items appear first on the tool bar and any other editor specific items appear at the end.

As we switched to the Java perspective, it will cause to appear several new items with the help of Java applications. And by using these perspectives creating new Java projects, files and packages become easier.

2.3.4.4 Customization of Available Actions
The items which are appeared upon toolbar and main menu bar can be controlled up to one extend. We can use perspective customization dialog to enable or disable the commands which are part of actions sets. Current perspective can be customized by clicking on Customize Perspective command in Window menu, which opens the Customize Perspective dialog. On the Commands page, the tool bar and menu command groups can be viewed after this. On this command page, command groups which are wanted to keep can be checked and all others can be unchecked.

2.4 REASONS FOR ECLIPSE’S SUCCESS

There are a lot of editors are available for java in the market, many of them provides good java IDEs. But among all of this the popular java IDE is Eclipse. It is permeably used at Asia-Pacific, North America, Europe, and the Middle East Africa. It means that it is having some special features that are why it is so popular.

Eclipse Foundation’s Skerrett stated “The power of Eclipse is the frequent platform that you can integrate different tools into”.

Eclipse is having a great feature that it is not dependent of any company. IBM has release the Eclipse and this technology has been made independent of any company by Eclipse foundation. It is not tied to a specific vendor. This is a big reason that eclipse is adopted so broadly.

The director of product marketing Borland’s, Rob Cheng stated “The more independent Eclipse is the more comfortable companies and developers feel using it”.

2.4.1 Lower costs

Development users are getting some tools free of cost and sometimes they pay for some tools. One relaxing thing is that Eclipse is providing all its tools completely free. Eclipse development platform is absolutely free of cost whereas other proprietary IDE systems like JetBrains’IntelliJ, JBuilder, JDeveloper and IDEA are costing up to $3,500 each. Use of Eclipse also reduces the cost of plug-ins as Eclipse is free. Beside this, the popularity of Eclipse makes many developers to work with it for developing their tools and develop tools which are compatible with it.

Borland’s Cheng said about Eclipse that Eclipse helps in Fast-moving innovation and development in a very transparent process. This is because Eclipse has made the
builds available for public download. Most of the plans, milestones and communications are also made public. Eclipse provides short-term builds in every month or every couple weeks so that people can work with it and can provide feedback timely. A lot of community involvement is there. Developers can access the source code and if he wants to modify the same he can do it as per his needs as Eclipse is an open source platform. In this way Eclipse provides a quick way to meet user’s needs for his development.

2.4.2 Elegant architecture

Eclipse foundation is a powerful foundation but it provides basic features also. This foundation provides a modular IDE with small, elegant architecture that is Eclipse platform. It provides integration of application without having the burden of dialog boxes, text boxes, drawing buttons and other widgets and property pages. So in this way, on elements of the program, developers have full control [24]. This helps the developer for creating new elements which they want to code for them.

2.4.3 Eclipse projects

Eclipse project is a bundle of binary files, source codes and its configuration. For doing some specific task, these all files are combined. In this way, buildable and reusable components are formed as these files are grouped. Projects in Eclipse cannot contain other projects as this makes hassle. For describing the purpose of the project, Natures can be assigned to Eclipse project. In the .project file in the project directory, Natures for a project are defined.

2.5 Environment Settings

This section put light on that how to do customization of Eclipse environment when various Eclipse preferences changes. For customizing Eclipse preferences, Preferences command from the Window menu is selected. A number of individual preference pages are opened on the left side of the dialog when this command opens the Preferences dialog. And then these preference pages grouped together in the hierarchy pane. Java preferences could be found in the Java group while General workbench preferences could be found in the Workbench group.
Preference dialog permits user to access hundreds of individual preferences. Values of preferences can be changed accordingly here and finally these values can be locked by clicking on the Apply button. User can continue with setting other preferences if he wants to do some other changes. After all the changes have been made, this dialog box can be closed by clicking the OK button and changes are locked for particular preference. Restore Defaults button is also there to reset the preferences on the current page to the system default values.

2.5.1. Workbench Preferences

Most general Eclipse preferences are provided by Workbench category of preferences. Here are some features of this Workbench Preferences -

- The Appearance page is used to determine whether view and editor tabs appear on the bottom or top.

- The Workbench page is used to provide various global build options and it determines that resource requires single or double-clicking for opening.

- The Text Editors page provides a variety of options which is useful for controlling the complete appearance of editors like the various item colors and annotations, current line highlighting and visibility of line numbers.

- The Editors page provides the options which are useful to control that how editors are opened and closed and at one time how many editors may be opened.

- The Fonts page contains the options for customizing the fonts which are needed at different places like for standard text font, header font, dialog font, and many others which are workspace elements.

- The File Associations page provides the association for different editor types. It provides the both internal and external association with different file types. For example, if it is needed to associate Microsoft FrontPage with HTML files.
2. Review of Literature

2.5.2 Some Important features of Eclipse

2.5.2.1 Resource management (workspace)

Some application programming interfaces are defined here to create and manage resources like files, folders and projects that are produced by tools and kept in the ale system.

2.5.2.2 Platform Runtime

The main feature of Eclipse is that it dynamically discovers plug-ins and in a platform registry it also maintains the information about the plug-ins and their services. OSGi framework is used to implementing the runtime. Plug-ins are loaded and launched, according to user operation of the platform, when it is required.

2.5.2.3 Workbench of Eclipse user interface

Workbench of Eclipse user interface implements the user ring for navigating the platform. It defines extension points so that user interface components such as views, menu and actions can be added to it. It supplies additional toolkits like JFace and standard widget toolkit for building user interfaces. Plug-ins which is Integrated Development Environment centric can define some additional function for the manipulation and navigation of the resources. Rich client applications that is independent of the resource management and workspace model Uses user interface plug-ins for building it. The user interface services are structured so that subsets of the user interface plug-ins to be used to build it.

2.5.2.4 Team support

It defines a programming model like a team for the versioning and management of the resources.

2.5.2.5 Help system

For providing help or other documentation as books it defines extension points for plug-ins.
2.5.2.6 Debug support

It defines user interface classes for building debuggers and launchers and a debug model which is language-independent.

2.5.2.7 Other utilities

Beside all these utilities, it also provides some other utilities. Some of these are performing builds using XML configuration files, plug-ins supply functions such as comparing resources and searching and updating the platform from a server dynamically.

If developer wants to develop application which can be run perfectible with all the operating system and portable also then Rich client platform on Eclipse is best choice.

Some plug-ins is generally provided by Eclipse and some plug-ins is designed by users by themselves. The platform user interface provides a standard model for navigation and Eclipse platform handles the logistics of finding and running the right code. With the help of these things developers can focus on designing and developing plug-ins that perform important business tasks such as diagramming, reporting, workflow, processing, publishing and any other business requirement.

2.6 ECLIPSE RICH CLIENT PLATFORM ARCHITECTURE

Eclipse provides rich client environment for applications and internet contents that will radically improve the quality of applications which are used by end-user. This makes the internet more relevant and useful for consumers and businesses [25]. With the introduction of Eclipse, Rich Client Platform is leading in the next major innovation in client-side applications and internet content.
In addition, Eclipse helps developers in customization and extending user interface components or plug-ins via extension points which is a mixture of XML markups and Java interfaces that facilitate their loose coupling and define component interfaces [26]. Eclipse’s Rich Client Platform includes an application shell with user interface facilities such as toolbars and menus and provides a parallel framework. A module-based application programming interface that lets developers build applications on top of this shell is also offered by Eclipse.

RCP is an IBM’s Eclipse integrated development environment (IDE) and it is a standard platform for building Integrated Development Environment, non-Integrated Development Environment, and even non-Graphical user interface applications for the server side as well as client side. Many applications like JP Morgan, Adobe, NASA, IBM and Swiss Rail all use RCP components for constructing version control products and spreadsheet software, to manage topological railroad network data, to store and retrieve images from the Mars Rover mission, and to write different types of software. Beside this, RCP is the basis for open source software such as the geographic information system (GIS), BitTorrent Client and Vuze [27].

To apply all three of the above described Data Capture projects, the Eclipse Rich Client Platform was chosen as the framework as it provides all the functionalities
which are provided by these projects. Choosing the Eclipse Rich Client Platform for these projects allows the sharing of knowledge and software components between these three projects. Some features of the Eclipse RCP are as following which make it a good choice for implementation of the Data Capture projects:

- Eclipse RCP implements a plug-in infrastructure which helps in structured Java development based on the OSGi standard.
- Eclipse RCP runs on a wide range of operating systems – Windows, Linux and Mac.
- It includes installation from both local and remote repositories and plug-in management.
- It provides support for flexible user interfaces and also have inbuilt window management.
- It also provides inbuilt support for deployment, packaging, configuration, preferences and help.
- A large developer community supports it with many contributed plug-ins and examples [28].

2.6.1 Eclipse RCP Overview

Eclipse is an open-source development environment that is very popular with professional Java developers [29]. As a modular IDE application Eclipse was originally started. The boom came in 2004 when the Eclipse 3.0 version released. This version supports that the Eclipse platform could be re-used to build standalone applications. These applications can be based on the same technology as the Eclipse IDE. The Rich Client Platform is used to build random applications in miscellaneous areas that include automotive, banking, medical, and space exploration that have nothing to do with software development. All these features makes Eclipse Platform is more than just a foundation for building development environments. For building random tools and applications it is a great foundation. Eclipse RCP is an excellent platform for developing applications that work in combination with application databases, servers, and other backend resources to deliver a rich experience to the user, as the name rich client implies.
The Eclipse Platform is turned into a C/C++ IDE by adding C/C++ development components and it is turned into a Java IDE by adding Java development components for example the JDT. It becomes together C/C++ and a Java development environment by adding up both sets of components. Some individual tools are integrated by Eclipse Platform as single manufactured goods providing a rich and consistent experience for its users. Eclipse Platform constructs a tool or application on top of it to enable the tool or application so that it can integrate with other applications and tools which are also written using the Eclipse Platform. One of the main aids of the Eclipse Platform is comprehended by its use as an integration point.

Integration is one of the main parts of the rich client space. For the ease of development, an organization can split up the application components into several parts across different development teams. And when different teams complete their work, all the work then integrated to get the results using the Eclipse Rich Client Platform. It does make the integration easier only and doesn't underestimate the process of developing large scale applications [30].

Eclipse applications and Eclipse RCP applications uses the plug-in architecture for development of projects. For creating of desktop applications, Eclipse RCP is basically used as it is flexible and extensible to use Eclipse Platform. A small installable and deployable software component is used by Eclipse which is known as Plug-in. A plug-in is a complete collection of files with a configuration file MANIFEST.MF which describes not only all the things about plug-in but all its dependencies also. Eclipse allows Eclipse and Eclipse RCP applications to get extended by third parties with the help of the plug-in architecture.

2.6.2 Advantages of Eclipse RCP

Eclipse RCP uses native user interface components to provide as much as possible a native look and feel. It also allows designing component based systems as it is having strong modularity approach. All these things are the basis for the most successful Java IDE and therefore very stable and broadly used.

Tapping into the Eco-system of Eclipse allows easy searching required resources and information which is needed. Eclipse RCP also nurtures a large community of
individuals which provide extensions, information, and support to the Eclipse framework.

As Eclipse is flexible, fast and continues to evolve, many companies like Google and IBM use the Eclipse framework heavily for their products.

2.6.3 Architectural view of Eclipse Rich Client Platform

At the core of eclipse, the thing which resides is Plug-in. It is the main functional unit. It helps Eclipse in discovering, loading, and running the components. The plug-ins determine whether it operates as an IDE or a general-purpose application and the platform functionality. The IDE of Eclipse is divided into several layers, each of them is having a small plug-in. The main IDE i.e. Java IDE in this layered structure, is layered on top of a general purpose tools platform. And this general purpose tools platform is again layered on the framework of Eclipse RCP. Eclipse RCP includes services such as the windowing system, preference pages, the help system, and an update manager. It also provides plug-ins that forms a very basic application infrastructure. The objectives for the development with the help of the Eclipse RCP platforms are to:

1. It supports widgets which are native-user interface elements instead of following them on supported operating systems.

2. It supports platform independent development so that it can work fine with all the available platforms as Windows, Linux, Macintosh and other operating systems.

3. It reduces the start-up time of plug-ins and components.

4. It reduces the use of memory so that it can be used for most critical things.

5. It reduces the designing efforts of an application by using the reusable components which are modules which are already developed and provides the help.

6. For general application development, it creates reusable modules for further usages.
Eclipse plug-ins is loosely coupled which is a programming aid and in this way developers can freely use them in different configuration. Using this plug-ins, help in solving the particular problem of the project. The Eclipse team thus decreasing development time with the grace of the project as it is building world-class applications while it is not compromising.

![Eclipse RCP Diagram](image)

*Figure 2.4: Eclipse’ Rich Client Platform*

There are five fundamental parts of Eclipse RCP-

1. user interface Workbench
2. Standard Widget Toolkit (standard widget toolkit)
3. JFace
4. Platform Runtime

5. Open Services Gateway Initiative (OSGi)

1. **User interface Workbench** It is most important part of the Eclipse RCP. This part is enriched with views, perspectives and editors.

2. **Standard Widget Toolkit** *(standard widget toolkit)* is provided in a graphical library and is a group of graphical user Interface components of java. These components are enriched with a special quality that they can hide differences which resides between platforms or operating systems because of their native implementation. This feature provides a great benefit to use single application programming interface for all the platforms for the developer.

3. **JFace** a Graphical user interface is used to display objects on the top of standard widget toolkit.

4. **Platform Runtime** It loads and initializes in the basis of the extension point model. It also provides loose coupling between the plug-ins.

5. **Open Services Gateway Initiative (OSGi)** it is a developer friendly framework which provides the facility of loading and unloading plug-ins. It is not required to restart the application for the same if OSGi is using. It also helps in management of lifecycle of plug-in.

2.6.4 **Eclipse user interface Workbench**

The perspective where user sees the application window, he finds there a lot of things like views, editors and actions. It provides the structure in which an application interacts with user and it gives good experience to user. As we know that java is a platform independent language so as this user interface Workbench. These all things can be arranged in a manner in which user wants. This facility is provided by Eclipse user interface Workbench. While using this workbench, developer feels free to reinvent a high performance user interface, which runs comfortably on different operating systems.
We can use editors to work with domain objects, these objects can be used in the forms of records, and these records can be about of anything like employee, person, and organizations. Editors use a general phenomenon of open-save-close, generally file systems use.

As we have discussed that editors work like as file system but editors work in more tightly integrated manner into the workbench. Editors can be helpful in contributing actions to the workbench toolbars and menus.

If the information related to the task which is currently performing by the user is needed then Views are helpful in providing this information, such as it can provide a list of objects that can be edited by developer at the time when user is editing any object. So we can say that a view is helpful for an editor about the document which is being edited by providing the complete information. Editors have complex lifecycle then the views. Beside this views have one good thing, that is, if we modify anything in a view, it has been saved immediately, and all the changes regarding this modification can be seen immediately at other parts of this user interface. We can also define user commands independently in user interface by using actions at their exact location. An Action is just a command which works like a trigger. It executes the given command by pressing the button or by selecting the menu item or any toolbar item. For the presentation of action, action must know the user interface properties like tool tip, label, icon etc. which are necessary for the construction of appropriate widgets. By doing this separation we can easily change the action without changing the code for the same action, not only this, we can also use the same action at different places of user interface also.

For the organization of views and editors we need perspective. For a particular user job, these all views and editors are arranged. These editors and views can be stacked or tiled on the screen also. For viewing, set the layout and action, all the responsibility is of perspective. But if user wants to control all these things in his own level, he can do it by opening the perspective. He can do whatever he wants; he can reschedule and modify a perspective. All these things are so suitable for the user, that if he wants to toggle to an alternative perspective for some other task he can immediately switch. Eclipse is so flexible that in a single window, a number of perspective can be opened,
which is easily controlled by user. There are a lot of visual design choices available to user so that he can organize user interface easily.

In these visual design choices, it provides tool bar, menu bar, status line and shortcut bar with several color and color gradients combinations to provide distinctive and integrated appearance. In the title bar of views they have titles with actions. And workbench has File, Edit, Help as well as Window lists. Editor area has part of every perspective. For a general purpose, simple application, every element of user interface is visible; it may be concealed if needed by application.

There is one more thing which is important that user has capability to modify and organize. It can be understand by the example that by simply dragging and dropping, user can rearrange the information of a person which is appearing on the screen, in his own way, he can do it.

There are some new functionality defined into user interface’s paradigm of editor’s view and perspective. For providing the facility of customizing their location, appearance and context in the application window provided for the user, all the views and editors implements access common API’s and also implement common interfaces. By adding new actions, views, or editors extension point, it allows plug-ins to extend the workbench for the enhancement of the development.

By use of standard widget toolkit and JFACE we can implement application programming interface for a user for native platform experience. We do not use AWT and Swing here as AWT does not provide a range of native widgets. They both are also lacking several recitals which are needed. Now days, we are using standard widget toolkit, JFace instead of AWT and Swing due to the ease of development it provides.

2.6.5 Standard widget toolkit

IT provides application programming interface for widgets and graphics which are operating system independent. It is used to present the information to the user about Eclipse platform User Interface and all related tools of it. Standard widget toolkit implements a structure for modular, native widgets to provide tight integration, which is compulsory for good programming, with the native window system so that it could
provide the platform which application targets. Finally, in this way it provides a responsive, high quality user experience. For the ease of developer so that he can develop a single API for all platforms by using widgets which provides high performance, standard widget toolkit hides the difference of different platforms operating systems. In this way, developer can write his code once on any platform and this code can work with all the platforms. Developer can use standard widget toolkit over JFace for small, light-weight applications for example, mobile application. As we know that standard widget toolkit is a part of workbench, hence, it provides a high quality and responsive experience for the Java based client application which is the main feature of the platform.

![Properties for MyProject](image)

**Figure 2.4 Developing standard widget toolkit Applications on Eclipse**

Java AWT provides widgets which works at low level like button, list, radio button, check boxes, text fields, scroll bars, but it does not provide high level widgets like rich text or trees. As it can be directly implemented with the help of native widgets, for all the window system, we cannot get the opportunity for the implementation of high level widgets. So there is always an issue for widget toolkit design between portable native window system integration.
If a developer develops a user interface by the help of only AWT, it will be the least common denominator for resolving this issue some widgets like trees, tables, rich text are added in swing of all window operating systems, that’s why it provides look and feel feature to application.

Standard widget toolkit provides an Application Programming Interface which is common, operating system-independent for widgets and graphics. For the presentation of information to the user in effective way, the entire Eclipse platform user interface and some tools which can be plug into it uses standard widget toolkit. Standard widget toolkit is implemented in such a way so that it can allow tight integration with the window system which is used natively. It provides an infrastructure which is required by application as native, modular widgets which are consistent with the platform. If a developer wants to program for a single application programming interface for all platforms using a consistent set of high-performance widgets, it is possible for him with the help of standard widget toolkit as standard widget toolkit hides differences between platforms or operating systems. A low-level graphics library of Java Graphical user interface components with native implementations is provided by standard widget toolkit for supporting this kind of programming. Finally this results in a high-quality, responsive user experience. As part of the user interface Workbench, for giving the user a high-quality responsive experience with the client application based on java that feels native to the platform, standard widget toolkit plays a main role in it. This supports developers to write the code once and make it look like a Windows or Macintosh application user interface. For developing small, lightweight applications to a low-level application programming interface, developers might choose standard widget toolkit over JFace such as a mobile application. In this kind of application a tiny footprint is important.

As Java AWT provides only low-level widgets like buttons text fields, and lists, but it does not provide high-level widgets like rich text or trees. AWT widgets are supported to directly implementation with native widgets on all underlying window systems. There is always an issue with widget toolkit design between portable toolkits and native window system integration.
Building a user interface with the help of AWT means developing a project, which is least common denominator to all operating system. This issue is resolved by the Java Swing toolkit by emulating widgets such as rich text, trees and tables. Swing also provides look and feel simulation layers that make applications look like the underlying native window system. Rivalled widgets invariably lack the look and feel of native widgets. Users who are working with these widgets notice a large number of differences from their native widgets.

Standard widget toolkit defines a common Application user interface, available across a number of supported window systems to overcome this problem. Standard widget toolkit implements common low-level widgets such as buttons, text fields, lists everywhere using native widgets. Some generally useful higher-level widgets, however, may also need to be imitated on some window systems. Standard widget toolkit provides a suitable emulation, when a native widget is available on one platform but not another. Wherever possible, standard widget toolkit uses native widgets for each different native window system. This approach allows standard widget toolkit lets the look and feel of the underlying native window system show through as much as possible. Beside this this approach is also allows to maintain a consistent programming model across all environments.

Internally, for each native window system the standard widget toolkit implementation provides separate and distinct implementations in Java while keeping the application programming interface identical across all platforms. The Java native libraries are completely different for each platform, with a minimum amount of code with each tying the specific underlying window system to the common application programming interface. Tight integration with the underlying native is required within window system is not strictly a matter of look-and-feel. Standard widget toolkit exposes the native window system specific application programming interfaces in cases where a particular underlying native window system provides a significant and unique feature that cannot be matched on other window systems. For allowing the user to drag and drop content from standard widget toolkit widgets to the other native applications which are not based on standard widget toolkit, standard widget toolkit also interacts with native desktop features such as drag and drop. Windows ActiveX is an example
of it. To indicate that it is inherently non portable, the window system specific application programming interface is separated into appropriate named packages. The standard widget toolkit implementation is expressed entirely in Java code that looks familiar to the native operating system developer because these window system specific application programming interfaces are implemented with a minimum amount of code. Since standard widget toolkit consists of calls to the Windows application programming interface those they already known to the developer, windows programmers using C will find the Java implementation of standard widget toolkit for Windows ActiveX instantly familiar.

As this technology allows all interesting development to be done in Java, it greatly simplifies implementing, maintaining, and debugging standard widget toolkit. This is not a concern for ordinary standard widget toolkit clients because native widgets are hidden behind the window system independent standard widget toolkit Application Programming Interface.

2.6.6 JFace
There is some difference between standard widget toolkit and JFace as they solve different kinds of problems. For handling many common user interface programming tasks at a higher level, JFace is a Java user interface toolkit with classes.

![Eclipse Workbench](image)

*Figure 2.5 Eclipse’s JFace and standard widget toolkit*

JFace builds upon common yet simple application programming interface to provide an easy-to-use user interface abstraction layer whereas standard widget toolkit provides a common application programming interface across all platforms. Using JFace, programmers intermingle with their main objects instead of their more primitive underlying user interface presentation. JFace is window system independent
in both its Application Programming Interface and implementation. JFace also includes the usual user interface toolkit components of dialog, image and font registries, preference, progress reporting, wizard frameworks for long running operations. For example, to display a list of records using JFace, the implementation of an interface describes that what should be displayed there for a given record, and in this way it is easier to manipulate lists of records. If lists of strings represent those records then it is not quite so easy. It provides three interesting features of it; those are label providers, content providers, and viewers.

Label providers produce the special icons and string labels which are needed to display any given domain element in the widget whereas Content providers know how to translate domain changes into appropriate viewer updates and how to map the viewer’s input element to expected viewer content. In this way clients are described about events and selections in terms of the domain elements they provide to the viewer.

The main work of Viewers is to provide object-oriented approach as well as wrappers to their associated standard widget toolkit components. In this way it provides higher-level semantics than standard widget toolkit widgets. These viewers are configured with a label provider and a content provider and can optionally be configured with sorters and filters which are element based. The standard viewers for tables, trees, and lists keep the widgets in sync with changes to that domain and also support populating the viewer with elements from the client’s domain.

All the common operations such as text colouring, navigating by character index, double-click behaviour, undo or line number are supported by the standard viewer. The viewer implementation holds the mapping between standard widget toolkit widgets and domain elements. It also adjusts for a view of the elements which is filtered and re-sorting when necessary. A document model is provided by the Text viewers to the client and by this way, it manages the conversion of the document required by the standard widget toolkit styled text widget to the information.

If the user selects a particular element from a list of record information then to inform other user interface elements, internal notifications are generated so that they can
update their content based on the new selection. By implementing the appropriate interfaces, existing domain objects can be displayed in the Eclipse user interface. On the same model or document, multiple viewers can be open. These all viewers are updated automatically when the model or document changes in any of them.

2.6.7 Platform Runtime

Plug-ins is used for extension and they declare how other plug-ins can extend them and how they extend other plug-ins. The Platform Runtime uses of extensions and extension point’s technology to facilitate loosely coupled program modules (plug-ins). Using this information, minimum set of plug-ins necessary for a given task is determined by the Platform Runtime and only that plug-ins is loaded. All this work has been done dynamically and thus this improves the start-up speed and reduces the memory footprint of the application. For determining which plug-ins is available when an application is running, the Platform Runtime uses the OSGi framework. This information provides each plug-in with information about which plug-ins extend it and how, along with the plug-in extension and extension-point information. Developer can more easily recombine plug-ins in different ways without having to reinvent the wheel to solve different problems because this information is not hardcoded within each plug-in, but it has been discovered at time of execution, dynamically.
2.6.8 OSGi Framework

The OSGi framework focuses on the plug-in lifecycle. The OSGi framework is responsible for the installation, updating, or removal of the plug-ins on the fly without disrupting the operation of the device. It is a thin layer that allows multiple Java-based components so that they can cooperate in a single JVM (Java Virtual Machine). Its software components can dynamically discover and use other components which reside in libraries or applications.

Figure 2.7 OSGi Framework

OSGi makes available many standard component interfaces including common functions such as security, HTTP servers, user administration, configuration, logging and XML. Components are lazily loaded on an as-needed basis, reducing the overall memory footprint and start-up time of the application. Plug-in-compatible implementations of these components can provide different optimizations and can be obtained from different vendors. Following functionalities are added to the Eclipse RCP platform by OSGi:

1. To facilitate service discovery without loading and executing the bundle, a service registry provides a cooperation model for bundles.
2. Bundles add an application programming interface to manage the modules in runtime and they also rely on the module layer for class loading. A lifecycle layer that adds bundles that can be dynamically installed started, updated, stopped, and uninstalled.

3. The class-loading policies are defined by a powerful and rigidly specified Java-based class-loading layer.

2.6.9 Plug-Ins and Their Operation

Plug-ins is coded in Java. A plug-in can be developed and delivered separately, also known as a bundle is the smallest functional unit of the Eclipse platform that very small applications might be written as a single plug-in. But several plug-ins are needed for the complex applications to blend their functionality across the project. Plug-ins can provide support for manipulating and editing additional types of resources such as Word documents, Java files, JSP (Java Server pages) files, C programs and HTML pages. All of the Eclipse platform’s functionality is located in plug-ins except for the Platform Runtime. A typical plug-in contains Java code in a JAR (Java archive) library, a manifest file, some read-only files, and some other resources such as message cat-logs, images, native-code libraries, Web templates etc. However a plug-in may not contain code at all. The ultimate functionality of the platform or application is determined by Plug-ins. This is the reason that for enhancement of its functionality, Eclipse SDK ships with additional plug-ins. All the things which are needed for running the plug-in are located combine in a directory, for an internationalized plug-in this mechanism is used to deliver separate language packs. A plug-in to be synthesized from separate fragments, each in its own directory or URL uses another mechanism for the same permission. A plug-in that contributes online help in the form of HTML pages is one example. The plug-in read-only content and code libraries are located together in a directory in the file system or at a base URL on a server.
How a plugin interacts with the system is defined in the manifest file of that plug-in. It includes what services that plug-in provides and what services that plug-in consumes to other plug-ins. The information for uniquely identifying the plug-in and its version is also included here. Eclipse Platform Runtime loads only the plug-ins which is necessary for the current operation of the application on the basis of the declarative approach to modularization.

2.6.10 Challenges of Building Applications Using Eclipse RCP

Eclipse RCP is facing several technical issues like writing loosely coupled code including getting up to speed-

**Stick to the application programming interface**- Sometimes developers write code which needs to access internal code in other modules. If the same approach is also used for an Eclipse RCP application then the code will not be maintainable as the internal code changes if the Eclipse platform is involving.

**Getting up to speed**- Eclipse is a very well supported and well designed object-oriented framework. Explaining the application programming interface Javadoc well documented in code that how to use a particular function or feature. Eclipse represents
a very large application programming interface and therefore it is very useful for someone new to the environment. For providing a general understanding and direct developers where to dig for more details, many training materials, books, and online articles are available.

**Loosely coupled code**- Writing plug-ins promotes reusability and flexibility of the code as it declares how they interact with other plug-ins and this feature is a milestone of Eclipse-based development. Many developers are used to writing code interacting with other modules but they do not want to write a manifest declaring how their code interacts with other modules. Developers need to think in terms of how it extends or consumes “services” of other plug-ins and what “services” a plug-in provides.

**Installers**- Commercial third-party installers can handle the problem of deployment and application updates. Well this is not difficult when using Eclipse RCP. It doesn’t have an installer, deployment and application updates more difficult.

**Testing tools**- In development the availability of testing tools can also be an issue. There are few automated Graphical user interface testing tools available for Eclipse RCP applications. Fortunately, some third-party solutions are rising for this purpose. JUnit is a standard Java test framework provided by Eclipse for this work but it does not easily address user interface testing without additional functionality layered on top of it.

**Plug-in quality, support and maturity**- Many organizations are needed to enhance Eclipse RCP with functions and features that match and work their business requirements. For some, Eclipse as an application framework is rich enough that it doesn’t require other plug-ins to add capabilities. This means they will have to immerse into the Eclipse plug-in ecosystem also but doing so brings up several issues. A parallel approach can be found in the commercial software world where shareware and freeware are offered. But in this approach the quality, support and maturity of these products are obvious considerations.

**2.6.11 Commercial Third-Party Tools**
Commercial Third-Party tools help with not only in user interface testing, creating help documentation but also, the Graphical user interface design and coding, and
packaging and deployment of the application. For developers who choose to create a rich client for their development and applications, there are some commercial tools provided, such as RCP Developer. Users can keep up to date with the latest versions of their applications with update manager in Eclipse.

Eclipse RCP provides an open source extensible application framework, and creates applications that can run on a variety of platforms but still it will have serious inference for the desktop computing strategies of enterprise organizations because it incorporates application development tools that are easy to use. So, many organizations are standardizing on RCP as the architecture for their internal IT applications, thus enabling them to run as an integrated, manageable suite rather than as disassociated units of functionality.

2.7 LITERATURE REVIEW OF SOME PROJECTS

2.7.1 Eclipse in starting

IBM released Eclipse 1.0 under the Common Public License (CPL) in 2001 then Eclipse was born. The current version is 4.2.2. A brief list of the most important phases of the history and future Eclipse is provided here-

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eclipse 1.0 release</td>
<td>November 2001</td>
</tr>
<tr>
<td>2</td>
<td>Eclipse 2.0 release</td>
<td>July 2002</td>
</tr>
<tr>
<td>3</td>
<td>Eclipse 3.0 release</td>
<td>June 2004</td>
</tr>
<tr>
<td>4</td>
<td>license switch from CPL to EPL (Eclipse Public License) complete</td>
<td>January 2005</td>
</tr>
<tr>
<td>5</td>
<td>Callisto simultaneous release, including Eclipse 3.2</td>
<td>June 2006</td>
</tr>
<tr>
<td>6</td>
<td>Europa simultaneous release, including Eclipse 3.3</td>
<td>June 2007</td>
</tr>
</tbody>
</table>
Table 2.1 Most Important Phases of Eclipse

<table>
<thead>
<tr>
<th>No.</th>
<th>Phase</th>
<th>Release Details</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Ganymede</td>
<td>simultaneous release, including Eclipse 3.4</td>
<td>June 2008</td>
</tr>
<tr>
<td>8</td>
<td>Galileo</td>
<td>simultaneous release, including Eclipse 3.5</td>
<td>June 2009</td>
</tr>
<tr>
<td>9</td>
<td>Helios</td>
<td>simultaneous release, including Eclipse 3.6</td>
<td>June 2010</td>
</tr>
<tr>
<td>10</td>
<td>Indigo</td>
<td>simultaneous release, including Eclipse 3.7</td>
<td>June 2011</td>
</tr>
<tr>
<td>11</td>
<td>Juno</td>
<td>simultaneous release, including Eclipse 4.2</td>
<td>June 2012</td>
</tr>
<tr>
<td>12</td>
<td>Kepler</td>
<td>simultaneous release, including Eclipse 4.3</td>
<td>June 2013</td>
</tr>
</tbody>
</table>

Eclipse was cited as the most popular professional Java IDE already in 2005, and from the beginning its growth continues. In 2006, Evans Data Corporation calculates approximately that the use of the Eclipse RCP get triple within two years. That was a great achievement for Eclipse. Eclipse is an open source success story which is moving forward continuously to the spotlight even faster than projects such as Mozilla and OpenOffice.

A survey of Eclipse.org community is conducted by IDC, in conjunction with the Eclipse Foundation, in order to better understand by participants. The main purpose was to observe-

1. The characteristics of this software.
2. The usages of Eclipse open source project software.
3. From their usage of Eclipse the business models of companies looking to make money.
Summary Observation which they find here are:

Eclipse community is “business-oriented” is said by-

1. 91% persons who are self-employed or employed by an organization.
2. 84% persons from respondents who use Eclipse for work-related reasons.
3. 71% persons of employed by organizations or IT solution providers.
4. 64% persons of unique organizations who are respondents use Eclipse to make money or to save money.
5. 29% end users.

In this observation over half i.e. 56% respondents are experienced with Eclipse more than 3 years.

Top 5 most used Eclipse projects are-

<table>
<thead>
<tr>
<th>88% of respondents</th>
<th>Java Development Tools (JDT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>54% of respondents</td>
<td>Web Tools Project, Web Standard Tools</td>
</tr>
<tr>
<td>44% of respondents</td>
<td>J2EE Standard Tools, Web Tools Project</td>
</tr>
<tr>
<td>42% of respondents</td>
<td>Eclipse Rich Client Platform (RCP)</td>
</tr>
<tr>
<td>37% of respondents</td>
<td>Eclipse Modelling Framework (EMF)</td>
</tr>
</tbody>
</table>

Table 2.2 Top 5 Projects of Eclipse

A number of applications have been designed using Rich Client Platform; three of them are described here-

2.7.2 Brain Box
BrainBox is a Web Services compatible tool which provides asynchronous simulations. BrainBox uses J2EE based server to provide this service. BrainBox is a simulation toolbox, similar to MatLab Simulink. It simulates mathematical and physics models based on description provided by a block diagram. It also a lot of extensions points for developers and provides a build-in JavaScript language to create their own customized toolboxes. Besides all these things, BrainBox provides a complete user interface which helps the user to interact with the simulation results and to create the block diagrams. More information about BrainBox can be getting from http://brainbox.sourceforge.net/en/index.

2.7.3jFire
The framework of jFire provides the common base for developers so that he can fulfil the need of an ERP system specialized for their business and it is designed in a way that it is highly customizable. All common tasks which are aimed by an ERP system it manages, like customer relationship management and accounting and the distribution of products. JFire is a flexible Open Source ERP framework based Eclipse RCP. Some main features of the JFire RCP client are:

- a classsharing plugin which allows the user to remote class loading from the JFire server,

- Provides integration with many Eclipse projects like GEF and BIRT or a Workbench-wide extendable exception handler for developer’s ease.

This comprises organisation and user management basic trading, product definition, and many more. For providing most basic features, graphical user interface and client side objects are included in the core system already. They can be customized or reused via extension-points in order to build new applications tailored to many kinds of businesses.
2.7.4 jLibrary -

jLibrary is developed with Eclipse Rich Client platform and it is an Open Source CMS system. jLibrary provides a rich set of features like native editor’s support, metadata, remote and local repositories etc. The main work jLibrary is to handle graphical relation browser, syntax highlighting, web spider to download the preferred web pages, categories, favourites with the Open Office, MS Word, acrobat, HTML/XML editor etc. Windows and Linux platforms are supported by jFire and currently the web is in Spanish, but the program is internationalized to support English and Spanish.

2.8 GOOLE MAPS

I have made Earth Navigator in the Eclipse using Rich Client Platform. This kind of facility is already provided by Google Maps. I will describe these differences in the 4th chapter. Google Maps is a web mapping service application and technology provided by Google, that powers many map-based services, including the Google Maps website, Google Ride Finder, Google Transit, and maps embedded on third-party websites via the Google Maps API. It offers street maps and a route planner for
traveling by foot, car, bike (beta), or with public transportation. It also includes a locator for urban businesses in numerous countries around the world. Google Maps satellite images are not updated in real time, however, Google adds data to their Primary Database on a regular basis and most of the images are no more than 3 years old.

Google Maps uses a close variant of the Mercator projection, so it cannot accurately show areas around the poles. A related product is Google Earth, a stand-alone program which offers more globe-viewing features, including showing polar areas.

Google Maps provides Satellite view. Most of the world's available satellite imagery are no more than 3 years old and updated on a regular basis. Google Maps provides high-resolution aerial or satellite images for most urban areas of the world. However, all areas of the satellite imagery do not appear in the same resolution; less populated areas usually appear with less detail.

Google's mapping engine prompted a surge of interest in satellite imagery, due to it's easily pawn able and searchable mapping and satellite imagery tool. Websites were created to feature satellite images of interesting natural and human-made landmarks, including such novelties as large writing visible in the imagery, famous stadiums, and unique geological formations. Although Google uses the word satellite, most of the high-resolution imagery of cities is aerial photography taken from aircraft flying at 800 feet (240 m) to 1,500 feet (460 m); however, most of the other imagery is from satellites. Google Aerial View is provided in some urban areas.

In 2005 the Australian Nuclear Science and Technology Organization (ANSTO) complained about the potential for terrorists to use the satellite images in planning attacks, with specific reference to the Lucas Heights nuclear reactor; however, the Federal Australian government did not support the organization's concern. At the time of the ANSTO complaint, Google had colored over some areas for security (mostly in the US), such as the rooftop of the White House and several other Washington, D.C., US buildings.

Google Maps provides a route planner under "Get Directions". Up to four modes of transportation are available depending on the area: driving, public transit, walking,
and bicycling. In combination with Google Street View, issues like parking, turning lanes and one-way streets can be viewed before traveling. Driving directions are covered as follows:

Most countries of mainland Eurasia and Africa are covered contiguously, including the United Kingdom, Ireland, the Canary Islands, Cyprus, Malta, Sri Lanka, most of Indonesia and Timor-Leste. China mainland, Hong Kong, Macau, Israel (including parts of the West Bank), Jordan, Lebanon and North Korea have directions available without connection to other states. Only public transit directions are provided for South Korea. All countries of mainland North and Central America are covered contiguously. All countries of mainland South America are covered. Argentina, Bolivia, Brazil, Chile, Ecuador, Paraguay, Peru, Trinidad and Tobago, Uruguay and Venezuela are treated contiguously, whereas Colombia, French Guiana, Guyana and Suriname are not connected to other states.

All inhabited countries and territories in the Caribbean are covered, though in general there are no connections between islands. Additionally, American Samoa, Australia, the Azores, Brunei, Cape Verde, The Comoros, The Cook Islands, the Faroe Islands, The Federated States of Micronesia, Fiji, French Polynesia, Guam, Hawaii, Iceland, Japan, Madagascar, the Maldives, Mauritius, Mayotte, New Caledonia, New Zealand, Niue, Northern Mariana Islands, Palau, the Philippines, Réunion, São Tomé and Príncipe, the Seychelles, Samoa, Taiwan, Tonga, Vanuatu, Wallis and Futuna are covered as stand-alone regions, as are Nuuk in Greenland, Sabah in Malaysia, parts of Papua New Guinea, parts of Solomon Islands and Socotra in Yemen.

2.8.1 Implementation of Google Maps
Like many other Google web applications, Google Maps uses JavaScript extensively. As the user drags the map, the grid squares are downloaded from the server and inserted into the page. When a user searches for a business, the results are downloaded in the background for insertion into the side panel and map; the page is not reloaded. Locations are drawn dynamically by positioning a red pin on top of the map images. A hidden IFrame with form submission is used because it preserves browser history. The site also uses JSON for data transfer rather than XML, for performance reasons. These techniques both fall under the broad Ajax umbrella. The
result is termed a slippy map and is implemented elsewhere in projects like OpenLayers. In October 2011, Google announced MapsGL, a WebGL version of Maps with better renderings and smoother transitions.

2.8.2 Extensibility and customization

As Google Maps is coded almost entirely in JavaScript and XML, some end users have reverse-engineered the tool and produced client-side scripts and server-side hooks which allowed a user or website to introduce expanded or customized features into the Google Maps interface.

Using the core engine and the map/satellite images hosted by Google, such tools can introduce custom location icons, location coordinates and metadata, and even custom map image sources into the Google Maps interface. The script-insertion tool Grease monkey provides a large number of client-side scripts to customize Google Maps data.

Combinations with photo sharing websites, such as Flicker, are used to create memory maps. Using copies of the Keyhole satellite photos, users have taken advantage of image annotation features to provide personal histories and information regarding particular points of the area.

2.8.3 Google Maps API

After the success of reverse-engineered mash ups Google launched the Google Maps API in June 2005 to allow developers to integrate Google Maps into their websites. It is a free service, and currently does not contain ads, but Google states in their terms of use that they reserve the right to display ads in the future.

By using the Google Maps API, it is possible to embed Google Maps site into an external website, on to which site specific data can be overlaid. Although initially only a JavaScript API, the Maps API was expanded to include an API for Adobe Flash applications but this has been deprecated, a service for retrieving static map images, and web services for performing geocoding, generating driving directions, and obtaining elevation profiles. Over 1,000,000 web sites use the Google Maps API, making it the most heavily used web application development API.
The Google Maps API is free for commercial use, provided that the site on which it is being used is publicly accessible and does not charge for access, and is not generating more than 25,000 map accesses a day. Sites that do not meet these requirements can purchase the Google Maps API for Business.

The success of the Google Maps API has spawned a number of competing alternatives, including the Yahoo! Maps API, Bing Maps Platform, MapQuest Development Platform, and OpenLayers.

In September 2011, Google announced it would discontinue a number of its products, including Google Maps API for Flash.

**2.8.4 Google Street View**

On May 25, 2007, Google released Street View, a new feature of Google Maps which provides 360° panoramic street-level views of various locations. On the date of release, the feature only included five cities in the U.S. It has since expanded to thousands of locations around the world.

In July 2009, Google began mapping college campuses and surrounding paths and trails. Street View garnered much controversy after its release because of privacy concerns about the uncensored nature of the panoramic photographs. Since then, Google has begun blurring faces through automatic face detection.